UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Creación de biblioteca WiFi ESP para microcontroladores PIC

TESIS

que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A N

Mario Iván Álvarez Bautista

Guillermo Seseña Pascasio

DIRECTOR DE TESIS

M.I. Ulises Martín Peñuelas Rivas

Ciudad Universitaria, Cd. Mx., 2020

Dedicatoria

A mis padres, Maricela y Mario, por su apoyo, cariño y enseñanzas que me han permitido llegar hasta este momento, así como impulsarme día a día a perseguir aquello que me apasiona y me hace feliz.
Mario

A mis padres, Francisco y Martha, por todo su amor, comprensión, confianza y apoyo en cada momento de mi vida; por el esfuerzo y sacrificio que realizaron para que yo cuente con una formación académica.

A mi hermana Carmen, por compartir tantas aventuras y desventuras.

A mi amigo Vladimir, que después de tanto tiempo sigue apoyándome.
Guillermo

Agradecimientos

A la Universidad Nacional Autónoma de México y la Facultad de Ingeniería,
por la formación profesional y personal que nos ha brindado.

A la DGAPA, por el apoyo otorgado para la realización de este trabajo,
a través del proyecto UNAM-DGAPA-PAPIME PE115419:
"Desarrollo de herramientas didácticas para la asignatura Circuitos Digitales"

Al M.I. Ulises Martín Peñuelas Rivas, por su paciencia y orientación
en este trabajo y para con nosotros.

Contenido

Resumen

Para proporcionar una opción para el desarrollo de prototipos que demandan las nuevas tendencias tecnológicas como el internet de las cosas, se diseñó la biblioteca ESP que permite a los microcontroladores PIC acceder a comunicación inalámbrica y a servicios de internet mediante un módulo WiFi que incorpore el SoC ESP8266 o los ESP32. La conjunción de estos elementos representa una opción flexible con relación al hardware, respecto a la variedad de tarjetas de desarrollo que actualmente se ofrecen, permitiendo que se adecúe mejor a las necesidades de los diseños, favoreciendo el uso eficiente de recursos y la redución de costos. La biblioteca ESP otorga hasta 28 aplicaciones, las cuales son el resultado de la elaboración de software tanto en el módulo a través de nuevos comandos AT, como en el microcontrolador PIC mediante la parcial adaptación de algunos de los protocolos de aplicación más populares de internet.

Se elaboró un manual de prácticas que podrá servir como material didáctico en la asignatura Circuitos Digitales de la carrera de Ingeniería Mecatrónica, el cual ilustra el uso de la biblioteca y a su vez muestra los alcances y limitaciones de cada una de las aplicaciones desarrolladas.

Capítulo 1

Introducción

1.1 Panorama

Los avances tecnológicos en comunicaciones inalámbricas han permitido que toda clase de dispositivos intercambien información a través de redes computacionales, incluyendo a la red más grande del mundo llamada internet. La tendencia hacia un mundo "hiperconectado" es cada vez más real, el surgimiento de nuevas tendencias tecnológicas como el internet de las cosas, o IoT por sus siglas en inglés *internet of Things,* y la esperada cuarta revolución industrial (*Industry 4.0*) están demandando nuevos productos que requieren de un enfoque aún mecatrónico, pero que a su vez demandan la inclusión de nuevas tecnologías de la información y la comunicación (TIC), áreas que eran ajenas en el campo de la mecatrónica y que plantean nuevos retos.

El *internet of Things* ha comenzado a demandar dispositivos que se caracterizan principalmente por conexión a internet a bajo costo, un bajo consumo energético y un reducido tamaño. Las primeras soluciones del *internet of things* están implementando comunicaciones inalámbricas como WiFi, Zigbee, Bluetooth y Sigfox, de las cuales destaca la tecnología WiFi por su presencia en el ámbito global, la interoperabilidad entre generaciones de esta tecnología y su bajo costo. Aunque la tecnología WiFi no satisface aspectos como bajo consumo energético o seguridad que solicita el IoT, sus fortalezas inherentes han hecho que se considere como una de las alternativas de comunicación más populares en la actualidad para esta clase de dispositivos.

1.2 Planteamiento del problema

Ante la necesidad de diseñar productos que se adapten a las características que solicitan las nuevas tendencias tecnológicas como IoT, las herramientas de aprendizaje para las áreas de conocimiento del STEM (*Science, Tecnology, Engineering and Mathemathics,* es decir, Ciencia, Tecnología, Ingeniería y Matemáticas) han adquirido relevancia en educación basada en la elaboración de proyectos que permite la asimilación de conocimientos requeridos en el diseño de estos nuevos productos. Una de las herramientas del STEM que está gozando de una amplia popularidad son las tarjetas de desarrollo que incluyen tecnologías inalámbricas como WiFi. Estas tarjetas ofrecen el *hardware* necesario para el diseño de prototipos rápidos, sin embargo, ofrecen muy pocas variantes respecto al *hardware* que las constituyen por lo que no se pueden adaptar de manera eficiente a las necesidades de un diseño riguroso.

En este trabajo se presenta una opción que ofrezca flexibilidad respecto al *hardware* en el desarrollo de prototipos IoT, por lo que se propone el uso de microcontroladores PIC y módulos ESP mediante una biblioteca que permita comunicar ambos componentes. Para la aplicación de esta biblioteca se ha desarrollado un manual de prácticas que permitirá ilustrar los alcances y limitaciones de este conjunto, así como también su uso.

1.3 Objetivo

Desarrollar una biblioteca para microcontroladores PIC que permitan utilizar a éstos en conjunto con un módulo WiFi, como una alternativa para el desarrollo de prototipos mecatrónicos que requieran de comunicación inalámbrica o acceso a internet.

1.4 Objetivos específicos

- Aprovechar la mayoría de las características que ofrezcan tanto los uC PIC como el módulo WiFi que sean empleados.
- Determinar las características que se requieren de los módulos WiFi y los microcontroladores PIC para que puedan ser empleados en conjunto, buscando que éste sea flexible respecto a *hardware* para adaptase a diversas necesidades de los prototipos.
- Ilustrar las aplicaciones, alcances y limitaciones del conjunto uC PIC-módulo WiFi.
- Desarrollar material didáctico para la asignatura Circuitos Digitales.

1.5 Resumen de la tesis por capítulo

La presente tesis está constituida por cinco capítulos y cinco apéndices. El primer capítulo presenta una introducción general del trabajo, donde se muestra el estado de tecnología para otorgar el contexto en que se ubica, así como los objetivos que fueron planteados.

En el segundo capítulo se detalla la propuesta de trabajo, indicando los requerimientos e implicaciones iniciales de ésta, asimismo se describe las características de los elementos utilizados y se explica el *hardware* mínimo de la propuesta.

En el tercer capítulo se desglosa las generalidades del diseño tanto del *software* directo como del indirecto, mostrando la estructura y funcionamiento interno de éstos, además se señalan las limitaciones del propio diseño.

En el cuarto capítulo se muestra los resultados generales del trabajo e incluye el manual de prácticas, elaborado para utilizarlo en la asignatura Circuitos Digitales. En el manual se plantea 51 actividades agrupadas en 30 prácticas, en cada una de ellas se introduce conocimientos generales necesarios para comprender dicha práctica, indicando tanto el *hardware* como *software* requerido, asimismo, se señala con mayor detalle los alcances y limitaciones de cada una de las 28 aplicaciones.

En el último capítulo se presentan las conclusiones del trabajo, indicando las limitaciones y posibles mejoras que se pueden efectuar en un futuro.

Respecto a los apéndices, en el primero se encuentra la documentación que explica cómo realizar la actualización del *firmware* del módulo que requiere la propuesta. En el segundo apéndice se describe el proceso para crear nuevos comandos AT en el módulo y su incorporación en el *firmware*. En el tercer apéndice se detallan las modificaciones realizadas en algunos de los archivos del *firmware* para obtener acceso a ciertas funciones de los módulos ESP. El cuarto apéndice da una descripción de cada una de las funciones desarrolladas en la biblioteca ESP. Finalmente en el quinto apéndice se da una descripción de cada uno de los nuevos comandos AT desarrollados para la biblioteca ESP.

Capítulo 2

Estado de la tecnología

2.1 Comunicaciones inalámbricas y nuevas tendencias tecnológicas

2.1.1 Comunicación inalámbrica

La comunicación inalámbrica es aquel tipo de comunicación en la cual la información se transmite a través del aire, para ello, los dispositivos cuentan con antenas que emiten o reciben información mediante ondas electromagnéticas. Estas antenas pueden ser direccionales u omnidireccionales, las primeras emiten ondas en una sola dirección y por lo tanto, requieren que la antena del emisor y la del receptor estén alineadas, mientras que las segundas emiten ondas en todas direcciones [1, 2].

Esta emisión de ondas electromagnéticas se da en diferentes frecuencias del espectro electromagnético, cubriendo principalmente tres tipos de radiación:

- Radiofrecuencias, consideradas comúnmente como las ondas electromagnéticas que operan a una frecuencia de entre 3 *kHz* y 1 *GHz*. Este tipo de ondas pueden penetrar muros, sin embargo, son muy propensas a interferencias y para su propagación utilizan antenas omnidireccionales. Usualmente son empleadas para cubrir grandes distancias y para multidifusión como: radio y televisión.
- Microondas, son ondas electromagnéticas que operan a una frecuencia de entre 100 *MHz* y 300 *GHz*, en las cuales se emplean antenas direccionales para su emisión, aunque a bajas frecuencias se pueden emplear también antenas

omnidireccionales, tal como es el caso de las antenas WiFi. Este tipo de onda no llega a cubrir grandes distancias, aunque, se pueden emplear repetidores para aumentar su alcance. Las microondas son empleadas en telefonía móvil, comunicación satelital y redes locales.

- Infrarrojas, son ondas electromagnéticas que operan a una frecuencia de entre 300 *GHz* y 400 *THz*, que para su propagación utilizan antenas direccionales. Este tipo de ondas no son capaces de penetrar objetos y su uso está limitado a interiores debido a interferencias causadas por la exposición al sol. Las radiaciones infrarrojas son utilizadas para comunicación de corto alcance, como se da en el control remoto de un televisor y periféricos de computadoras [1, 2].

En las últimas décadas, la comunicación inalámbrica ha tenido un gran éxito, principalmente por su uso en telefonía y redes de computadoras, ámbitos que han revolucionado la forma en que se interactúa con el mundo y en la que se accede a la información [3]. Sin embargo, la aceptación de este tipo de comunicación ha sido paulatina, ya que en un principio la comunicación inalámbrica tenía un rendimiento pobre en comparación con la alámbrica, además de la falta de estandarización y la gran diversificación de tecnologías.

El origen de las comunicaciones inalámbricas se remonta a 1897, cuando Marconi realizó la primera trasmisión de radio en la Isla de Wight. Después de ello la evolución de este tipo de comunicaciones ha seguido diferentes caminos. Enfocándose únicamente en aquellas relacionadas con redes de dispositivos computacionales, se encuentra por un lado la telefonía inalámbrica o móvil, que tiene origen en Estados Unidos de América (EUA) en 1915 cuando se logró la primera transmisión de voz inalámbrica entre Nueva York y San Francisco, posteriormente en el año 1946 EUA estableció el servicio público de telefonía móvil en 25 de sus ciudades, sin embargo, cada una de éstas tenían un solo trasmisor que limitaba el número de usuarios. Esta limitante fue resuelta por los investigadores de *AT&T Bell Laboratories*, los cuales desarrollaron el concepto de telefonía celular, en la que el área de cobertura está divida en “celdas” no superpuestas de trasmisores [4].

Para esta primera generación de telefonía celular se trasmitía voz de manera analógica. A principios de la década de 1990, la segunda generación de telefonía celular sustituyó al servicio de voz analógica por el servicio de voz digital, posteriormente la telefonía fue desarrollando nuevas generaciones que proporcionaron servicios como mensajería instantánea, correo electrónico y acceso a internet, dejando de ser únicamente una red de telefonía para convertirse en una red de dispositivos computacionales [4].

En el caso de las redes computacionales como tal, su origen se remonta a 1971 cuando la Universidad de Hawái desarrolló la red ALOHAnet, la cual comunicaba a siete computadoras esparcidas en cuatro islas con una computadora central, cuya transmisión era realizada por medio de radiofrecuencias que enviaban paquetes de señales digitales. Posteriormente, el ejército de EUA mostró interés por este tipo de red y a través de la Agencia de Proyectos de Investigación Avanzada en Defensa (*Defense Advanced Research Projects Agency*, DARPA) invirtió una gran cantidad de recursos para implementarlas en tácticas de combate, pero los resultados no fueron los esperados, puesto que las redes tenían un desempeño pobre. Del mismo modo, el desarrollo de este tipo de redes en la década de 1990 para uso comercial no tuvo éxito debido a sus velocidades bajas y costos altos, por lo que fueron sustituidas por telefonía móvil y por las emergentes redes inalámbricas de área local [4].

Las redes inalámbricas de área local, es decir, redes que tienen un alcance que puede cubrir hogares, edificios y escuelas, surgieron luego de que la Comisión Federal de Comunicaciones (*Federal Communications Commission*, FCC) de EUA permitiera el uso público de las bandas ISM, siglas en ingles de *Industrial, Scientific and Medical*, bandas que habían sido reservadas para equipos que generaban o utilizaban radiofrecuencias para fines industriales, científicos y médicos. Sin embargo, las primeras redes tuvieron una lenta aceptación, ya que eran costosas con velocidades de transferencia bajas, en comparación con las redes de área local cableadas como el Ethernet; además, estas redes inalámbricas contaban con poca seguridad y no estaban estandarizadas.

Para finales de la década de 1990, el uso de redes de área local inalámbrica se convirtió en una necesidad en zonas de trabajo y hogares, pues se popularizó el uso de computadoras portátiles, sin embargo, para acceder a internet estas aún estaban atadas a los puertos Ethernet, era necesario una red inalámbrica que permitiera comunicar a estos equipos con las conexiones fijas (*router,* enrutador o rúter). Ante esto, surgieron dos estándares que competían por satisfacer dicha necesidad, por un lado, se encontraba la Red de Área Local de Alto Desempeño (*High Performance Local Area Network*-HIPERLAN) desarrollada por el Instituto Europeo de Estándares en Telecomunicaciones (*European Telecommunications Standards Institute*, ETSI) y por el otro estaba el grupo de estándares 802.11 desarrollado por Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers*, IEEE), imponiéndose eventualmente este último, que sería la base para el origen de la marca WiFi [3].

Posteriormente, el IEEE desarrolló otros grupos de trabajo para estandarizar redes con características diferentes, permitiendo así establecer una clasificación de las redes inalámbricas, la cual es comúnmente descrita conforme al alcance que cada una de éstas presenta (ver figura 1.1):

- Redes inalámbricas de área personal (*Wireless Personal Area Networks*, *WPAN*), son redes que tienen un alcance de hasta 10 metros, se caracterizan por bajo consumo energético, bajo costo y bajas velocidades de transmisión de datos.
- Redes inalámbricas de área local (*Wireless Local Area Networks*, *WLAN*) tienen un alcance frecuentemente de más de 10 metros y menos de 100 metros, son una alternativa a las redes alámbricas conocidas como Redes de Área Local (*Local Area Networks*, LAN), las cuales son utilizadas en el hogar, escuela, oficina e industria.
- Redes inalámbricas de área metropolitana (*Wireless Metropolitan Area Networks*, *WMAN*) son redes de largo alcance que cubren hasta 50 *km*.
- Redes inalámbricas de área amplia (*Wireless Wide Area Networks*, *WWAN*) son redes de muy largo alcance que cubren grandes extensiones de la superficie terrestre de 50 *km* [5].

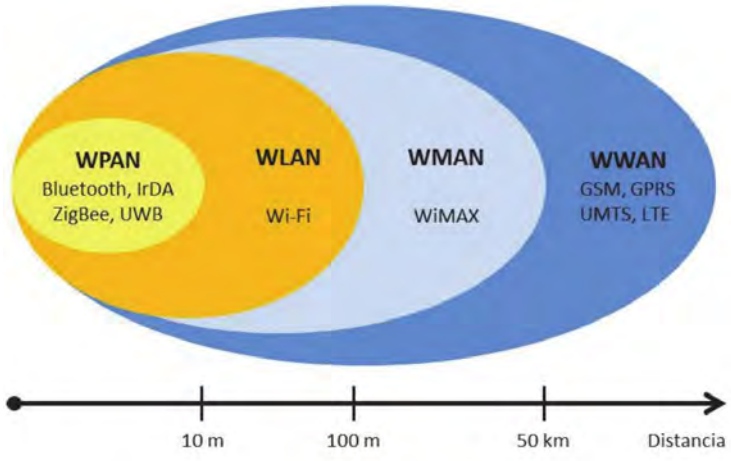


Figura 1.1 *Clasificación de las redes inalámbricas[5].*

La proliferación de las redes inalámbricas es en gran medida consecuencia del mejoramiento del rendimiento de la comunicación inalámbrica junto al decremento de su costo, aunado a una mayor movilidad y rápida implementación que ofrece, desplazando de manera significativa a la comunicación cableada, principalmente en el acceso a internet, gracias a tecnologías como telefonía móvil y WiFi.

De hecho, se estima que para el 2021 los dispositivos con conexión alámbrica representarán tan solo el 27% del tráfico de internet, mientras que el 73% será para dispositivos móviles y dispositivos con WiFi [6] (ver figura 1.2).

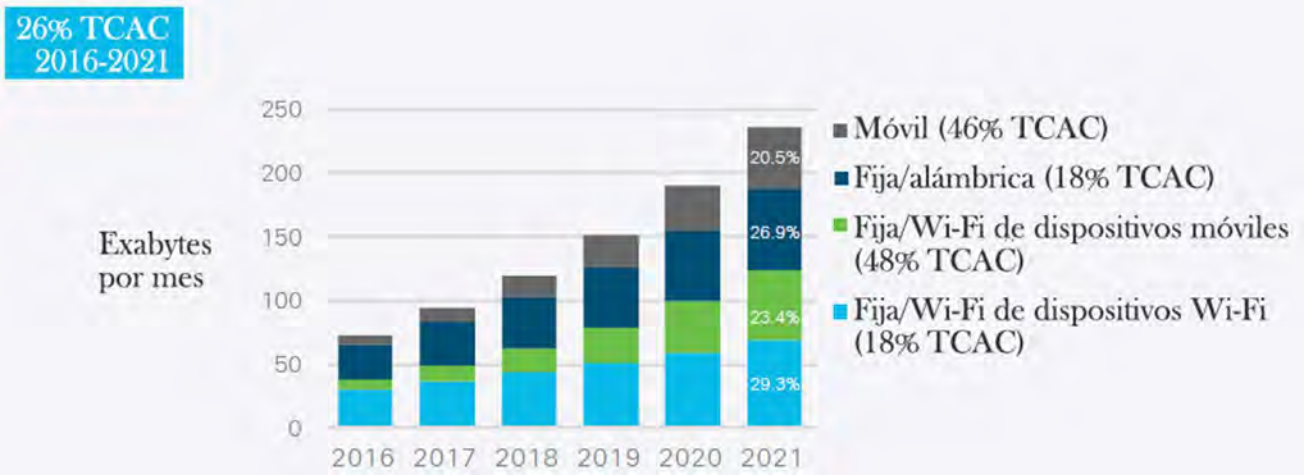


Figura 1.2 *Tráfico mundial de internet, cableado e inalámbrico (adaptado de [6]).*

2.1.2 Nuevas corrientes tecnológicas

Características como un bajo consumo energético, bajo costo y transferencia de datos a altas velocidades que ofrecen las actuales tecnologías inalámbricas, aunado a la miniaturización de las tecnologías de cómputo, han permitido que toda clase de dispositivos puedan ser conectados a internet. Esto ha generado una tendencia hacia un "mundo hiperconectado", dando lugar a nuevas corrientes tecnológicas como *internet of Things* e *Industry 4.0* [7].

Internet of Things (IoT) es una tendencia tecnológica que ha ganado una notable popularidad en los últimos años. La idea detrás de esta tendencia consiste en conectar objetos comunes a internet, con la finalidad de mejorar la experiencia del usuario con dichos objetos y con el entorno, permitiendo así ofrecer nuevos servicios con una reducida intervención humana [8]. Estos objetos generalmente están dotados con sensores y sistemas de comunicación, que emiten información a internet para su almacenamiento y análisis en plataformas de procesamiento de datos (ver figura 1.3), permitiendo a los usuarios conocer el estado actual de su entorno, haciendo más simple o incluso automática la administración de los recursos de éste [9].

Figura 1.3 *Estructura general del internet of Things (adaptado de [9]).*

IoT tiene un futuro prometedor; proyecciones de Huawei anticipan que para el año 2025 existirán aproximadamente 100 mil millones de conexiones IoT y tendrá un impacto financiero en la economía mundial de entre 3.9 y 11.1 mil millones de dólares, de acuerdo con Instituto Global McKinsey [7]. También tendrá un impacto en el desarrollo de aplicaciones dentro de una gama amplia de sectores:

- Edificios y casas, control de consumo energético y sistemas de seguridad
- Ciudades, optimización de administración de infraestructuras inteligentes (transporte, energía, agua, etc.), monitoreo ambiental y gestión de recursos
- Salud, monitoreo de salud de personas y mejoramiento en el manejo de enfermedades
- Comercio, sistemas de autopago y optimización de inventarios
- Agricultura, monitoreo de cultivo, ganado y medio ambiente
- Industria, mejora de eficiencia operativa y optimización del uso de recursos [7, 9].

El desarrollo de aplicaciones IoT que están destinadas a la industria, pertenecen a un subgrupo especial llamado *Industrial internet of Things (IIoT)*, es decir, internet de las cosas industrial; este subgrupo está orientado específicamente a las necesidades de producción inteligente. El objetivo del IIoT no es automatizar los procesos industriales, sino la obtención de información y análisis de ésta para comprender mejor estos procesos,

permitiendo así optimizar la producción y obtener una mejor adaptabilidad a los cambios en la demanda. La información se obtiene realizando conexiones máquina a máquina, entre todos, o una cierta cantidad de elementos que componen a una planta industrial (máquinas, sistemas de control, sistemas de información, entre otros) y es enviada a internet para su análisis o incluso para monitorear ciertos procesos [8].

A diferencia del resto de aplicaciones de IoT, el IIoT exige requerimientos más robustos como comunicación fiable, alto nivel de seguridad y latencias pequeñas y presenta un manejo de información más complejo, ya que tiene que procesar una inmensa cantidad de datos (de varios terabytes por minuto). Sin embargo, al igual que otras aplicaciones del IoT comparten características como conexión a internet a bajo costo, *hardware* limitado y escalabilidad de la red [8].

El modelo del IIoT está cimentando una nueva revolución industrial conocida como *Industry 4.0,* la cual también tiene como base los Sistemas Físicos Cibernéticos (*Cyber-Physical Systems*, CPS) (ver figura 1.4), que son sistemas computacionales dinámicos que permiten obtener el modelo virtual de una unidad física, controlarla y simular su comportamiento a través de la obtención de datos en tiempo real [10]. Esta nueva revolución industrial buscará crear fábricas inteligentes, que sean capaces de adaptar sus procesos de producción en tiempo real a las necesidades del mercado, esto gracias a la recolección de información de las plantas que propone el IIoT, más la información generada por los clientes y la automatización que implica la integración de los CPS [8, 11].

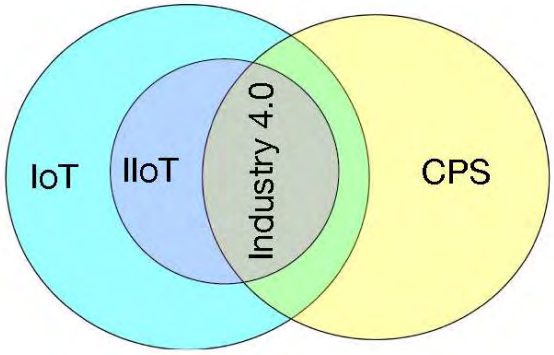


Figura 1.4 *Composición de la Industry 4.0 [8].*

Sin embargo, para que esta revolución industrial pueda cumplir su cometido, tendrá que lidiar con muchos obstáculos respecto a la interconexión de los sistemas, ya que las comunicaciones industriales exigirán características como: latencia de menos de 5 milisegundos, una densidad de al menos 100 dispositivos en un metro cuadrado, y niveles altos tanto de seguridad como de confiabilidad [12].

En la actualidad, no existe estándar de comunicación que pueda satisfacer las diferentes exigencias de todos los procesos industriales, ya que hasta ahora los requerimientos de las comunicaciones industriales no habían sido tomados en cuenta por los desarrolladores de estándares. Esto ha provocado que existan múltiples soluciones de automatización que en su mayoría son cableadas, provocando incompatibilidades entre los sistemas, problema que se heredará y agudizará durante *Industry* 4.0, puesto que es poco probable que los sistemas actuales sean reemplazados rápidamente por sistemas que incorporen nuevas tecnologías[12]. Lo anterior plantea el desarrollo y uso de redes heterogéneas, que implementen múltiples tecnologías de comunicación y que sean armonizadas a través de puertas de enlace y *middlewares*, lo cual hará que las redes industriales sean más complejas [12, 13].

Hoy día también se tiene una gran expectativa por la próxima generación de telefonía celular (5G), que promete comunicar de manera generalizada a los dispositivos de las nuevas tendencias tecnológicas y formar la base de una red heterogénea que integrará otras tecnologías inalámbricas y alámbricas [8, 11]. Aunque la 5G pueda impulsar aún más las tendencias tecnológicas mencionadas, es poco probable que satisfaga por completo los requerimientos de éstas.

2.1.3 Mecatrónica y las nuevas tendencias tecnológicas

Mecatrónica es el término utilizado para denotar el enfoque de un campo interdisciplinario de ingeniería que, mediante una integración coordinada y concurrente, se ocupa tanto del diseño como del desarrollo de productos y sistemas, cuyas funciones se basan en la integración de componentes mecánicos y electrónicos coordinados por una arquitectura de control, tomando como base la transferencia de funcionalidades del dominio físico al dominio de la información, lo cual ha dado lugar al desarrollo de productos cada vez más complejos, flexibles e inteligentes capaces de ser rediseñados, reprogramados, de entender y reaccionar a su entorno [14, 15].

Este enfoque multidisciplinario de la mecatrónica ha marcado el camino a seguir en el desarrollo de dispositivos IoT, ya que muchos de éstos resultan ser esencialmente mecatrónicos al requerir una compleja integración de elementos mecánicos y electrónicos con las tecnologías de la información, y en los que se mantiene la idea de transferir funcionalidades al dominio de la información. Tras el surgimiento de nuevas tendencias tecnológicas que tienen como núcleo el manejo de información, se están imponiendo nuevos requisitos sobre la tecnología utilizada para respaldar la producción actual de productos y servicios, así como la forma de compartir y procesar información.

Lo anterior implicará la utilización de sistemas compuestos por dispositivos IoT capaces de recolectar información y tomar decisiones inteligentes, los cuales a su vez resultarán ser mecatrónicos [16-18]. Esto abre un nuevo panorama sobre las nuevas áreas de aplicación en las que la mecatrónica puede participar, pero al mismo tiempo exige un cambio en la forma en que la mecatrónica es concebida, así como en la forma en que los sistemas y componentes de ésta son actualmente diseñados y fabricados, para satisfacer las nuevas demandas del mercado (ver figura 1.5). Esto no es algo nuevo, los avances tecnológicos desde siempre, han sometido a los sistemas mecatrónicos a revisiones para conocer qué los constituye y qué no, sobresaliendo ahora la necesidad de actualizar sus competencias en tecnologías de la información y la comunicación, en búsqueda de una integración sinérgica e inteligente de multisistemas [16-18]. Entre estas nuevas tecnologías se tienen:

- Análisis de grandes cantidades de datos, conocido como *Big Data*
- Minería de datos (*Data Mining*, DM)
- Inteligencia Artificial (*Articial* Intelligence, AI)
- Aprendizaje automático (*Machine learning*, ML)
- Aprendizaje profundo (*Deep Learning*, DL)
- Redes de sensores inalámbricos (*Wireless Sensor Networks*, WSN)
- Plataformas de nube
- Plataformas basadas en Cómputo de Borde (*Edge Computing*)
- Sistemas Físicos Cibernéticos (*Cyber-Physical Systems*, CPS) [16].

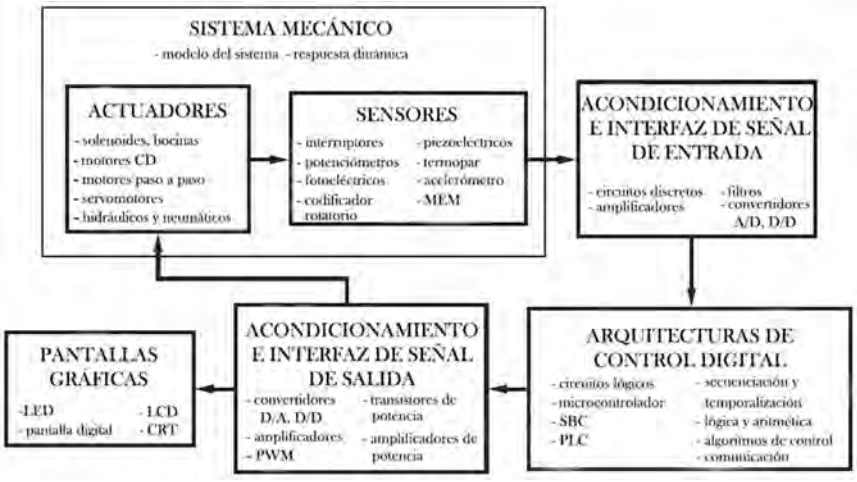


Figura 1.5 *Elementos de un sistema mecatrónico "clásico"* [15].

Esta nueva concepción de la mecatrónica dará lugar a un nuevo tipo de sistemas mecatrónicos (ver figura 1.6), los cuales incluyen tecnologías de comunicación inalámbrica para la interconexión de sus componentes o con otros sistemas, satisfaciendo necesidades de movilidad, autonomía, monitoreo y control de manera remota, lo cual da lugar a la creación de sistemas más grandes y complejos que sean personalizables, inteligentes y autónomos [16, 19]. Estas tecnologías también les permitirán a estos nuevos sistemas acceder a plataformas de nube o basadas en *Edge Computing,* para almacenar, procesar y analizar grandes cantidades de información, tareas que resultarían imposibles de realizar dentro de la mayoría de los sistemas mecatrónicos debido a sus recursos limitados [16].

Estos nuevos sistemas mecatrónicos permitirán crear productos inteligentes y lo suficientemente autónomos para no requerir ninguna o muy poca intervención humana. Esto será un reflejo de sus capacidades de autoaprendizaje, auto optimización, auto configuración, auto diagnóstico derivadas del acceso a aplicaciones eficaces de monitoreo, control y gestión, que serán independientes de la ubicación de éstos y que les permitirán una toma de decisiones y de actuaciones inteligentes [16].

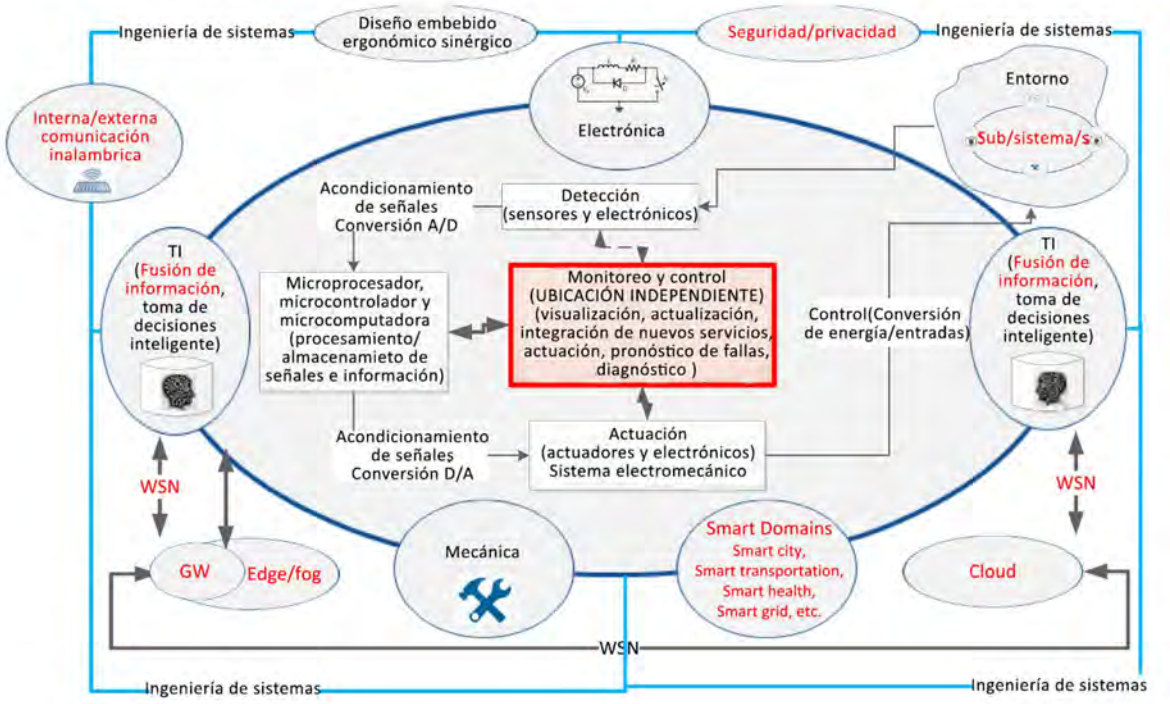


Figura 1.6 *Componentes principales de un sistema mecatrónico avanzado (Advanced Mechatronics Systems, AMS) y su interacción con el entorno y otros AMS.*[16].

2.1.4 Actuales tecnologías inalámbricas

El IoT ya ha dado sus primeros pasos en el desarrollo de aplicaciones que generalmente están destinadas a participar en redes WPAN y WLAN, lo que ha provocado que ciertas tecnologías de comunicación tomen una mayor relevancia. A continuación, se muestran algunas de las tecnologías actuales más populares que son utilizadas en esta tendencia:

a) NFC

NFC (*Near Field Communication* o Comunicación de campo cercano) es una tecnología basada en el estándar ISO 18092, que opera en la banda ISM a una frecuencia de 13.56 *MHz* con una velocidad máxima de transferencia de 424 *Kbps*. Esta tecnología se caracteriza por una comunicación segura, ya que ésta se efectúa cuando dos dispositivos están a una distancia menor de 20 cm [20]. Existen tres modos de operación de acuerdo con su aplicación:

- Modo escritura/lectura, utilizado para escribir o leer *tags* NFC, es decir, pequeñas etiquetas con un chip NFC que son incorporados comúnmente en carteles inteligentes.
- Modo emulación de tarjeta, empleado para realizar pagos o control de acceso, en el que el dispositivo funciona como una tarjeta sin contacto que sustituye a las tarjetas de crédito, tarjetas de débito, tarjetas de acceso, etc.
- Modo punto a punto (*peer to peer*), empleado para intercambiar información como mensajes de texto, registro de contactos u otro tipo de datos [20].

b) Thread

Thread es una tecnología opera en la banda ISM de 2.4 *GHz* y está basado en el estándar inalámbrico 6LoWPAN, que incorpora el Protocolo de internet versión 6 (IPv6) junto con un consumo energético bajo y que a su vez, se basa en el estándar de redes WPAN IEEE 802.15.4 lo que le permite acceder a internet y comunicarse con otras redes que empleen este protocolo, tales como Ethernet, WiFi o 3G/4G [21] [22]. Las redes *Thread* pueden soportar hasta 250 dispositivos con una velocidad máxima de transferencia de 250 *Kbps* y heredan de 6LoWPAN una topología de red en malla (ver figura 1.7) [21, 22], que está constituida por tres elementos:

- Enrutador de borde (*edge router*) es un tipo de enrutador que se encarga de intercomunicar a los dispositivos de la red, además de comunicarlos a internet.
- Enrutadores sirven para retransmitir la información a otros nodos.
- *Hosts* o dispositivos finales, son el origen o destino de la información [21, 22].

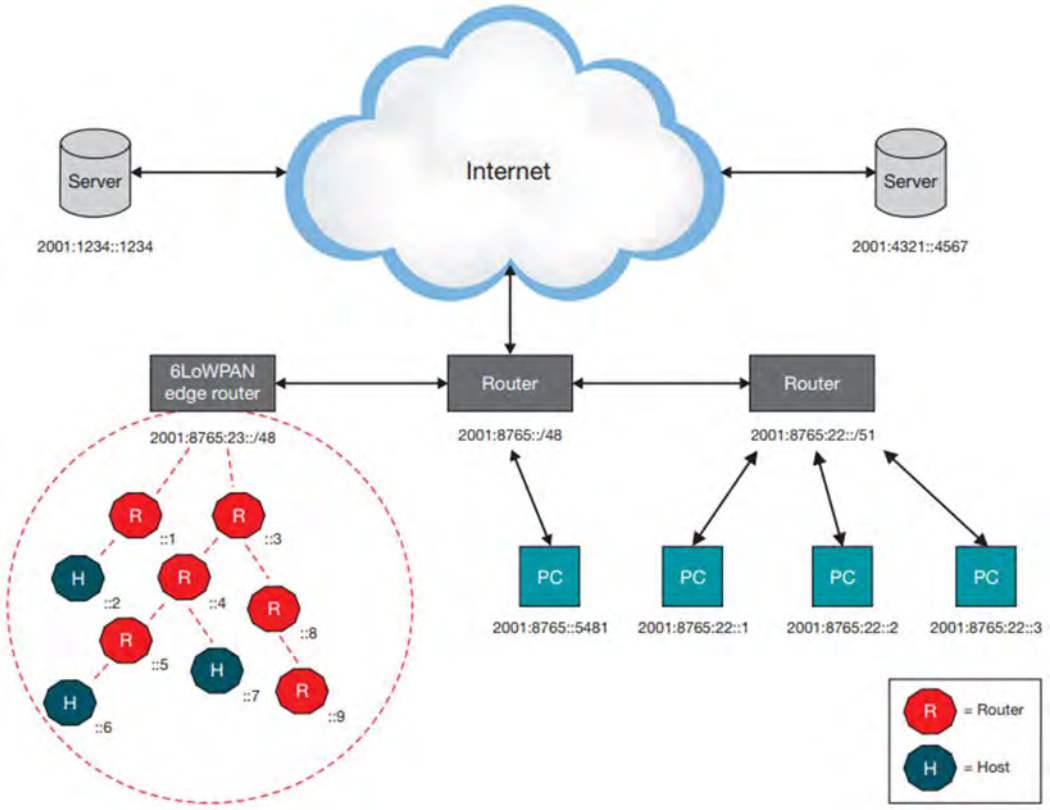


Figura 1.7 *Ejemplo de una red 6LoWPAN [22].*

Thread actualmente está siendo utilizado en aplicaciones para el hogar, edificios comerciales, electrodomésticos, control de acceso, control de temperatura, iluminación y seguridad.

c) Zigbee

Zigbee es una tecnología basada en el estándar IEEE 802.15.4, que opera en la banda ISM de 2.4 *GHz*, aunque también puede trabajar a 900 *MHz* y 868 *MHz* con una velocidad máxima de 250 *Kbps* [21]. Las redes *Zigbee* pueden trabajar en tres diferentes topologías: malla, árbol y estrella (ver figura 1.8). Cuando trabajan en malla son sumamente tolerantes a fallas porque son escaladas a través de multisaltos, es decir, que para que la información llegue a su destino se usan otros nodos como retransmisores [21, 23]. Esta configuración está constituida por tres elementos:

- Coordinador, el cual se encarga de inicializar, mantener y controlar la red.
- Enrutadores, los cuales transmiten la información entre los dispositivos finales y el coordinador.
- Dispositivos finales, que son quienes emiten o reciben la información [21, 23].

Las redes *Zigbee* son fácilmente adaptativas, ya que existen dos clases de dispositivos: los dispositivos de funcionalidad completa (*Full Function Device*, FFD) que pueden operar como coordinador, enrutador o dispositivo final; y los dispositivos de funcionalidad reducida (*Reduced Function Device*, RFD) que sólo operan como dispositivos finales [5]. *Zigbee* es utilizada frecuentemente para monitoreo y control de aplicaciones domóticas, además, empresas como Comcast, Deutsche Telekom, Alibaba, Philips y Osram emplean esta tecnología en sus productos y servicios [21].

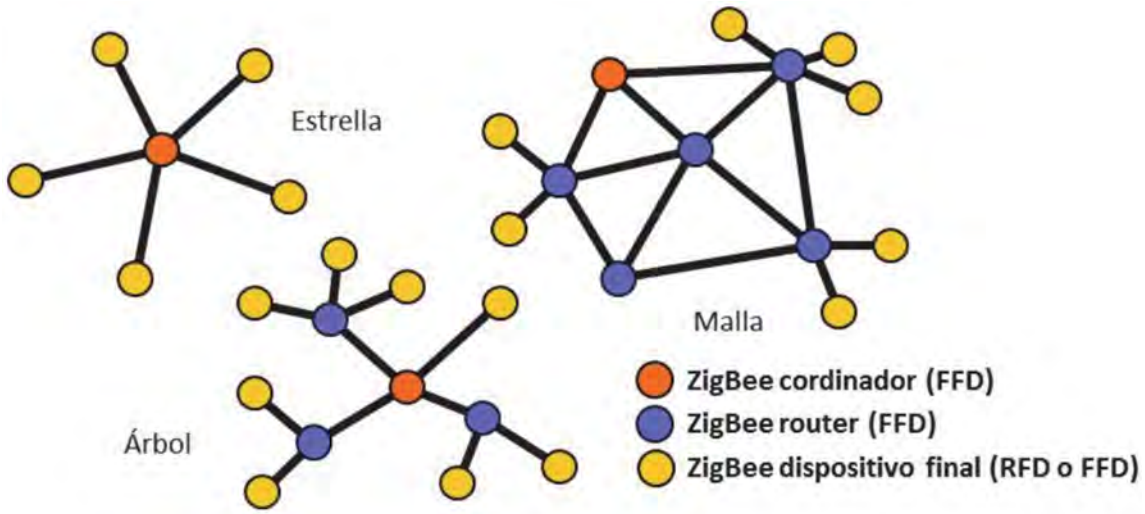


Figura 1.8 *Topologías soportadas por Zigbee [5].*

d) Bluetooth

Bluetooth es una tecnología que inicialmente estuvo basada en el estándar IEEE 802.15.1, que opera en la banda ISM de 2.4 *GHz* [21]. Esta tecnología emplea una arquitectura esclavo-maestro entre sus dispositivos, las redes creadas a partir de esta tecnología se conocen como *piconet* (ver figura 1.9), que se componen de un maestro y un máximo de 7 esclavos. Los dispositivos que utilizan *Bluetooth* son divididos en clases de acuerdo a su alcance máximo: clase 1 con 100 m, clase 2 con 10 m, y clase 3 con 1 m [5].

Existen principalmente dos tipos de *Bluetooth*, uno conocido como *Bluetooth Classic* que está presente en las primeras versiones y *Bluetooth* de baja energía (*Bluetooth Low Energy*-BLE) que ha sido implementado desde la versión 4.0. *Bluetooth Classic* ha sido muy popular en periféricos de computadoras y accesorios de teléfonos móviles como lo son teclados, ratones y auriculares. En cambio, BLE ha apostado al mundo del IoT principalmente por el reducido consumo energético y la conectividad a internet, para la última versión (5.0), ofrece una velocidad de transferencia hasta 2 *Mbps,* además de

mejorar la conexión a internet, gracias a la inclusión de perfil de soporte de protocolo de internet (*Internet Protocol Support Profile*, IPSP) compatible con IPv6 [21].

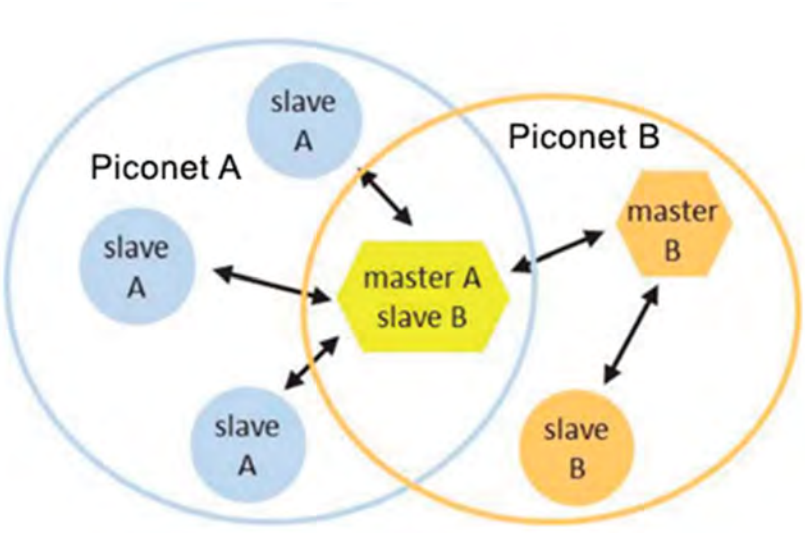


Figura 1.9 *Ejemplo de un piconet* (adaptado de *[5])*.

e) Z-Wave

Z-Wave es una tecnología desarrollada por *Z-Wave Alliance*, que opera en las bandas ISM de 908.42 *MHz* para Estados Unidos y de 868.42 *MHz* para Europa, con una velocidad máxima de 100 *Kbps* [24]. A diferencia de muchas tecnologías inalámbricas, ésta no está basada en algún estándar del IEEE. Las redes *Z-Wave* siguen una topología de red en malla, que puede soportar hasta 232 dispositivos. Existen dos tipos de dispositivos: el controlador que se encarga de coordinar el envío de información y de acceder a internet; y los esclavos que son el origen o destino y que también pueden retransmitir información cuando un nodo no está al alcance del controlador. Esta tecnología es ampliamente utilizada en domótica como en cerraduras inteligentes, sistemas de iluminación, control de temperatura y sistemas de seguridad [24, 25].

f) Sigfox

Sigfox es un proveedor de red similar a las operadoras móviles, que opera en la banda ISM de 915 *MHz* en Estados Unidos y 868 *MHz* en Europa, proporcionando una conectividad de muy largo alcance, bajo costo y bajo consumo energético, la cual es bidireccional, pero frecuentemente es utilizada como si fuera unidireccional [21].Actualmente, la cobertura de esta tecnología abarca gran parte de Europa, Estados Unidos y Latinoamérica [26]. Esta tecnología está dirigida para aplicaciones de IoT con latencia alta y transmisiones de pequeñas cantidades de datos, con un máximo de 12

bytes por mensaje y un máximo de 140 mensajes al día [21]. Los mensajes emitidos por los dispositivos son detectados por la estación base más cercana, son decodificados y transmitidos por la red pública de *Sigfox*, llevando la información a la nube de *Sigfox* que finalmente puede ser accedida a través de las plataformas de modo administrador o *back end* (ver figura 1.10) [21, 26]. *Sigfox* es utilizado para alarmas de incendios, gestión de control de instalaciones, iluminación telecontrolada, entre otras.

g) Telefonía móvil

La telefonía móvil ha tenido un enorme desarrollo, pasó de ser una comunicación analógica exclusiva para llamadas a una comunicación digital que permite acceder a internet, de hecho, actualmente es una de las principales tecnologías de acceso a esta gran red [6, 27]. La evolución de la telefonía móvil está marcada por generaciones, cada una de éstas ofrecen nuevas características o mejoras [27]. La presente generación es la 4G, la cual está basada completamente en el protocolo IP, permitiendo la comunicación a internet. Además, la transferencia de datos puede alcanzar hasta un 1 *Gbps*. Dentro de esta generación existe la tecnología LTE, por las siglas en inglés de *Long Term Evolution* o evolución a largo plazo, que permite una transmisión de datos a más de 300 *Mbps* en movimiento [5, 27].

Figura 1.10 *Ejemplo de arquitectura de red Sigfox [26].*

h) WiFi

WiFi es una marca de una tecnología certificada por la *WiFi Alliance*, la cual está basada en el estándar IEEE 802.11 y opera comúnmente en la banda ISM de 2.4 *GHz*, aunque puede operar en otras bandas. Se diseñó como reemplazo inalámbrico del estándar Ethernet IEEE 802.3 con el propósito de acceder a internet, a través de la pila TCP/IP. Esta tecnología tiene una topología de red en estrella, en el que el punto de acceso (*Access Point*, AP) se encarga de comunicar los dispositivos y de ingresar a internet (ver figura 1.11). Algunos AP pueden llegar a comunicar hasta 250 dispositivos, aunque los más comunes soportan hasta 50 dispositivos. También, existen algunos dispositivos WiFi que se pueden comunicar en forma *peer to peer* sin necesidad de un AP, esta tecnología se conoce como *WiFi Direct* [21].

Figura 1.11 *Ejemplo de una red WiFi [5].*

Gracias a su bajo costo, que trabaja en espectros de frecuencia no licenciados, la facilidad de implementación de una red WiFi y su interoperabilidad de versiones, lo que ha convertido a ésta en una de las tecnologías inalámbricas más importantes y con mayor presencia en el mundo, siendo su principal uso el de brindar conectividad a internet, asimismo, la gran infraestructura disponible, también ha facilitado su implementación en el IoT. Aunque el WiFi tiene un mayor consumo energético que otras tecnologías, ésta ofrece velocidades mayores de transferencia, haciéndola atractiva para diversas aplicaciones IoT aún sin considerar sus constantes actualizaciones para adecuarse a esta tendencia [21, 28].El uso de esta tecnología no es ajeno a la mecatrónica, ya que suele ser empleada para establecer comunicaciones inalámbricas con otros sistemas o subsistemas mecatrónicos, que junto con lo anterior llevaron a la elección de esta tecnología para el trabajo que aquí se presenta.

2.2 WiFi

2.2.1 Origen y evolución

Para hablar del WiFi es necesario remontarse al origen de las redes de área local. Todo inició con ALOHAnet, que como se mencionó previamente, fue la primera red inalámbrica de computadoras y que, si bien, por su gran cobertura podría considerarse actualmente como una red WWAN, es conocida como la primera red WLAN, ya que ilustró los conceptos de una estación base, la topología estrella y la comunicación mediante paquetes, conceptos que la mayoría de redes WLAN aún implementan en la actualidad [4].

A pesar del desarrollo de ALOHAnet, el impulso para el desarrollo de las redes WLAN llegaría hasta el año de 1980, cuando la FCC liberó el uso de las bandas ISM, tras lo cual se observaría durante los siguientes diez años un incremento en la demanda de conexiones inalámbricas, especialmente en computadoras. Diversos fabricantes como IBM, *Digital Equipment y Symbol Technologies*, comenzaron a desarrollar sus propias tecnologías WLAN. Esto generó un problema para los consumidores, ya que no existía compatibilidad entre los productos que estaban en el mercado [29].

Ante esta situación, el IEEE reconoció la necesidad de crear un estándar que permitiera uniformar las diferentes tecnologías WLAN que estaban surgiendo, por lo que el Comité de Estándares de LAN/MAN o *LAN/MAN Standards Committee* (IEEE 802 LMSC), decidió crear en el año de 1990 el grupo IEEE 802.11 para este fin. En el año de 1997, el IEEE lanzó el primer estándar para las redes WLAN con el nombre de 802.11, el cual define una tasa de datos de hasta 2 *Mbps* a una frecuencia de 2.4 *GHz* [29].

Aunque el lanzamiento de este estándar incrementó el uso de redes WLAN, especialmente en almacenes y zonas de fabricación, no logró que éstas fueran lo suficientemente atractivas para ser implementadas en otros sectores del mercado. Esto llevó al IEEE 802.11 a generar grupos de trabajo que se encargarían de desarrollar extensiones del estándar original, las cuales estarían enfocadas a satisfacer las necesidades de los diferentes sectores del mercado, lo cual marcó el nacimiento de la familia de estándares 802.11, los cuales definen las características y requisitos de un tipo de red WLAN [29, 30].

A pesar de que los estándares lanzados por la IEEE dieron uniformidad a las características de las redes WLAN, los equipos que eran desarrollados bajo estos estándares no resultaban ser totalmente compatibles entre sí. Esto se debió a un problema de interpretación que suele ocurrir frecuentemente en estándares que resultan ser

ambiguos, ya que brindan a los fabricantes la posibilidad de aplicar diferentes técnicas o tecnologías para alcanzar los resultados deseados [30].

Ante esta situación, en el año de 1999 las empresas 3COM, Aironet (Cisco), *Lucent Technologies* (Agere), Intersil, Nokia y *Symbol Technologies* crearon la Alianza de Compatibilidad Ethernet Inalámbrica (*Wireless Ethernet Compatibility Alliance*, WECA), una asociación sin fines de lucro cuyo principal objetivo era impulsar la implementación de las redes WLAN en los diferentes sectores del mercado. Para lograr esto, la WECA crearía una certificación basada en la familia de estándares 802.11, la cual aseguraría a los consumidores la interoperabilidad entre los productos de diferentes fabricantes [30-32] . Como la WECA sería la encargada de diseñar dicha certificación, fue necesaria la creación de una marca comercial que identificara dentro del mercado a los productos que fueran certificados [32].

WiFi sería el nombre elegido por WECA para su marca comercial, el cual no tiene un significado y está inspirado en el acrónimo de *High Fidelity* (Hi-Fi), con la intención de que las personas asociaran el termino WiFi con el establecimiento de una comunicación inalámbrica de alta calidad [30, 33]. Sería hasta el año 2000 cuando oficialmente se da a conocer el nombre de la marca en el mercado, cuyo logo (ver figura 1.12) era aplicado únicamente a los productos que aprobaran la primera certificación de la WECA llamada ahora *WiFi CERTIFIED b,* también lanzada el mismo año*,* la cual está basada en el estándar 802.11b y define una transmisión de datos de 11 *Mbps* en una banda de 2.4 *GHz.* En este mismo año, la WECA cambiaría su nombre a Alianza WiFi o *WiFi Alliance* en inglés, con la finalidad de ser consistentes con el lanzamiento de su marca [31, 34] . A partir de este punto, el WiFi comenzó su evolución y expansión dentro del mercado, adaptándose a las continuas necesidades que aparecían en éste.

Figura 1.12 Logo WiFi [35].

Tabla 1-1 Evolución del WiFi [31, 34] .

Año	Evento	Descripción
2002	*WiFi Alliance* lanza *WiFi CERTIFIED a* basado en el estándar 802.11a	Define una transmisión de datos de 54 *Mbps* en una banda de 5 *GHz*
2003	*WiFi Alliance* lanza *WiFi CERTIFIED g* basado en el estándar 802.11g	Define una transmisión de datos de 54 *Mbps* en una banda de 2.4 *GHz*
2004	*WiFi Alliance* lanza WPA2 (*WiFi Protected Access 2*) basado en el estándar 802.11i	Utiliza AES como técnica de encriptación satisfacía todos requerimientos para ser considerada de grado gubernamental
2007	*WiFi Alliance* comienza la certificación de productos basados estándar *802.11n draft 2.0*	Incorpora la tecnología MIMO (*Multiple-Input-Multiple-Output*) y define transmisiones de datos de hasta 300 Mbps, marcando el inicio de una nueva era del WiFi
2009	*WiFi Alliance* lanza WiFi CERTIFIED n basado en el estándar 802.11n	Fue anunciado como una actualización del certificado WiFi CERTIFIED 802.11n draft 2.0
2010	*WiFi Alliance* lanza WiFi Direct	Es un certificado para dispositivos que pueden conectarse directamente para compartir información y sincronizar datos
2012	*WiFi Alliance* lanza *WiFi CERTIFIED Passpoint*, basado en las especificaciones *Hostpot 2.0*	Los dispositivos móviles detectar y conectarse a puntos de acceso WiFi de manera automática
2013	Se unifican la Wireless Gigabit Alliance (WiGig) y la WiFi Alliance y se lanza Wi Fi CERTIFIED ac basado en el estándar 802.11 ac	Define transmisiones de datos del orden de varios Gbps en una banda de 5 GHz y un reducido tiempo de latencia
2016	*WiFi Alliance* lanza WiFi HawLow basada en el estándar 802.11ah	Diseñada para dispositivos de bajo consumo energético en una banda de 900 MHz y con un alcance que duplicaba el de las redes Wi Fi convencionales. Especial para IoT.
2016	*WiFi Alliance* lanza WiFi CERTIFIED WiGig basada en el estándar 802.11ad	Define una tasa de trasmisión de datos del orden de varios Gbps en una banda de 60 GHz y un bajo tiempo de latencia en un radio menor a 10 metros
2018	*WiFi Alliance* lanza WiFi CERTIFIED WPA3	Certificado de seguridad con mayor fortaleza de cifrado respecto a su antecesor.
2018	*WiFi Alliance* introduce nueva clasificación generacional el WiFi	Se identifica como WiFi 4 a los productos basados en 802.11n, WiFi 5 a los productos basados 802.11ac y WiFi 6 a los productos basados en 802.11ax
2019	*WiFi Alliance* lanza WiFi CERTIFIED 6, basado en el estándar 802.11ax	Busca satisfacer los nuevos requerimientos de transmisión en tiempo real de las aplicaciones como video 4K/8K, realidad virtual y el bajo consumo que demanda el *internet of Things*

En la *tabla 1-1* se señalan los eventos más relevantes acontecidos durante la evolución de esta tecnología al paso de los años. Aunque el WiFi sólo es el nombre de una marca comercial, la gran cantidad de equipos certificados bajo ésta y que son utilizados para la construcción de redes WLAN, ha hecho que el término WiFi se haya convertido en un sinónimo de redes WLAN.

2.2.2 Actualidad

En la actualidad, el WiFi se ha convertido en una de las tecnologías inalámbricas con mayor uso a nivel global, posicionándose también como uno de los principales medios para el tráfico de internet. Esto ha hecho que esta tecnología sea considerada cada vez con mayor frecuencia, en un sentido amplio, como una herramienta para el desarrollo de las actividades diarias, mejorar la experiencia en ciertos lugares, para la comunicación y satisfacción de necesidades de conexión. Esto ha llevado a que se dé por hecho su presencia en la mayoría de los dispositivos que consumen información como celulares, computadoras y tabletas, así como en aeropuertos, cafeterías, escuelas, centros comerciales, hospitales, hoteles, casas, oficinas entre otros espacios exteriores e interiores[28].

A pesar de que el WiFi no es la única tecnología que puede ser empleada para brindar acceso a internet, la relación que se ha construido entre éstas provoca que sean utilizados como sinónimos, generando la aparición de diferentes modelos de negocio y servicios que permiten rentabilizar esta tecnología. Ejemplo de esto es el ofrecimiento de acceso a internet con el eslogan “WiFi Gratis” como forma de atraer clientes a los establecimientos, característica que ya es vista por algunas personas como un requerimiento en aquéllos. Lo anterior también se ve apoyado por lo fácil y económico que resulta agregar la tecnología a los dispositivos, lo cual les permite utilizar la infraestructura ya disponible[28].

A pesar de que el sector industrial ha sido renuente al uso de tecnologías inalámbricas por factores como:

- Falta de confiabilidad (perdida de paquetes de información) por afectaciones debidas a la interferencia electromagnética como de radio frecuencia producida por los equipos industriales (EMI y RFI).
- La falta de interoperabilidad entre la diversidad de protocolos de comunicación existentes de corto y largo alcance, así como su falta de estandarización para un uso industrial.
- Problemas para incorporar estas tecnologías a equipos con los que ya se cuentan, que no fueron diseñados para recolectar información y comunicarse con otros.

- Falta de tecnologías que cumplan los requerimientos de velocidad, carga de transmisión, distancia o consumo energético que se requieren.
- Problemas asociados con la seguridad y la integridad de la transmisión de la información[36, 37].

Tabla 1-2 Aplicabilidad de WiFi (adaptado de [37]).

Área	Tarea	Nivel
Monitoreo y control basado en flujo	Monitoreo de procesos	●
	Control de supervisión	●
	Control de realimentación	■
	Condiciones de alarma	■
Monitoreo y control basado en trabajo	Monitoreo de fábrica	●
	Ensamblaje: Sensado	■
	Ensamblaje: Actuación	■
	Robots: Supervisión	■
	Robots: Control de realimentación	■
	Inspección de calidad	■
Seguridad personal (Reporte de problemas para prevenir lesiones)	Prevención de caídas	♦
	Espacios confinados	♦
	Detección de eventos críticos	■
	Colocación Humano-Máquina	♦
Back-houl (Red que comunica una red de nivel inferior con una red de nivel superior)	Cercanos o interiores	●
	Lejanos: LOS	●
	Lejanos: BLOS	●
Rastreo	Localización de máquinas en interiores	■
	Materiales en almacenes	▲
	Materiales en producción	▲
	Herramientas	▲
	Personal	▲
Seguridad	Comunicación por voz o video	●
	Vigilancia en video	●
	Control de tierra	●
	Vigilancia basada en drones	●
	Autorización de personal	●
Monitoreo de forma remota	Monitoreo de boca de pozo	■
	Monitoreo de tuberías	♦
	Monitoreo de nivel de tanque	■
Soporte de mantenimiento	Monitoreo del estado de máquina	■
	Automatización de edificios	●
	Realidad aumentada	●

● Satisface, ■ Satisface con practicidad, rendimiento, latencia, confiabilidad o limitaciones de energía, ♦ No recomendado, ▲Los requisitos de energía de dispositivos, supuestamente alimentados con una batería, restringen su aplicabilidad.

El WiFi ha empezado a ser considerado como una opción para ciertas tareas de este sector, entre las que destacan monitoreo, comunicación y control no crítico, sin embargo, su uso dependerá de los requerimientos de la aplicación En la tabla 1-2 se mencionan algunas actividades en las que el WiFi puede tener participación y su idoneidad

La adopción de la tecnología WiFi ha tenido dos grandes impactos en el mundo. El primero de ellos es económico ya que tan sólo en el año 2018 se estimó que el valor económico del WiFi fue de 1.96 billones de dólares a nivel global y se estima que crezca hasta los 3.47 billones en el año 2023[28]. Este impacto en la economía se deriva de cuatro áreas en las que el WiFi ha tomado una mayor relevancia:

- Desarrollo de tecnologías alternativas para expandir la elección del consumidor
- Creación de modelos comerciales innovadores para brindar servicios únicos
- Expansión del acceso a los servicios de comunicaciones para redes fijas y móviles
- Complementación de tecnologías celulares y cableadas para mejorar su efectividad[28].

El otro impacto del WiFi se da en el ámbito social, ya que debido a su amplio uso a nivel global para la conectividad, su rentabilidad y la facilidad de desplegar redes WiFi entre otras características, hacen que ésta comience a ser vista como una herramienta para la reducción de la brecha digital[28], idea que es retomada e impulsada por diversas iniciativas entre las que destaca el Día de WiFi Mundial (*World WiFi Day*) de la Alianza de Banda ancha inalámbrica (*Wireless Broadband Alliance*, WBA), la cual declara al 20 de junio como el día mundial del WiFi para hacer conciencia en las personas, empresas y gobiernos del papel que tiene el WiFi en el desarrollo socioeconómico de las ciudades y en la comunicación que se da entre éstas y las comunidades y, por tanto, impulsar el desarrollo de proyectos innovadores que permitan llevar el WiFi a más lugares, de manera asequible, para romper la brecha digital [38].

Estas acciones se materializan en las ciudades, al ofrecerse en éstas puntos de acceso WiFi gratuitos[28], tal es el caso de la Ciudad de México, en la cual se ha impulsado la elaboración de convenios y proyectos con empresas para brindar acceso WiFi a toda la población como parte de su estrategia de promover la conectividad en la ciudad. Entre los proyectos más sobresalientes se tienen:

- La instalación de equipos para la conectividad WiFi gratuita en más de 13,000 postes vía infraestructura del C5 [39].
- 96 espacios públicos con WiFi gratuito, ilimitado y funcional con valor de 17 millones de pesos como parte del programa Ciudad Digital [39].

- 14,500 puntos con acceso gratuito a WiFi con un valor de $11.5 millones de pesos como parte del programa Ciudad Segura [39].
- Actualmente AT&T ha desplegado su cobertura 4G LTE y WiFi en 6 de las 12 líneas del metro, lo que representa 120 kilómetros de conectividad gratuita [40].

WiFi ha tenido una notable integración en los dispositivos del IoT, a tal grado que los puntos de acceso WiFi son considerados claves para el desarrollo de los servicios y aplicaciones para esta tendencia, pronosticándose hasta 698 millones de puntos de acceso públicos a nivel global para el año 2023[31, 41]. A pesar de la constante evolución en los requerimientos del IoT, el lanzamiento de nuevas generaciones WiFi para satisfacer estos requerimientos; su presencia actual a nivel global y su integración en los nuevos servicios; sus fortalezas inherentes como la interoperabilidad entre diferentes generaciones WiFi y la inversión que se ha realizado en esta tecnología hacen que el WiFi muy difícilmente pueda ser reemplazado de manera rápida aún con el despliegue de 5G [28], por lo que es más probable su incorporación en otras tendencias como lo son las redes heterogéneas.

2.3 Actualidad de herramientas de desarrollo WiFi

Con el incremento de productos que se pueden considerar mecatrónicos, se generó una necesidad por la formación de ingenieros que tuvieran conocimientos sobre sistemas embebidos y altas habilidades en programación, lo cual impulsó el diseño y popularizó el uso de las denominadas tarjetas de desarrollo[42]. Estas tarjetas no son más que placas de circuito impreso diseñadas para un microcontrolador o procesador, las cuales cuentan tanto con los elementos mínimos de *hardware* como su propio *software* que les permiten ser empleadas como elementos de control. Además, cuentan con un tamaño reducido, así como con un nivel de abstracción computacional tal, que no hace necesario conocer la arquitectura subyacente de éstas para su utilización. Con la llegada de las nueva tendencias tecnológicas, las tarjetas de desarrollo comienzan a tomar un papel relevante en las áreas de conocimiento del STEM (*Science, Tecnology, Enginnering and Mathemathics*) para ser utilizadas como herramientas de aprendizaje en un modelo orientado a la elaboración de proyectos[43].

El IoT y la creciente necesidad en el desarrollo de sistemas distribuidos propiciaron la aparición en el mercado de nuevas tarjetas de desarrollo, y componentes para las ya existentes, que incluyen tecnologías como WiFi, Bluetooth, GPRS, LTE. Actualmente, el mercado ofrece una gran variedad de herramientas de desarrollo que incorporan WiFi; ante esta variedad es necesario realizar una clasificación de éstas para entender la utilidad que cada una ofrece. Estas herramientas se categorizarán en tres grupos:

- *Shields* o tarjetas complementarias
- Módulos WiFi
- Tarjetas de desarrollo que integran WiFi

Las tarjetas de desarrollo serán categorizadas en un subgrupo que toma como criterio principal tanto a su elemento de control, así como el número de tecnologías de comunicación que incorporan:

- Tarjetas de desarrollo basadas en microcontroladores
- Tarjetas de desarrollo basadas en microprocesadores
- Tarjetas de desarrollo híbridas
- Tarjetas de desarrollo dedicadas.

2.3.1 Shields

Una de las primeras opciones que aparecieron en el mercado son las denominadas *shields,* las cuales son placas de circuito impreso que no pueden ser usadas de manera independiente y que integran elementos de *hardware* para añadir a tarjetas de desarrollo alguna funcionalidad extra, colocándose generalmente encima de éstas. La conexión entre la *shield* y la tarjeta de desarrollo se realiza mediante la interconexión de sus pines (siempre que se respete la misma distribución), permitiendo la comunicación entre ambos y en algunos casos, el suministro de energía desde la tarjeta de desarrollo hacia la *shield* [44]. Para su uso, generalmente estas incluyen su propio *software* en forma de una biblioteca. Algunos tipos de *shield* permiten que se pueda apilar otra sobre ésta, lo que añade más funcionalidades a la tarjeta de desarrollo [44].

Como ejemplo de esta primera forma de integrar la tecnología WiFi se tiene la *Arduino WiFi Shield* (ver figura 1.13), la cual fue desarrollada para la tarjeta Arduino Uno y si bien, actualmente ha sido descontinuada por parte de Arduino, aún es posible encontrarla dentro del mercado. Sus características se muestran en la tabla 1-3.

Tabla 1-3 Características Arduino WiFi Shield [45].

Protocolos WiFi	**802.11b/g**
Encriptación	WEP y WPA 2
Comunicación con la tarjeta	SPI
Voltaje de operación	5 V
Tarjeta compatible	Arduino UNO y Arduino Due

Figura 1.13 *Arduino WiFi Shield [45].*

2.3.2 Módulos WiFi

Los módulos WiFi, también conocidos como *Transceiver WiFi SERIAL,* es decir trasmisores-receptores seriales WiFi, son dispositivos que principalmente son utilizados para brindar la tecnología WiFi a cualquier tarjeta de desarrollo u otros dispositivos que implementen alguna interfaz de comunicación compatible, encargándose del manejo de toda la pila de protocolos para logar realizar tanto la transmisión como la recepción de información [46, 47]. Estos módulos están construidos a partir de un SoC WiFi, que son la siglas den inglés de sistema sobre un chip (*System On a Chip*), el cual es un chip que integra todos los elementos necesarios como lo pueden ser microprocesadores, periféricos, interfaces WiFi entre otros, reduciendo significativamente el *hardware* para su utilización, caracterizándose por un menor tamaño y consumo energético [47, 48].

Si bien el *hardware* y el *software* de un módulo WiFi varían conforme al fabricante y al SoC que incorporan, el nivel de abstracción de éstos hace que estén a la par de la mayoría de las tarjetas de desarrollo existentes en el mercado. Debido a lo anterior, los módulos WiFi pueden ser utilizados de dos formas:

- De manera independiente, convirtiéndose en el elemento de control para el desarrollo de alguna aplicación con capacidades de comunicación mediante WiFi, utilizando para este fin los periféricos del SoC para los cuales el módulo brinde acceso. Esto implica cargar directamente en el módulo un programa para las tareas que se quiere que éste realice, de la misma forma en que se programa una tarjeta de desarrollo convencional.
- Como un elemento esclavo de un microcontrolador o de un sistema de microcontroladores más grande, con la finalidad de brindarle a éste la tecnología WiFi, convirtiendo al módulo en un adaptador, puesto que éste se encarga de las tareas relacionadas con la comunicación. Para este fin, el módulo y el elemento que

actúe como su maestro deberán comunicarse mediante alguna interfaz, siendo muy probable la utilización de una biblioteca o *software* en el lado del maestro para simplificar la programación de la interacción de éstos [46, 49].

Actualmente, uno de los módulos WiFi más populares es el ESP-01 (ver figura 1.14), debido a su reducido precio, los lenguajes con los que se puede programar y a las características que éste presenta (ver tabla 1-4). El *firmware* que incorpore el módulo definirá en gran medida el modo en que éste trabajará.

Tabla 1-4 Características módulo ESP-01 [46, 50].

Protocolos WiFi	**802.11 b/g/n**
Frecuencia de operación	80-160 *MHz*
Memoria de programa **Memoria de datos**	64 K*B* 96 K*B*
Memoria Flash QSPI Externa	Hasta 16 *MB*
Interfaces de comunicación	SDIO, I2C, SPI, UART

Figura 1.14 *Módulo ESP-01 [50].*

También es común encontrar módulos como el que se observa en la figura 1.15, los cuales se conocen como módulos de montaje superficial, que suelen estar reservados para placas de circuito impreso diseñadas para alguna aplicación en específico. Ejemplo de ello es su presencia en placas de circuito impreso para utilizarlos como elementos de control de una tarjeta de desarrollo, integrados como complemento en una tarjeta que integra otro microcontrolador o inclusive en algún *shield*.

Figura 1.15 *Módulo ESP-12 [51].*

2.3.3 Tarjetas de desarrollo

El primer tipo de tarjetas de desarrollo al que se refiere la clasificación corresponde a las basadas en microcontroladores. Este tipo de tarjetas son utilizadas para el desarrollo de aplicaciones específicas en los que se realizan pocas o inclusive una única tarea, las cuales se caracterizan por ser repetitivas, ya que el nivel de procesamiento y nivel de cálculo que presentan no es tan elevado. Suelen incorporar un puerto de comunicación serial, así como un fácil acceso a los GPIO analógicos y/o digitales del microcontrolador que incorporan para el control de sistemas físicos. También son más simples y fáciles de programar que las tarjetas basadas en microprocesadores.

Las tarjetas que ya integran WiFi u otra tecnología de comunicación resultan ser más costosas respecto a las que no la integran. Una de las tarjetas más populares de este tipo es la NETduino 3 WiFi, mejor conocida como N3 WiFi (ver figura 1.16), cuyas características se muestran en la tabla 1-5. Otras tarjetas que son de este tipo que ya incorporan la tecnología WiFi aparecen en la tabla 1-6.

Figura 1.16 *NETduino 3 WiFi [52].*

Tabla 1-5 Características N3 WiFi [52].

Protocolos WiFi	**802.11b/g/n con SSL/TLS**
Flash	1408 *kB*
RAM	164 *kB*
Memoria externa SD	2 *GB*
MCU	Cortex-M4 @ 168 *MHz*

Tabla 1-6 Tarjetas de desarrollo basadas en microcontroladores [45, 53, 54]

Tarjeta	**WiFi**	**RAM**	**Flash**	**MCU**
Arduino YUN	802.11b/g/n	64 MB DDR2	16 *MB*	Atmega32u4 @ 16 *MHz*
Photon	802.11b/g/n	128 *kB*	1 *MB*	ARM Cortex M3 @ 120 *MHz*
NodeMCU ESP8266	802.11b/g/n	*Data* 96 *kB* *Instruction* 32 *kB*	4 *MB*	ESP-12 Tensilica Xtensa LX3 @ 80/160 *MHz*
Arduino MKR1000	802.11b/g/n	SRAM 32 *kB*	256 *kB*	ARM SAMD21 Cortex @ 48 *MHz*
Tessel 2	802.11b/g/n	64 *MB* DDR2	32 *MB*	Atmel SAMD21 @ 48 *MHz*

El segundo tipo de tarjetas de desarrollo son las que se basan en microprocesadores, las cuales son utilizadas para el desarrollo de aplicaciones que requieran ejecutar varias tareas a la vez, ya que ofrecen grandes velocidades de procesamiento y una gran potencia de cálculo para este fin. Además, pueden incorporar varias tecnologías de comunicación y otros elementos como tarjetas gráficas, adaptadores de audio y puertos USB que permitien el uso de periféricos más demandantes.

Una de las tarjetas de este tipo que actualmente está ganado popularidad es la Omega 2 Plus (ver figura 1.17), que por sus características es considerada como una minicomputadora, la cual utiliza alguna versión personalizada de Linux como sistema operativo y, por tanto, puede ser programada en una gran variedad de lenguajes [55].

Figura 1.17 *Omega 2 Plus [55].*

La Omega 2 Plus se caracteriza por su bajo costo ya que no integra periféricos como lo hacen otras tarjetas de este tipo, sino que éstos deben ser comprados por separado como módulos de expansión para esta tarjeta. En la tabla 1-7 se muestran las principales características de esta tarjeta. El precio de este tipo de tarjetas suele ser mayor respecto a las tarjetas de desarrollo basadas en microcontroladores. Otras tarjetas de este tipo que incorporan la tecnología WiFi se muestran en la tabla 1-8.

Tabla 1-7 Características de la Omega 2 Plus [55]

Protocolos WiFi	**802.11 b/g/n**
CPU	MIPS @ 580 *MHz*
Memoria	128 *MB* DDR2 DRAM
Almacenamiento	32 *MB*
Voltaje de operación	3.3 *V*
Protocolos	UART, I2C, SPI

Tabla 1-8 Tarjetas de desarrollo basadas en microprocesador [50, 53, 56].

Tarjeta	Conexión	CPU	RAM	GPIO
Raspberry Pi 3 *Model B+*	802.11b/g/n/ac	ARM Cortex-A53 @ 1.54 *GHz*	1 *GB* LPDDR2 SDRAM	40
UDOO NEO EXTENDED	802.11b/g/n	ARM Cortex-A9	1 *GB*	32
Beagle Bone Wireless	802.11b/g/n	ARM Cortex-A8 @ 1 *GHz*	512 *MB* DDR3 RAM	2x46

El tercer tipo de tarjetas de desarrollo son las hibridas, las cuales incorporan más de una tecnología de comunicación como WiFi, Bluetooth, LTE, Ethernet entre otras, sin importar si están basadas en un microcontrolador, microprocesador o inclusive construidas tomando como base a un módulo WiFi. Debido a esto, estas tarjetas suelen ser más caras y existe una menor variedad de ellas. Una de las tarjetas de este tipo más populares actualmente, es la ESP32 DevKitC (ver figura 1.18) la cual está construida a partir de un módulo ESP32, lo que le permite incorporar tanto WiFi como Bluetooth y puede ser programada mediante varios lenguajes [51]. Las características más importantes de esta tarjeta de desarrollo se muestran en la tabla 1-9.

Tabla 1-9 Características ESP32 DevKitC ESP32 [51, 57].

Protocolos WiFi	**802.11b/g/n/e/i**
Bluetooth	v4.2 BR/EDR y *Bluetooth Low Energy* (BLE)
CPU principal	Tensilica Xtensa 32-bit LX6 @ 240 MHz

Figura 1.18 *ESP32 DevKitC ESP32 [57].*

Si bien, las tarjetas mencionadas en la tabla 1-8 también entran dentro de esta categoría, existen en el mercado otras tarjetas de desarrollo que resultan ser más representativas, las cuales se muestran en la tabla 1-10

Tabla 1-10 Tarjetas de desarrollo hibridas [54, 58].

Tarjeta	**Comunicación**	**RAM**	**Flash**	**Protocolos**
Pycom WiPy 3.0	WiFi y Bluetooth	4 *MB*	8 *MB* (externa)	2 x UART, 2 x SPI, I2C, I2S
Pycom GPy	WiFi, Bluetooth LE y LTE-M	4 *MB*	8 *MB*	2 x UART, 2 x SPI, I2C
Pycom LoPy4	WiFi, Bluetooth LE, LoRa y Sigfox	4 *MB*	8 *MB* (eterna)	2 x UART, SPI, 2 x I2C, I2S,
LinkIt™ ONE	WiFi, Bluetooth, GMS, GPRS y GNSS	4 *MB*	16 *MB*	I2C, SPI, UART

Finalmente, las tarjetas de desarrollo denominadas dedicadas son aquéllas que, independientemente de si incorporan un microcontrolador, procesador o inclusive un módulo WiFi, están diseñadas para un tipo de aplicación en particular y, por tanto, integran componentes de *hardware* específicos, poco usuales de encontrar en otras tarjetas. Debido a esta característica, también es posible que cuenten con su propio *software* o archivos de biblioteca para facilitar la programación.

Lo anterior no significa que todas las tarjetas pertenecientes a esta categoría no puedan ser empleadas para el desarrollo de cualquier otra aplicación diferente para las que fueron diseñadas originalmente, aunque son más raras de encontrar. Así mismo, el costo de éstas suele ser mayor al de una tarjeta de desarrollo convencional.

Como ejemplo de una tarjeta de desarrollo dedicada se tiene la ESP-EYE *Development Board* (ver figura 1.19), la cual incorpora el SoC ESP32, un micrófono y una cámara de 2 megapíxeles. Esta tarjeta puede procesar audio, realizar reconocimiento de imagen y realizar la transmisión de imagen de manera inalámbrica mediante WiFi [59]. En la tabla *1-11* se muestran más de sus características. Otras tarjetas de este tipo se mencionan y describen en la tabla *1-12*.

Tabla 1-11 Características de la ESP-EYE[59].

Protocolos WiFi	**802.11b/g/n/e/i**
Cámara	2 megapíxeles
Micrófono	Micrófono digital
Flash/SRAM	4 MB/ 8 MB
Interfaces	I/O y USB

Figura 1.19 *ESP_EYE[59].*

Tabla 1-12 Tarjetas de desarrollo dedicadas[59].

Tarjeta de desarrollo	**Aplicación**	***Hardware* Integrado**
ESP32 - LyraT	Para aplicaciones de audio, con capacidad para su control remoto basado en comandos de voz	Salidas de audio, botones tipo *touch* para control de audio, 2 micrófonos, entre otros componentes.
ESP32-MeshKit-Sense	Sensado y detector de consumo energético de otras placas	Sensor de temperatura, sensor de humedad, sensor de luz de ambiente y conector para pantallas LCD

Capítulo 3

Descripción de la propuesta

3.1 Propuesta

Tras el surgimiento de nuevas tendencias tecnológicas como el IoT, se están imponiendo nuevos requisitos tanto en los servicios como productos, lo cual ha provocado que requieran en éstos una mayor integración de diversas tecnologías, haciéndolos cada vez más complejos. Estos dispositivos también deben ser de bajo costo y lo suficientemente flexibles para adaptarse a diferentes entornos e interactuar con otros dispositivos a su alrededor. Lo anterior ha generado la aparición de nuevas herramientas de desarrollo para la creación de prototipos de dispositivos que estas tendencias necesitan, los cuales deben permitir el uso de periféricos y sensores para interactuar con su entorno y a su vez, intercambiar información con otros dispositivos o con la propia internet [60].

Ante la diversidad de herramientas de desarrollo existentes en el mercado, es importante conocer las limitantes o restricciones de *hardware* o *software* que pueden imponer el uso de éstas y como impactan en el diseño de un prototipo. En el caso particular de las tarjetas de desarrollo se puede distinguir dos tipos, las que ya cuentan con las tecnologías de comunicación y las que no.

Las tarjetas de desarrollo que no cuentan con estas tecnologías tienen a su disposición a las denominadas *shield,* que son placas de circuito impreso que añaden a la tarjeta de desarrollo alguna funcionalidad extra, las cuales presentan el inconveniente de no ser compatibles con todas las tarjetas de desarrollo que están disponibles, ya que son

generalmente diseñadas para un modelo específico. Así mismo, su uso puede contraer limitaciones del *hardware,* ya que es posible que físicamente bloqueen el acceso a algunas terminales de la tarjeta, ésto no puede ser corregida debido a que sólo encajan en éstas de una manera específica, e implican un incremento en las dimensiones de la tarjeta. También está el hecho de que las *shield* surgieron como una solución temprana para atender las necesidades de las nuevas corrientes tecnológicas, por lo que varias *shield* ya se han discontinuado, complicando su adquisición.

En el caso del otro tipo de tarjetas, existe una gran variedad con distintos periféricos, pero suelen tener precios elevados, además de que cada familia de éstas utiliza un *software* diferente, por lo que el diseño de prototipos está restringido a una sola familia en específico. Estas familias no son lo suficientemente diversas para adaptarse de manera eficiente a los requerimientos de *hardware,* por lo que se podría pagar un precio mayor por una tarjeta de desarrollo de la cual no se requerirán todos o la gran mayoría de los periféricos que proporciona, lo cual contrasta con la tendencia de desarrollo de dispositivos de bajo costo, en caso contrario, el *hardware* que incorporan podría no satisfacer completamente las necesidades del usuario. De cualquier manera, el usuario está obligado a adaptarse a la distribución de los elementos y a las dimensiones que estas tarjetas presenten. Así mismo, es posible que el *software* que utilizan esté parcialmente restringido, en búsqueda de simplificar la programación de éstas, lo cual impide aprovechar todas las características del *software* que pueden ofrecer [60].

Recientemente han aparecido módulos WiFi, los cuales son usados de manera independiente como una opción en lugar de las tarjetas de desarrollo convencionales. Sin embargo, su *hardware* presenta algunos inconvenientes como requerir, en algunos casos, de circuitos para la utilización de sus periféricos, además dependiendo del módulo elegido, éste puede estar construido de manera tal, que físicamente están bloqueados algunos de los periféricos del SoC que incorpora. Respecto al *software,* el principal inconveniente que presentan es que al encargarse de las tareas de control y comunicación se pierde la ventaja de ejecución de tareas en forma paralela [60], además de que la elección del *firmware* se vuelve crucial, puesto que éste limita el número de características a las que se tendrán acceso.

Para no perder la posibilidad de ejecutar tareas de manera simultánea, estos módulos son usados como esclavos de un microcontrolador o una tarjeta de desarrollo para encargarse de las tareas de comunicación. Aunque el papel que toman es el mismo que la de una *shield*, éstos tienen la ventaja de que pueden trabajar con cualquiera de los antes mencionados, siempre y cuando éstos presenten una interfaz de comunicación compatible. Sin embargo, este uso no es común por la falta de *software* para este fin.

El presente trabajo busca brindar una opción en el desarrollo de prototipos mecatrónicos que requieran de comunicación inalámbrica o de participar en las nuevas tendencias como el IoT. Partiendo de la relevancia que tienen tanto los microcontroladores en el diseño de sistemas mecatrónicos como la tecnología WiFi en la actualidad, se optó por el desarrollo de una biblioteca que permita la conjunción de microcontroladores y módulos WiFi para el desarrollo de prototipos. La biblioteca ha sido nombrada *biblioteca ESP* y fue desarrollada para que microcontroladores PIC usen como esclavos a módulos WiFi que integren el SoC ESP8266 o los SoC ESP32, atendiendo así, la ausencia de *software* que impedía que estos elementos en conjunto fueran utilizados como una alternativa en el desarrollo de prototipos. La biblioteca ESP permite el uso de diferentes microcontroladores PIC y módulos WiFi, de tal forma que se proporcione una alternativa flexible en comparación con las existentes en el mercado. Además, se buscó aprovechar la mayoría de las características que los módulos WiFi ofrecen.

Para la biblioteca ESP se ha desarrollado un manual de prácticas que muestra los alcances y limitaciones del conjunto PIC-módulo WiFi, el cual puede ser utilizado como material didáctico en la carrera de Ingeniería Mecatrónica, en particular en la asignatura Circuitos Digitales que es donde se enseña el uso de microcontroladores PIC, permitiendo a sus estudiantes el desarrollo de aplicaciones inalámbricas o que requieran acceso a internet y que estén apegados a las necesidades de las nuevas tendencias.

3.2 Microcontroladores PIC

Se ha optado por el uso de microcontroladores PIC ya que éstos cuentan con una enorme variedad de familias que permiten adecuarse mejor a las necesidades de diferentes aplicaciones. Además, son utilizados ampliamente en la enseñanza de control digital, que a su vez son parte de la enseñanza de la mecatrónica. Este tipo de microcontroladores, que son desarrollados por *Microchip Technology Inc*, fueron diseñados inicialmente a finales de la década de 1970 bajo el nombre de Controlador de interfaz periferíca (*Peripheral Interface Controller,* PIC) y estaban destinados a aplicaciones de control simple. En la década de 1990 fue cuando estos microcontroladores comenzaron a ser populares, superando a varios de sus competidores gracias a su gran diversidad, junto con herramientas de desarrollo fáciles de usar [61]. Otros factores importantes que ayudaron a su consolidación son su buena relación entre precio/prestaciones, fácil programación, circuitos normalizados y una vasta documentación [62].

Una de las principales características que distinguen a estos microcontroladores es su enorme variedad de periféricos como: puertos de entrada/salida, temporizadores, comunicación serial síncrona y asíncrona, convertidores analógico a digital y digital a analógico, moduladores de ancho de pulso (*Pulse Width Modulation,* PWM), interrupciones, entre otros. La clasificación principal de los microcontroladores PIC está relacionada con el tamaño del *bus* de la memoria de datos que puede variar 8, 16 y 32 *bits*:

- Los PIC de 8 bits son el tipo de PIC más popular, que cuentan con gran diversidad de periféricos.
- Los PIC de 16 bits son microcontroladores de alto rendimiento y robustez, con bajo consumo energético a un costo accesible, son utilizados en aplicaciones que sobrepasan las capacidades de los PIC de 8 bits.
- Los PIC de 32 bits ofrecen alta capacidad de cómputo junto con una diversa conectividad, están destinados para aplicaciones de control de propósito general, aplicaciones seguras del IoT y aplicaciones de gráficos avanzados.

3.3 Módulos WiFi

En la actualidad existe en el mercado una gran variedad de módulos WiFi, los cuales se diferencian principalmente por el SoC que incorporan, el *hardware* que integran, la cantidad de memoria que poseen y el *firmware* que utilizan. En este caso, se optó por trabajar con los módulos WiFi que incorporaran alguno de los populares SoC ESP8266 y ESP32 fabricados por Espressif Systems, ya que empresas reconocidas en el ambito mundial como AI-Thinker Inc, Wemos, Olimex y SparkFun, fabrican sus módulos WiFi basados en estos SoC [46, 48] y por tanto, existe una mayor cantidad de módulos que pueden ser considerados para trabajar con los microcontroladores PIC. Esta variedad permite que se pueda seleccionar el conjunto microcontrolador PIC-módulo WiFi más adecuado para un proyecto en específico, ya que a pesar de que los módulos WiFi están basados en los mismos SoC, éstos no comparten las mismas características debido, en gran medida, al resto de elementos que se utilizan para su construcción.

Figura 2.1 *SoC ESP8266 [63].*

3.3.1 ESP8266

El SoC WiFi ESP8266 (ver figura 2.1) es un chip WiFi de un solo núcleo desarrollado por *Espressif Systems*, que, de acuerdo con el fabricante, características como eficiencia energética, confiabilidad y rendimiento que éste presenta le permiten trabajar en entornos industriales y satisfacer algunos de los requerimientos del IIoT. Este SoC está certificado por la *WiFi Alliance*, y por sus características es considerado un sistema de microcontrolador completo, el cual brinda todas las funcionalidades necesarias para

Tabla 2-1 Características del SoC ESP8266[63, 64].

Procesador	**32-bit RISC Tensilica Xtensa LX106**
Frecuencia de operación	80 *MHz* ~ 160 *MHz*
Memoria RAM	160 kB
Paquete	QFN 5x5 (*mm*) de 32 pines
Potencia de salida	+19.5 *dBm*
Corriente en estado de reposo y en operación	<20 *μA* 80 *mA*
Periféricos	UART x 2 GPIO x 17 I2C x 1 I2S x 2 SDIO x1 PWM x 4 ADC 10 bit x1 SPI /HSPI x 2 Control remoto por IR x 1
Voltaje de operación	2.5 *V* ~ 3.6 *V*
Protocolos WiFi	802.11 b/g/n (HT20)
Rangos de frecuencia WiFi	2.4 GHz ~ 2.5 GHz
Modos WiFi	Estación/SoftAP/ Estación+ SoftAP
Seguridad y encriptación	WPA/WPA2 WEP/TKIP/AES
Protocolos de red	IPv4, TCP/UDP/HTTP
Interfaces WiFi	2

realizar la comunicación mediante WiFi. Este SoC está integrado por unidades de memoria SRAM y ROM, un procesador de 32 bits Tensilica L106 de muy bajo consumo y alto rendimiento, amplificadores de potencia, así como por otros elementos que le permiten ser implementado de manera tal, que se requiere un mínimo número de circuitos externos para su uso [46, 49, 64]. En la tabla *2-1* se muestran algunas de sus características más sobresalientes.

3.3.2 ESP32

Los ESP32 en este momento engloban a una serie 4 SoC (ver tabla *2-2*) de ultra bajo consumo desarrollados por *Espressif Systems* y que de acuerdo con el fabricante, están diseñados para utilizarse en dispositivos móviles, portátiles y en aplicaciones IoT, capaces de funcionar de manera confiable incluso en entornos industriales y siendo su principal diferencia con respecto al SoC ESP8266, la integración tanto de WiFi como Bluetooth [65]. Cada SoC de esta serie está integrado por uno o dos microprocesadores de 32 bits Xtensa LX6 de bajo consumo con frecuencias de operación desde 80 *MHz* hasta 240 *MHz*, amplificadores de potencia y de recepción de bajo ruido, módulos de administración de energía entre otros, lo que ha permitido que estos SoC también sean considerados como un microcontrolador completo [46, 65].

Tabla 2-2 SoC de la serie ESP32[63, 65].

SoC	Núcleo	Memoria	Frecuencia de operación
ESP32-D0WD	Dual	520 k*B* SRAM 448 k*B* ROM	Hasta 240 *MHz*
ESP32-D0WDQ6	Dual	520 k*B* SRAM 448 k*B* ROM	Hasta 240 *MHz*
ESP32-D2WD	Dual	520 k*B* SRAM 448 k*B* ROM	Hasta 160 *MHz*
ESP32-S0WD	Simple	520 k*B* SRAM 448 k*B* ROM	Hasta 160 *MHz*

Figura 2.2 *SoC ESP32 D0WD[63].*

En la figura 2.2 se observa uno de los SoC pertenecientes a esta serie y en la tabla *2-3* se listan las principales características que comparten todos los SoC de la serie ESP32. Para mayor información sobre algún SoC en particular, se recomienda consultar [63].

Tabla 2-3 Características en común de los SoC ESP32[63, 65].

Procesador	**Xtensa LX6 de 32 bits**
Frecuencia de operación	80 *MHz* ~ 240 *MHz*
Memoria RAM	520 *kB* SRAM 448 *kB* ROM 16 *kB* SRAM en RTC
Paquete	QFN 5x5 (*mm*) y 6x6 (*mm*) para ESP32-D0WDQ6
Potencia de salida	+19.5 *dBm*
Periféricos	GPIO x 34 SAR ADC 12 bit x1 DAC 8 bits x 2 SPI x 4 I2C x 2 UART x 3 Interfaz Ethernet MAC x1 Hall sensor x1 GPIO de sensado capacitivo x 10 Control remoto por IR x 1 PWM
Aceleración de *hardware* criptográfico	AES, Hash (SHA-2), RSA, ECC
Voltaje de operación	2.5 *V* ~ 3.6 *V*
Bluetooth clásico	L2CAP, SDP, GAP, SMP, AVDTP, AVCTP, A2DP (SNK) y AVRCP (CT)
Bluetooth Low Energy (BLE)	L2CAP, GAP, GATT, SMP, y GATT
Modo WiFi	Estación, SoftAP con soporte para uso simultaneo de Estación, SoftAP y modo promiscuo
Protocolos WiFi	802.11 b/g/n
Interfaces WiFi	4

3.3.3 Módulos compatibles

Debido a la popularidad del SoC ESP8266 y los ESP32, existe actualmente una gran variedad de *firmware* para los módulos que incorporan a estos SoC. Los *firmware* han sido diseñados para facilitar la programación de los módulos WiFi con diferentes plataformas y lenguajes de programación como Lua, JavaScript, ESP8266 Basic, AT, C++, MicroPython, Arduino, Mongoose OS, DeviceHive ESP8266 y Q-IOT. El acceso de ciertas caracteriticas de los SoC, en especial respecto a *software* depende del *firmware* que se utilice, no obstante, la mayoría de los módulos WiFi que incorporan estos SoC actualmente soportan cualquier *firmware* que esté disponible, independientemente del fabricante y, por tanto, el *hardware* de éstos es el que toma una mayor relevancia para su elección. Sin embargo, Espressif Systems aún recomienda que se utilice el *firmware* desarrollado por el fabricante del módulo o tarjeta de desarrollo que integre alguno de sus SoC [66]. Por lo anterior, se optó por utilizar un *firmware* de Espressif Systems para aprovechar todas las características de estos SoC y, en consecuencia, se eligieron únicamente los módulos desarrollados por dos fabricantes.

El primer fabricante es el propio Espressif Systems por lo que sus módulos WiFi y tarjetas de desarrollo, resultan ser compatibles para el conjunto microcontrolador PIC-Modulo WiFi propuesto. En la tabla *2-4* se describen las principales características de los módulos WiFi que integran el SoC ESP8266. Para mayor información sobre éstos, consultar [51]. En la tabla *2-5* se describen las principales características de los módulos WiFi que integran algún SoC ESP32. Para mayor información sobre éstos consultar [51].

Tabla 2-4 Módulos ESP8266 compatibles[51].

Módulo ESP	Tarjeta de desarrollo	Dimensiones en *mm*	Pines	FLASH (MB)	Antena
WROOM-02D	ESP8266-DevKitC	18x20x3.2	18	2	PCB
WROOM-02U	ESP8266-DevKitC	18x14.3x3.2	18	2	IPEX
WROOM-02	ESP-Launcher	18x20x3	18	2	PCB
WROOM-S2	N/A	16x23x3	20	2	PCB

Tabla 2-5 Módulos ESP32 compatibles[51].

Módulo ESP32	Tarjeta de desarrollo	Dimensiones en *mm*	Pines	FLASH (MB)	Antena
Módulos con doble núcleo, con WiFi y Bluetooth modo dual					
WROOM-32D (SoC D0WD)	ESP32 DevKitC	18x25.5x3.1	38	4	PCB
WROOM-32U (SoC D0WD)	ESP32 DevKitC	18x19.2x3.2	38	4	IPEX
WROOM-32 (SoC D0WDQ6)	ESP32 DevKitC	18x25.5x3.1	38	4	PCB
WROVER-B (SoC D0WD)	ESP32 DevKitC ESP WROVER-KIT-VB ESP WROVER-KIT	18x31.4x3.3	38	4	IPEX
WROVER-IB (SoC D0WD)	ESP32 DevKitC ESP WROVER-KIT-VIB	18x31.4x3.3	38	4	PCB
WROVER (SoC D0WDQ6)	ESP-WROVER-KIT ESP32-LyraT ESP32-LyraTD-MSC	18x31.4x3.3	38	4	PCB
WROVER-I (SoC D0WDQ6)	N/A	18x31.4x3.3	38	4	IPEX
Módulos con un núcleo, con WiFi y Bluetooth modo dual					
SOLO-1 (SoC S0WD)	ESP32 DevKitC	18x25.5x3.1	38	4	PCB

El segundo fabricante es AI-Thinker Inc que cuenta quizá con la mayor variedad de módulos WiFi, basados principalmente en el SoC ESP8266, los cuales pueden utilizar sin ningún problema los *firmware* desarrollados por Espressif Systems. En el siguiente enlace se puede obtener más información sobre los módulos de este fabricante:

https://www.ai-thinker.com/product/overview

En la tabla *2-6* se describen algunos de los módulos más populares de AI-Thinker basados en el ESP8266. Es importante señalar que cualquier módulo que soporte los *firmware* desarrollados por Espressif Systems es compatible con el conjunto propuesto.

Tabla 2-6 Módulos de la familia ESP8266 [48, 67].

Módulo	Número total de pines	Antena	Dimensiónes en *mm*	Notas
ESP-01	8	Grabada en PCB	14.3 x 24.80	Éste es el módulo más común, existen múltiples variantes.
ESP-02	8 conexiones de superficie, sin embargo, es posible soldar terminales	Conector UFL	14.2 × 14.2	Existen múltiples variantes.
ESP-03	14 conexiones de superficie	Cerámica	17.3 × 12.1	Éste es el módulo más popular y utilizado en internet
ESP-04	14 conexiones de superficie	Ninguna	14.7 × 12.1	Tipo de antena personalizable por el usuario.
ESP-05	5	Conector UFL	14.2 × 14.2	Existen múltiples variantes
ESP-06	12 conexiones bajo la placa	Ninguna	14.2 × 14.7	El escudo de metal dice contar con certificación FCC Ce
ESP-07	16 conexiones de superficie; otras versiones pueden incluir 14	Ambas, cerámica y conector UFL	20.0 × 16.0	Algunas versiones tienen un error en serigrafía, la tapa dice contar con certificación FCC Ce.
ESP-08	16 conexiones de superficie; otra versión puede incluir 18	Ninguna	17.0 × 16.0	Igual al ESP-07, pero con antena personalizable por el usuario.
ESP-09	18 conexiones bajo la placa; otras versiones pueden incluir 12	Ninguna	10.0 × 10.0	Es el módulo más pequeño del mercado, 1 MB de memoria flash.
ESP-10	5 conexiones de superficie	Ninguna	14.2 × 10.0	Solo interfaz UART
ESP-11	5 conexiones de superficie; otras versiones pueden incluir 8	Cerámica	17.3 × 12.1	Poca información disponible
ESP-12	16 conexiones de superficie	Grabada en PCB	24.0 × 16.0	El escudo de metal dice contar con certificación FCC, posee 4 MB de memoria flash.
ESP-12E ESP-12F	22 conexiones de superficie	Grabada en PCB	24 x 16	----------
ESP-13	18 conexiones de superficie	Grabada en PCB	18 x 20	----------
ESP-14	22 conexiones de superficie	Grabada en PCB	24.3 x 16.2	Incluye un STM8

3.4 *Firmware* del módulo WiFi

3.4.1 Comandos AT

La utilización de los *firmware* desarrollados por Espressif Systems permite aprovechar todas las características de los SoC de este mismo fabricante, siendo también más probable que éstos tengan un mayor soporte tanto en la corrección de errores como en el lanzamiento de nuevas características. Actualmente, Espressif Systems brinda una gran variedad de *firmware* dependiendo de las necesidades del usuario; sin embargo, no todos pueden ser utilizados indistintamente en el SoC ESP8266 o ESP32, debido a las características que éstos presentan.

En este caso se optó por el *firmware* de comandos AT desarrollado por Espressif Systems, principalmente porque éste está disponible tanto para el SoC ESP8266 como los ESP32. Este *firmware* provee acceso a una gran cantidad de APIs de estos SoC, lo cual permite configurar, controlar y llevar a cabo tareas de comunicación en un módulo WiFi mediante comandos AT (comandos Hayes). Existen dos versiones de este *firmware,* las cuales se diferencian entre sí por la base de *software* sobre la que se desarrollaron.

La primera versión es la del *firmware* de comandos AT que está construido sobre un Kit de desarrollo de *software (Software Development Kit,* SDK), es decir, un conjunto de controladores, bibliotecas precompiladas y APIs del dispositivo que se encargan de toda la configuración y permiten desarrollar fácilmente aplicaciones dirigidas principalmente para el IoT. Dependiendo de si las aplicaciones desarrolladas están basadas en un sistema operativo, el SDK se clasifica en dos tipos, cambiando la forma y estructura de programación:

- Sin sistema operativo (*non Operating System,* NONOS SDK)
- Sistema operativo en tiempo real (*Real Time Operating System,* RTOS SDK).

El *firmware* de comandos AT basado en NONOS está compuesto por un conjunto de APIs que permiten acceso y uso total de las características del SoC sin tener que conocer la arquitectura de éste, siendo una de sus principales características, el uso de temporizadores para la ejecución programada de funciones, así como de las denominadas *callback* (funciones de llamada de vuelta), es decir, funciones que tras la ocurrencia de un evento son llamadas desde el núcleo del SDK, permitiendo realizar acciones en tiempo real. El término NONOS hace referencia a que no está basado en un sistema operativo y, por tanto, está pensado principalmente el desarrollo de aplicaciones por eventos, en las cuales el usuario requiera control total sobre la secuencia de ejecución del código.

Así mismo, este tipo de SDK se caracteriza por el uso de la interfaz de red espconn (*espconn network interface)* que simplifica el desarrollo de aplicaciones de red [66, 68, 69], al tener únicamente que declarar y programar el código de las funciones que se desee ejecutar para cada evento posible, procedimeinto muy similar a la programación de una interrupción de un micrcontrolador PIC.

La segunda versión del *firmware* de comandos AT, que es la más reciente, está basada en el denominado Marco de desarrollo IoT de Espressif (*Espressif IoT Development Framework*, ESP-IDF) o mejor conocido como *IDF Style*, un entorno de desarrollo basado en FreeRTOS, un sistema operativo multitarea en tiempo real, el cual permite un rápido desarrollo de aplicaciones IoT [70] y está diseñado principalmente para los módulos de Espressif Systems, aunque ya se ha probado con éxito en los módulos de AI-Thinker.

Ambas versiones pueden ser descargadas de manera gratuita desde la página oficial de Espressif Systems:

https://www.espressif.com/en/support/download/at

Si bien entre estas versiones no existe una diferencia notable para el usuario ya que comparten la mayoría de comandos AT, en la versión ESP-IDF algunos de éstos difieren en la cantidad de parámetros, por lo que la versión ESP-IDF no puede ser tratada como una actualización de *firmware* para los proyectos desarrollados en la versión SDK NONOS [71]. Sin embargo, ésta no es la única diferencia entre estas versiones, la versión basada en el SDK NONOS es compatible únicamente con los módulos WiFi que incorporen el SoC ESP8266, mientras que la versión basada en IDF es compatible con los módulos que incorporen el SoC ESP8266 o los ESP32. También existe una diferencia respecto a las APIs a las que se tienen acceso de los SoC; derivado del tiempo de desarrollo de ESP-IDF respecto a SDK NONOS, así como en la sintaxis y estructura de las funciones para el desarrollo de nuevos comandos AT. Ambas versiones permiten la utilización de varias interfaces de comunicación para los comandos AT como UART, Socket, SDIO y HSPI (ESP-IDF).

Aunque Espressif Systems anunció que dejarían de desarrollar comandos AT para la versión basada en el SDK NONOS y únicamente brindaría soporte para corrección de errores por un tiempo limitado [71], se optó por la utilización de ambas versiones con objeto de que más módulos WiFi fueran compatibles con el conjunto PIC-módulo WiFi propuesto ya que ESP-IDF requiere de una mayor cantidad de memoria en los ESP8266 que algunos módulos no satifacen, además de que al encontrarnos en la etapa de transición, la versión ESP-IDF para el ESP8266 aún cuenta con menos APIs respecto a NONOS.

También contribuyó a la elección del *firmware* de comandos AT, el que éste permite al microcontrolador PIC llevar un control total sobre la secuencia de configuración del módulo y en caso de un error, el microcontrolador es capaz de detectar y realizar acciones concretas ante ésto, como evitar seguir con la configuración. Esta elección también estuvo influenciada por otros factores, entre los que se encuentran:

- El uso de comandos AT simplifica la comunicación de los elementos, ya que pone al módulo WiFi a la espera de instrucciones por parte del uC PIC. Si se eligiera otro *firmware*, se tendría que desarrollar toda la estructura de comunicación para reconocer las instrucciones provenientes del uC PIC y su posterior ejecución, generando una estructura similar a la que presenta ya el *firmware* de comandos AT.
- Aunque existe una diferencia entre las versiones del *firmware* de comandos AT, un análisis exhaustivo mostró que las diferencias radican en los parámetros de ciertos comandos, así como en comandos que no aparecen en alguna de las versiones. Sin embargo, estas diferencias no resultaron ser cruciales para el desarrollo de las aplicaciones que puedan ser usadas en ambas versiones del *firmware*.
- Como el *firmware* de comandos AT puede utilizarse en los módulos que integren tanto al SoC ESP8266 como los SoC ESP32, facilitó el desarrollo del *software* en el lado del PIC al utilizar la misma estructura y funciones, lo cual se traduce en tiempos menores para depuración y desarrollo, caso contrario al utilizar *firmware* distintos.
- Si bien el *firmware* de comandos AT, independientemente de la versión, sólo ofrece acceso a algunas características, éste provee una estructura para crear nuevos comandos AT mediante el SDK NONOS o ESP-IDF, por lo que es posible acceder a más características que poseen los SoC.

3.4.2 Implicaciones y requerimientos en los microcontroladores PIC

La elección del *firmware* de comandos AT tiene implicaciones en el microcontrolador PIC, siendo quizá la más sobresaliente la cantidad de memoria que se requiere en éste para almacenar los comandos AT, ya que el *software* desarrollado en el microcontrolador PIC deberá gestionar la ejecución de dichos comandos y las funciones asociadas a éstos. Así mismo, la utilización del *firmware* de comandos AT implica la utilización de la interfaz de comunicación serial UART como primera opción, ya que la gran mayoría de los módulos vienen diseñados para implementar este tipo de comunicación, habilitando el acceso a dichas terminales. Lo anterior no resulta contraproducente, ya que esta interfaz está presente en una gran cantidad de microcontroladores PIC [14] y las velocidades son compatibles con las que presentan los módulos WiFi.

Teniendo en cuenta lo anterior, es necesario que al menos el microcontrolador PIC cuente con las siguientes características para que puedan establecer una comunicación adecuada con los modulos WiFi:

- Oscilador que permita alcanzar una velocidad de transmisión de 115.200, dicha velocidad es la comúnmente preestablecida en estos módulos.
- Interrupción por la recepción de datos (serial), ya que existen comandos AT cuya respuesta es entregada en un cierto tiempo tras su ejecución y no es deseable detener la ejecución del programa esperando dicha respuesta.
- Temporizador e interrupción asociada a éste, necesario para atender respuestas de los comandos AT y determinar si existió un problema durante la comunicación como el no obtener una respuesta.

3.5 Circuito base

Independientemente del microcontrolador PIC y el módulo WiFi que se puedan emplear, existe un circuito base que muestra las conexiones mínimas entre ambos elementos para que éstos puedan interactuar de manera correcta y que está diseñado considerando que tanto el microcontrolador PIC como el módulo WiFi trabajen a 3.3 *V* (ver figura 2.3). En este circuito, el consumo de corriente dependerá tanto del módulo WiFi como del uC PIC empleados. En la misma figura del circuito base se pueden observar las seis partes que constituyen a éste, las cuales están señaladas mediante números romanos.

La primera parte del circuito base (cuadro naranja) hace referencia a una etapa de regulación de voltaje utilizada para suministrar el voltaje requerido por el conjunto microcontrolador PIC-módulo WiFi. Esta parte es opcional ya que, si la fuente otorga el voltaje requerido, ésta no es necesaria. La segunda parte del circuito base (cuadro verde olivo) hace referencia a las conexiones mínimas del módulo WiFi como lo son sus terminales de alimentación las cuales operan generalmente a 3.3 *V*. Algunos módulos proveen acceso a una terminal del SoC llamada CHIP_EN en el ESP8266 o CHIP_PU en los ESP32, la cual permite encender o apagar el SoC y cuya conexión siempre debe ser a 3.3 *V* a través de un ressistor.

Dependiendo del módulo empleado, puede que sea necesario conectar algunos de sus GPIO a ciertos niveles de voltaje durante el encendido, para que éste opere de manera correcta. En la tabla *2-7* se muestran los niveles de voltaje de los GPIO para los módulos que incorporan un SoC ESP8266 o los SoC ESP32. Así mismo, por modificaciones realizadas en el *firmware* de comandos AT que se explicarán más adelante, el GPIO identificado como GPIO UPG en el circuito base debe estar conectado mediante un

resistor a VCC, el cual corresponde al GPIO 2 para la versión basada en SDK NONOS y ESP-IDF ESP8266, mientras que para la versión basada ESP-IDF ESP32 corresponde por defecto al GPIO 19.

Tabla 2-7 Niveles de voltajes durante el encendido del módulo ESP[72].

SoC	**GPIO**	**Nivel de voltaje**
ESP8266	0	Alto/VCC (con un resistor)
ESP8266	2	Alto/VCC (con un resistor)
ESP8266	15	Bajo /GND (directamente o con un resistor)
ESP32	0	Alto/VCC (con un resistor)

La tercera parte del circuito base (cuadro verde) hace referencia a las conexiones de la interfaz de comunicación serial UART entre el módulo WiFi y el microcontrolador PIC, en la cual se conectan las terminales TX y RX del módulo WiFi a las terminales RX y TX del uC PIC respectivamente, y entre ellas está un interruptor DPST para desconectar estas terminales cuando se actualiza el programa que incorpora el uC PIC a traves del puerto serial. Aunque también este interruptor puede ser remplazado por un botón en configuración *pull-up* conectado a CHIP_EN para apagar el SoC durante el mismo proceso o modificar el *bootloader* del uC PIC para realizar el funcionamiento del botón a través de un GPIO del microcontrolador

Cuando el uC PIC y el módulo WiFi operan al mismo voltaje, estas terminales pueden conectarse de manera directa, en caso contrario debe utilizarse un circuito de conversión de voltaje entre éstas, como los mostrados en la figura 2.4. El circuito mostrado en la parte inferior de la figura 2.4 es un divisor de voltaje que se propuso para la conversión de voltaje de 5 *V* a 3.3 *V,* mientras que el circuito mostrado en la parte superior es un circuito propuesto por el fabricante AI-Thinker Inc para estos mismos niveles de voltaje en [73]. Ambos circuitos pueden ser usados indistintamente.

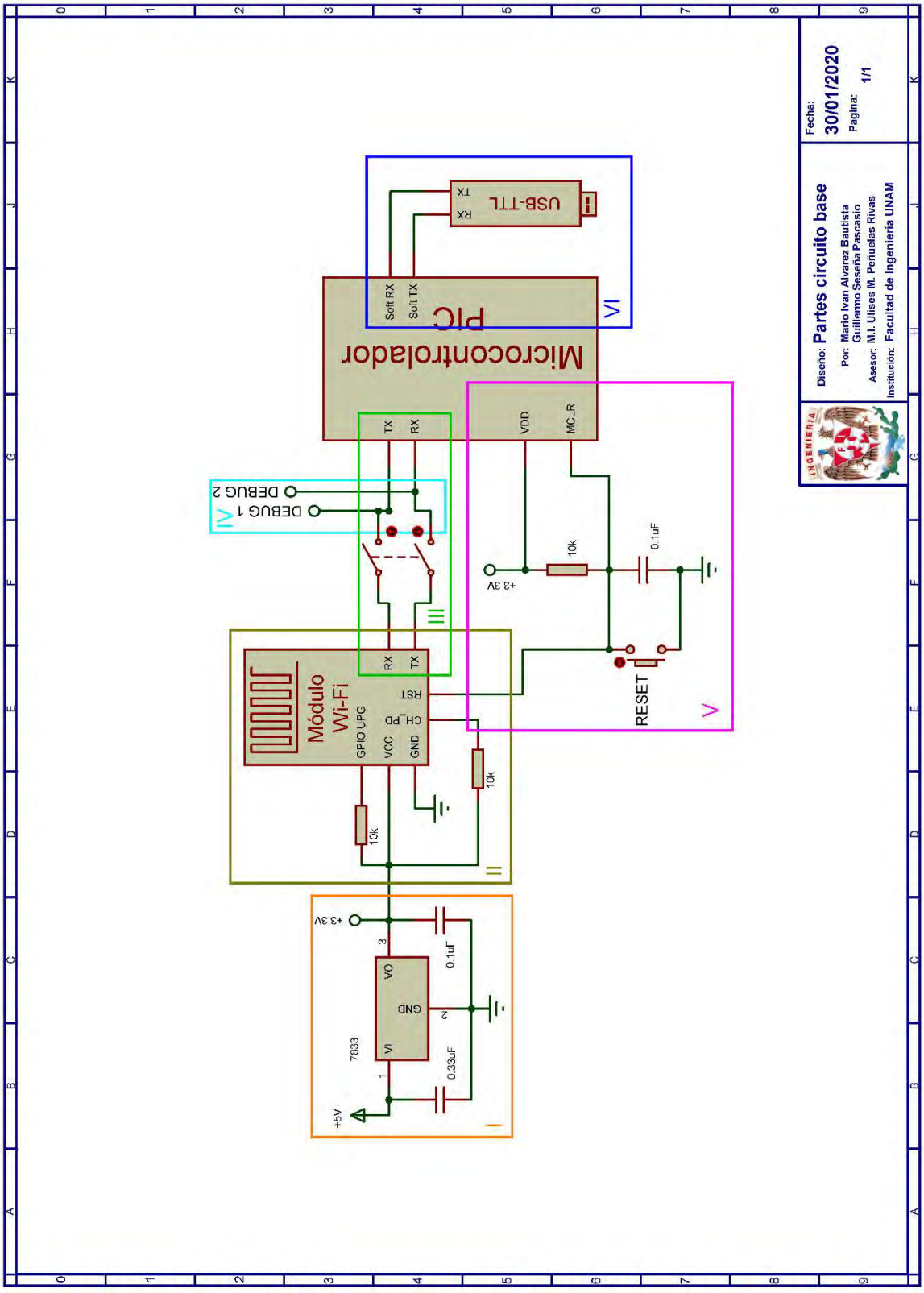


Figura 2.3 *Circuito base y las partes que lo constituyen.*

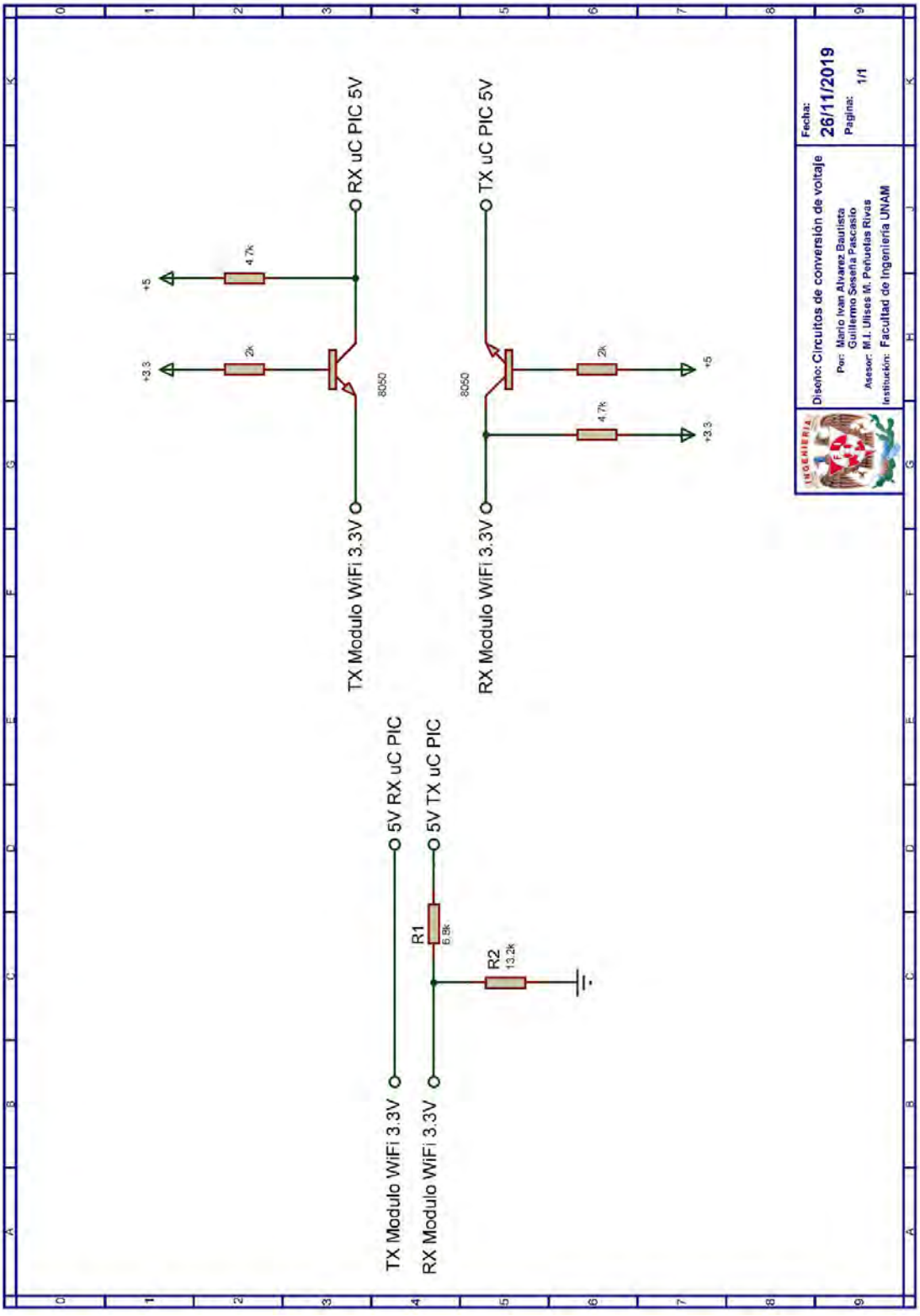


Figura 2.4 *Circuitos de conversión de niveles de voltaje.*

La cuarta parte del circuito base (cuadro azul celeste) hace referencia a un convertidor serial que se conecta, ya sea en el canal de transmisión o recepción del UART, para monitorear el intercambio de mensajes que se da en el conjunto PIC-módulo WiFi. Este convertidor es utilizado para depurar errores, ya que se observarán tanto los comandos que se envían al módulo WiFi (canal de transmisión uC PIC) y la respuesta completa del módulo ante éstos (canal de recepción uC PIC), permitiendo obtener información más detallada del origen de los errores.

La quinta parte del circuito base (cuadro rosa) es un circuito de un botón en configuración *pull-up* que reinicia tanto al módulo WiFi como al uC PIC de manera simultánea. Este botón resulta necesario debido a que existen comandos AT que por sus características no permiten la ejecución de otro comando AT hasta la ejecución completa de sus tareas, lo cual puede provocar que el microcontrolador PIC pierda el control de la configuración del módulo WiFi, al reiniciarse únicamente el microcontrolador. Para realizar el reinicio, es necesario mantener presionado el botón al menos tres segundos.

La sexta parte del circuito base (cuadro azul rey) tiene la intención de señalar la existencia de un módulo de comunicación serial UART creado por *software* al usuario, ya que el de *hardware* es utilizado por el módulo WiFi. Las terminales de comunicación de este UART pueden ser establecidas en el archivo de cabecera *esp.h* de la biblioteca ESP.

Capítulo 4

Desarrollo de *firmware*

Para aprovechar las características de los microcontroladores PIC y los módulos WiFi que componen al conjunto propuesto, se requiere crear *software* en ambos lados. En el caso del microcontrolador, el *software* creado corresponde a la biblioteca ESP que posibilita el control de los módulos WiFi, permitiendo a estos microcontroladores utilizar redes WiFi para su comunicación, así como acceso a las APIs que ofrecen los módulos. En el caso del *software* creado para los módulos WiFi, éste se presenta en forma de nuevos comandos AT que permiten acceder a características y APIs que no están disponibles en las versiones del *firmware* de comandos AT que proporciona el fabricante, lo que amplía el catálogo de aplicaciones del conjunto microcontrolador PIC-módulo WiFi. A continuación, se detallan las características sobre el *software* creado para ambas partes.

4.1 Código indirecto

Los *firmware* de comandos AT ofrecen el acceso a diferentes funcionalidades del SoC que incorporan los módulos WiFi. El número de comandos disponibles para los módulos WiFi, y por tanto, las funciones a las que se tiene acceso dependen de la versión del *firmware,* la memoria disponible y del SoC, que suelen estar clasificados en:

- Básicos. Relacionados principalmente con el control y configuración de periféricos y del funcionamiento del SoC.

- WiFi. Relacionados principalmente con la capa de enlace.
- TCP/IP. Relacionados principalmente con la capa de transporte.
- BLE. Relacionados principalmente con la configuración y uso de Bluetooth.
- Particulares. Desarrollados para implementar aplicaciones específicas como MQTT, Ethernet, HTTP, WPA2, etc.

Aun así, los *firmware* de comandos AT no permiten explotar todas las características que ofrecen el SoC ESP8266 y los ESP32, ya que fueron desarrollados con fines demostrativos e introductorios [60]. Es por ello, que se optó por desarrollar nuevos comandos AT, aprovechando la facilidad que ofrecen los *firmware* en este sentido y con el objetivo de:

- Incrementar el número de funcionalidades a las que se tiene acceso.
- Brindar herramientas para el desarrollo de aplicaciones dentro de un campo específico, más que para una aplicación final.

El desarrollo *software* del lado del módulo permite utilizar los recursos de éste que eran desaprovechados en el *firmware* original de comandos AT, y simplifica el desarrollo de *software* en el lado del microcontrolador PIC, al personalizar las respuestas de los nuevos comandos AT y delegar la ejecución de algunas tareas al módulo WiFi. Aunque las aplicaciones para las que se crearon estos nuevos comandos AT no son las únicas que se pueden desarrollar, éstas resultan ser lo suficientemente complejas para ilustrar cómo acceder a nuevas funcionalidades del módulo, realizar modificaciones en las bibliotecas incorporadas al *firmware* y agregar nuevos comandos AT, tanto para la versión del *firmware* basada en SDK NONOS como la basada en ESP-IDF.

Ambas versiones del *firmware* fueron modificadas para hacer uso de dos GPIO del módulo WiFi, que no podrán ser utilizados por el usuario, para realizar la actualización de manera inalámbrica del programa de un microcontrolador PIC (consultar Apéndice C). En la versión basada en SDK NONOS estos GPIO corresponden al GPIO 0 y GPIO 2, mientras que en la versión basada en ESP-IDF existe la posibilidad de establecer estos GPIO en el archivo *user_def.h* mediante las definiciones UPGRADE_GPIO1 y UPGRADE_GPIO2. Por defecto, los GPIO asignados para la versión ESP-IDF corresponden al GPIO 18 y GPIO 19 para los ESP32, mientras que para el ESP8266 se siguen respetando los GPIO de la versión SDK NONOS. Cuando no se requiere iniciar por *hardware* la actualización del programa de un uC PIC el GPIO 2 en SDK NONOS o el GPIO 19 en ESP-IDF deberán estar conectados mediante un resistor a VCC.

Debido a las modificaciones realizadas, es necesario actualizar el *firmware* del módulo con las versiones modificadas para tener acceso a todas las funcionalidades desarrolladas en la biblioteca ESP. A continuación, se detallan las modificaciones realizadas en ambas versiones del *firmware* de comandos AT.

4.1.1 Nuevos comandos AT en SDK NONOS

Para esta versión del *firmware* de comandos AT se desarrollaron 49 nuevos comandos para 13 aplicaciones (consultar Apéndice E para más detalles), las cuales se describen a continuación:

- Comandos RSSI (Indicador de intensidad de señal recibida, Received Signal Strength Indicator)
  Comandos creados para el desarrollo de aplicaciones que utilicen como variable principal el valor RSSI, que permite el desarrollo de aplicaciones para estimación de distancia, localización en interiores, distinción de objetos, entre otros.

- Comando *upgrade server* (servidor de actualización)
  Comando creado para iniciar por *software* la actualización de manera inalámbrica del programa de un microcontrolador PIC mediante una arquitectura cliente-servidor.

- Comandos segundo servidor TCP
  Comandos creados para permitir la creación de un segundo servidor TCP (habilitar un puerto extra) en el módulo WiFi.

- Comandos MDNS
  Comandos creados para la utilización de MDNS, que requirió la modificación del *firmware* (consultar Apéndice C para más detalles) permitiendo:
  - Reclamar un *hostname* (nombres únicos de dispositivos), realizando las etapas de *probing* y *announcing* (sondeo y anuncio).
  - Anunciar hasta dos servicios MDNS.
  - Establecer un TTL (Tiempo de vida, *Time to live*) personalizado.
  - Añadir hasta dos descripciones por servicio anunciado.
  - Consultar la dirección IP de un *hostname.*
  - Dejar un grupo MDNS notificando, o no, a los participantes

- Comandos *simple pair*
  Comandos creados para la utilización de la API simple pair de los SoC de Espressif Systems.

- Comandos espnow
  Comandos creados para la utilización de la API espnow de los SoC de Espressif Systems.

- Comandos tracking (rastreo WiFi)
  Comandos creados para la utilización del módulo WiFi en un modo conocido como promiscuo para capturar el tráfico alrededor del módulo con la finalidad de rastrear o detectar la presencia de otros módulos WiFi que emitan un paquete *probe request* (sonda de solicitud). En este caso, únicamente se detectarán los *probe request* cuyo SSID sea ESP8266.

- Comandos servidor SSL
  Comandos creados para implementar un servidor SSL en el módulo WiFi, con la posibilidad de establecer un *timeout* (tiempo de espera máximo) al cliente que se conecte.

- Comandos *fingerprint*
  Comandos creados para autentificar a un servidor SSL durante la conexión mediante fingerprint. Para la creación de estos comandos se modificaron los archivos de la biblioteca mbedtls del *firmware* (Para más información consultar Apéndice C).

- Comandos criptográficos
  Comandos creados para la utilización de los métodos de cifrado y descifrado que proporciona la biblioteca mbedtls. Para la creación de estos comandos, se realizaron modificaciones en la biblioteca mbedtls.

- Comandos PWM
  Comandos creados para utilizar 4 canales PWM de los SoC que incorporan los módulos WiFi

- Comandos *status LED*
  Comandos creados para notificar el éxito o fracaso durante la conexión a un punto de acceso. Utiliza un GPIO del módulo WiFi que se emplea.

- Comandos cliente especial
  Comandos creados para permitir la creación de un cliente SSL o TCP con un filtro aplicado a los mensajes que éste recibe. Este filtro es utilizado para reducir la cantidad de información que recibe el microcontrolador PIC del módulo WiFi.

No todas las aplicaciones anteriormente listadas son accesibles para los módulos que utilicen el *firmware* de comandos AT basado en el SDK NONOS. Sólo la versión más

reciente del *firmware* de comandos AT v1.7.3.0, contenida en el SDK NONOS v3.0.3, incluye todas las aplicaciones anteriormente listadas, pero exige un mapa de memoria flash de 1024 *kB* + 1024 *kB.* Mientras que para los módulos WiFi con un mapa de memoria flash de 512 *kB* + 512 *kB* se utiliza una versión previa, es decir, la versión AT v1.6.2 contenida en el SDK v2.2.1, la cual se modificó para que también utilice la biblioteca *mbedtls*. La razón de emplear aún la versión previa es para utilizar módulos con memoria de 8 Mbit, que generalmente no poseen el mapa de memoria requerido en la nueva versión y es por esta limitación de memoria que, en la versión previa, a pesar de tener la capacidad, no se incluye:

- Comandos PWM.
- Comandos *status* LED.
- Comandos cliente especial.
- MD4, *camellia*, aes cfb128 y aes cfb 8 en los comandos criptográficos.

Originalmente, las versiones SDK NONOS v3.0.3 y SKD NONOS V2.2.1 presentan diferencias como lo son la corrección de algunos errores e inclusión de nuevas APIs, para más detalles consultar [74]. A pesar de que la versión SDK NONOS del *firmware* ha dejado de ser actualizada, es completamente funcional y más intuitiva en las APIs de red para el desarrollo de nuevos comandos. Al ya no ser actualizada, permite una mayor flexibilidad para realizar modificaciones en los archivos de ésta y en los archivos de sus bibliotecas.

4.1.2 Nuevos comandos AT en ESP-IDF

A diferencia de la versión basada en SDK NONOS, esta versión permite una mayor libertad en cuanto al uso de las funcionalidades del SoC y presenta una mayor cantidad de éstas. Para esta versión del *firmware* de comandos AT, se desarrollaron 42 nuevos comandos en el ESP32 y 41 comandos en el ESP8266 para 10 aplicaciones (consultar Apéndice E), ambas sobre la versión del núcleo de comandos AT 9a63a027, las cuales fueron desarrolladas buscando homogeneidad con la versión SDK NONOS, ya que actualmente ESP-IDF no cuenta con algunas APIs que están presentes en la versión SDK NONOS. Las aplicaciones desarrolladas se describen a continuación:

- Comandos RSSI.
- Comandos segundo servidor TCP.
- Comandos *espnow*.
- Comando *upgrade server*.
- Comandos criptográficos (para las modificaciones consultar Apéndice C).
- Comandos GPIO, comandos creados para controlar los GPIO del módulo WiFi empleado.
- Comandos ADC, basado en ADC1 en el ESP32.

- Comandos PWM basado en el periférico LED control en el ESP32
- Comandos MDNS, sin realizar modificación alguna de los archivos LWIP.
- Comandos *tracking*.

A diferencia de la versión SDK NONOS, ESP-IDF está en continuo desarrollo por lo que no resulta conveniente realizar modificaciones a los archivos de éste, como lo pueden ser los archivos de las bibliotecas LWIP o MBEDTLS, ya que éstos tendrían que ser replicados en cada versión, considerando los nuevos cambios que se presenten en las funciones de éstas.

4.1.3 Creación de comandos AT

La creación de nuevos comandos AT tiene el objetivo de ampliar el número de APIs a las que se tiene acceso y el desarrollo de aplicaciones generales o específicas en torno a éstas, liberando la ejecución de ciertas tareas o incluso del manejo de datos al microcontrolador PIC, aprovechando las características del procesador que incorporan al SoC ESP8266 y los ESP32. La forma en que se crearon los comandos AT para cada una de las versiones del *firmware* se detalla en el Apéndice B.

4.2 Biblioteca ESP

4.2.1 Generalidades

La biblioteca se diseñó para el compilador *CCS Compiler* que presenta las siguientes características:

- Utiliza lenguaje C, incluyendo instrucciones de preprocesador.
- Soporte para las familias: PIC10/12/16 con instrucciones de 12 bits y 14 bits; PIC10/12/16 con instrucciones de 14 bits; PIC18 con instrucciones de 16 bits; y PIC24/dsPIC con instrucciones de 24 bits.
- Permite migrar programas con facilidad entre los microcontroladores PIC.
- Integra diversas bibliotecas.
- Fácil manejo de interrupciones.
- Es uno de los compiladores más empleados en microcontroladores PIC, existiendo una gran comunidad que comparte y apoya el desarrollo de código.

Aunque este compilador trabaja con el lenguaje C, no está basado en ANSI C que es el estándar para dicho lenguaje, por lo que el código que se desarrolla en éste no es portable hacia otros compiladores, con ello presenta diferencias en el sistema de datos, formato de

almacenamiento (*Endianness*) y uso de varias funciones. En consecuencia, se requiere hacer modificaciones en la biblioteca ESP para que ésta pueda ser migrada a otro compilador.

Generalmente, las bibliotecas para los microcontroladores suelen ser un archivo con extensión (.a) o (.lib) dependiendo del compilador que se utilice, creado a partir de la colección de archivos objeto generados tras la compilación de uno o varios archivos fuente (ver figura 3.1) [75].

Figura 3.1 *Proceso de generación de una biblioteca [75].*

Desafortunadamente, el compilador CCS C no permite la creación de este tipo de archivos, por lo que para la biblioteca ESP se ha recurrido al diseño bajo el esquema de compilación separada, en la que, para reutilizar parte del código de un programa, se debe separar en una entidad independiente llamada módulo [76], el cual está compuesto principalmente por dos archivos:

- Archivo .h, es el encabezado que contiene información pública para la utilización del módulo, incluyendo macros, prototipos de funciones, constantes y variables globales.
- Archivo .c, en el que están definidas las funciones del módulo.

Para utilizar estos módulos, éstos deben ser incluidos en el programa principal refiriendo a sus archivos de cabecera. Conforme a lo anterior, la biblioteca está conformada por los siguientes archivos:

- *esp.c*, en el que se encuentran definidas todas las funciones disponibles de la biblioteca.
- *esp.h*, que es el archivo de cabecera, en que se ajustan los parámetros necesarios para utilizar la biblioteca, además de una sección para la habilitación de las aplicaciones que se utilizarán.

La biblioteca ESP contiene código para desarrollar diferentes aplicaciones, las cuales podrían ser agrupadas en diferentes módulos; sin embargo, dichas aplicaciones

comparten las mismas rutinas de interrupción y, en algunos casos, los mismos recursos que al seccionar su código resulta redundante y en un mayor consumo de memoria. Por ello, se ha desarrollado un solo módulo, en el que el código se ha dividido mediante directivas de preprocesador, con el fin de cargar solo el código necesario en un proyecto y cada una de las partes puede ser habilitada dentro del archivo *esp.h*. Esto permite hacer un uso más eficiente de los recursos del PIC.

Como se muestra en la figura 3.2, un proyecto que utilice la biblioteca ESP incluye al menos dos unidades de compilación y requiere de cuatro archivos: los dos principales de la biblioteca (*esp.c* y *esp.h*); el programa principal (*main.c*) que es realizado por el usuario; y la cabecera del controlador del PIC que se esté usando (*pic_driver.h*). En la primera unidad se encuentra el archivo *esp.c* que en su interior solicita los archivos *esp.h* y *pic_driver.h*, mientras que, en la segunda se encuentra *main.c*, que solicita los archivos *esp.h* y *pic_driver.h*. Al ser compilados se producen los objetos *esp.o* y *main.o* respectivamente, que finalmente son utilizados en el proceso de enlace para producir el archivo ejecutable del proyecto en un archivo con extensión (*.hex)*.

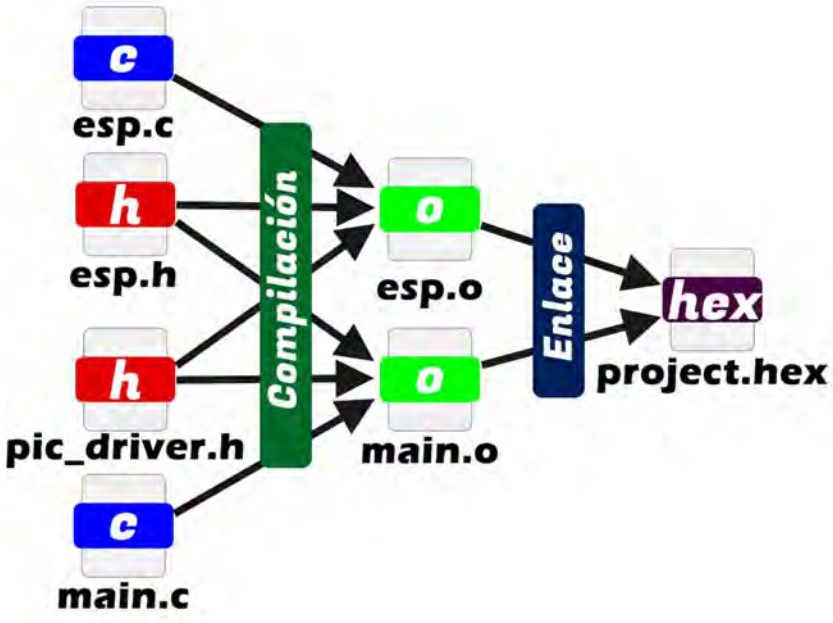


Figura 3.2 *Proceso de creación del ejecutable de un proyecto que utilice la biblioteca ESP.*

4.2.2 Funcionamiento interno

Como los módulos WiFi utilizan el *firmware* de comandos AT, la biblioteca ESP se encarga principalmente de gestionar el intercambio de comandos AT entre el PIC y el módulo, realizando cuatro principales tareas: configurar el módulo, proporcionar información que envía el módulo; procesar información que llegue al módulo; y solicitar información al

módulo. Para cumplir con las tareas anteriores, la biblioteca hace uso principalmente de dos periféricos, uno es el módulo UART que es utilizado para comunicación y el segundo es un temporizador (*timer*) empleado para administrar los tiempos de respuesta en dicha comunicación.

Para el proceso de intercambio de comandos AT, se hace uso de las interrupciones de los periféricos antes mencionados. La interrupción por desbordamiento del *timer* sirve para ejecutar el conteo del tiempo que el PIC espera por la respuesta del módulo, la cual varía conforme al comando AT, mientras que la interrupción por recepción de datos RDA es utilizada para procesar la respuesta del comando. Cada vez que exista un nuevo byte disponible para leer en el UART, se ejecuta la rutina de interrupción en la cual el byte es leído, almacenado en un *buffer* y se compara dicha respuesta con la respuesta exitosa que le corresponde a cada comando.

El código de comparación de respuestas al encontrarse en la rutina de interrupción RDA, debe ser lo más breve posible o al menos la ejecución dentro de ella no debe sobrepasar el tiempo de llegada entre cada byte, evitando así un desbordamiento por recepción en la unidad de comunicación serial. Se optó por tener una “lista positiva” de respuestas y, en consecuencia, sólo se puede saber si la ejecución del comando fue exitosa cuando se recibe una respuesta afirmativa del módulo. Se identifica una falla sólo cuando el tiempo de espera de la respuesta se ha vencido, esto a pesar de haber recibido una respuesta negativa por parte del módulo durante este tiempo. Lo anterior no permite conocer los detalles del fallo, razón por la cual en el circuito base se señala la utilización de un adaptador serial para este fin.

La secuencia general utilizada para ejecutar un comando AT se describe en las figuras 3.3 y 3.4. El primer paso es ajustar el límite del contador de la rutina de interrupción por desbordamiento del temporizador, seguidamente se activa ésta y la interrupción por recepción de datos RDA, posteriormente se asigna el valor de la respuesta que le corresponde al comando a ejecutar, se activa la bandera que mantiene la ejecución en un *loop* y se escribe en el registro TX el comando AT junto con sus parámetros. La bandera que permite el *loop* (bucle)*,* se desactiva por dos causas, la primera es que la rutina RDA ha recibido la respuesta esperada o que el tiempo de espera ha expirado. Si la causa es porque se ha encontrado la respuesta esperada, entonces a la variable error se le asigna un valor nulo; en caso contrario la rutina asigna el error por tiempo; para ambas situaciones se asigna valor nulo a la respuesta del comando y al contador del *timer*, para finalmente desactivar ambas interrupciones.

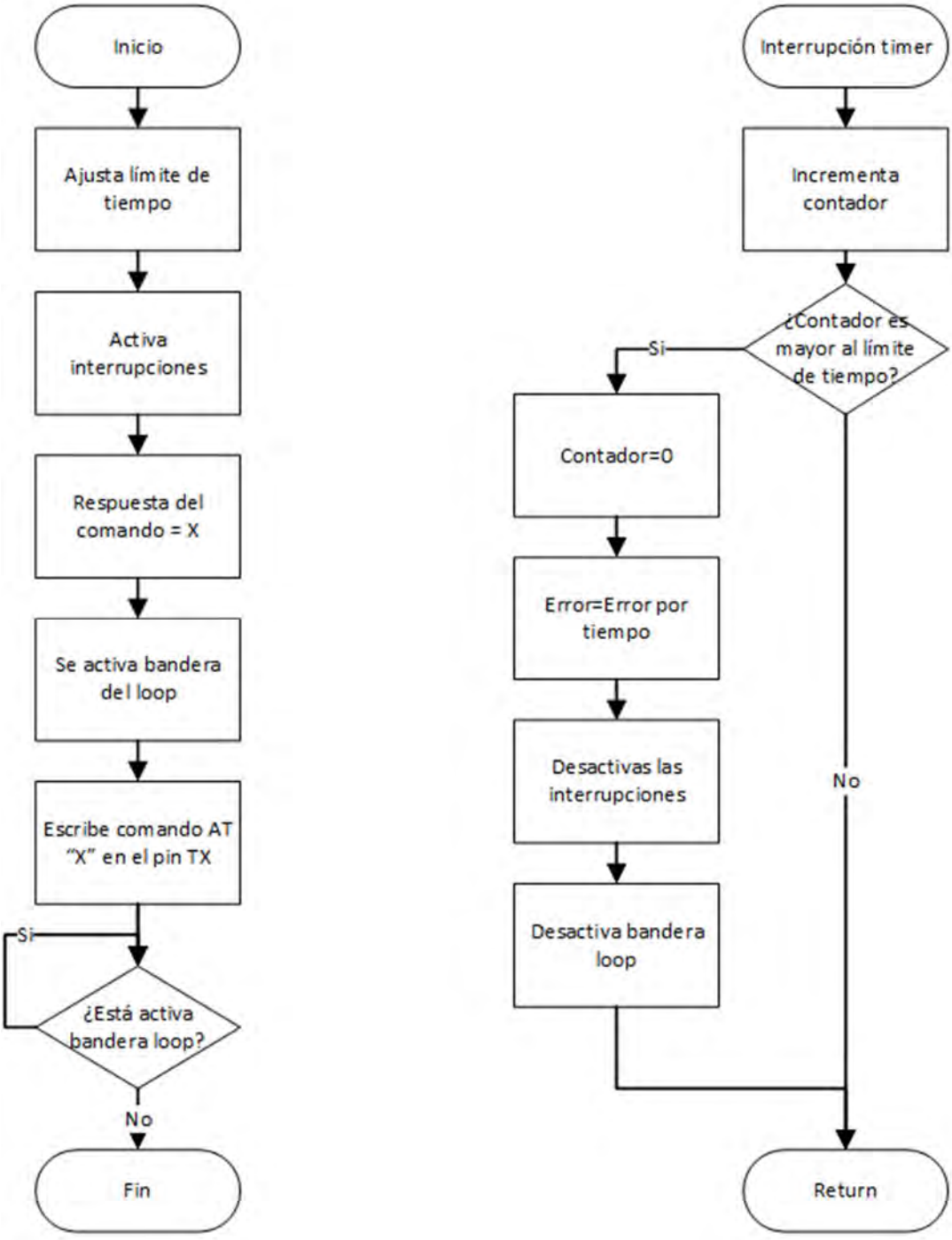


Figura 3.3 *Diagramas de flujo de ejecución de un comando AT y de la rutina de desbordamiento del timer*

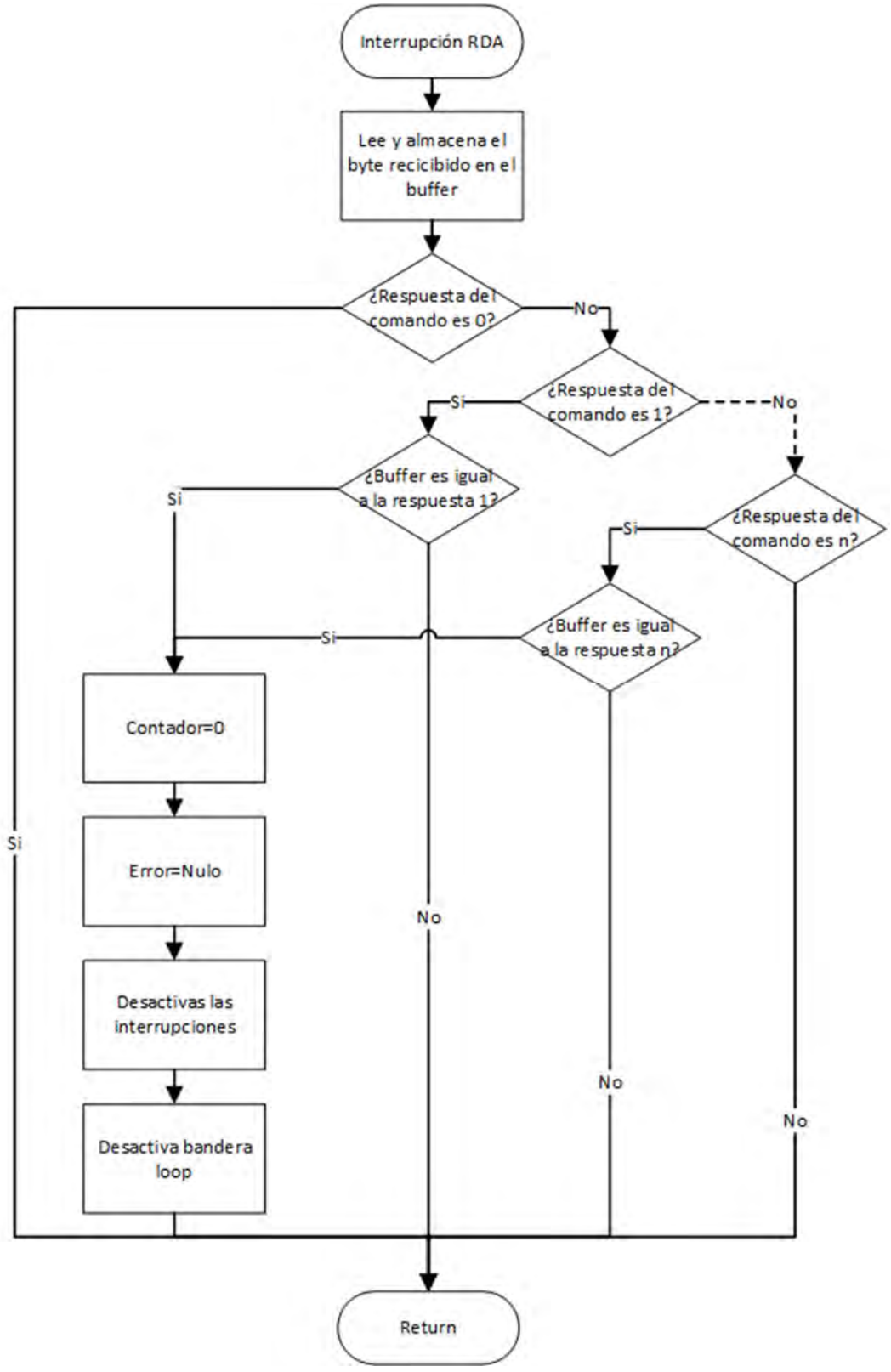


Figura 3.4 *Diagrama de flujo de la captura de la respuesta de un comando AT.*

La configuración del módulo sirve para ajustar una o varias de las aplicaciones disponibles en la biblioteca, estas configuraciones son realizadas por funciones que emiten uno o varios comandos AT (ver figura 3.5). Algunas de las configuraciones son dependientes de otras, las cuales deben ejecutarse previamente, por lo que para algunas aplicaciones es necesario ejecutar una secuencia de funciones de configuración. Por ello, se han diseñado niveles de ejecución que indican el estado actual de la configuración del módulo, existiendo niveles de ejecución generales que son utilizados para todas las aplicaciones y subniveles para ciertas aplicaciones. A continuación, se describen los cinco niveles generales:

- Nivel 0. Estado inicial del microcontrolador, se realiza la configuración inicial, se ejecutan los primeros comandos AT para reestablecer el módulo y se verifica que la comunicación entre el PIC y módulo sea correcta.
- Nivel 1. Indica que la comunicación entre el microcontrolador y el módulo es correcta; en este nivel sólo se puede establecer el módulo como un punto de acceso o conectarlo a una red WiFi.
- Nivel 2. Indica que el módulo se ha establecido como un punto de acceso o se ha conectado a una red; en este nivel se realizan las configuraciones necesarias para una o varias aplicaciones de las que dispone la biblioteca.
- Nivel 3. Indica que se llevó con éxito la configuración de una o varias aplicaciones, que intercambian información entre el microcontrolador-módulo y algún dispositivo conectado por WiFi. También en este nivel se pueden realizar ajustes adicionales o regresar al nivel 2.
- Nivel 4. Nivel especial reservado para un cliente transparente.

- Las aplicaciones que cuentan con sus propios subniveles son RSSI, *espnow*, *simple pair*, mdns, gpio, pwm y sntp.

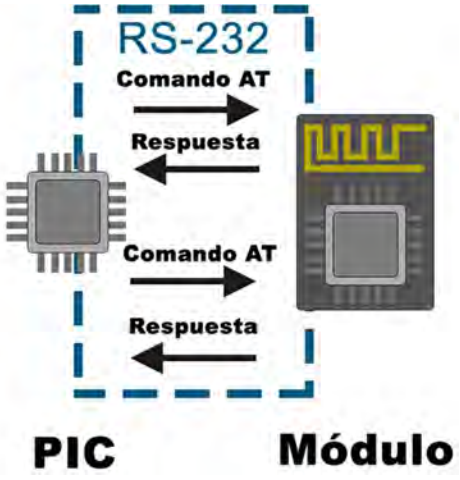


Figura 3.5 *Secuencia de ejecución de comandos AT en funciones de configuración.*

De manera general, el proceso de ejecución de una función de configuración puede ser descrita a través de la figura 3.6, en la que el primer paso es comprobar que la función pueda ser ejecutada en el nivel actual de ejecución, en caso de que no sea posible, la función regresa un valor de error. Si es posible ejecutarla, entonces, se envían los comandos necesarios para dicha configuración, si falla la ejecución de algún comando AT, finaliza la ejecución y regresa un valor de error. Si no hay error, regresa un valor nulo, indicando que la configuración fue ejecutada exitosamente. Un ejemplo de ejecución de comandos AT que realizan las funciones de configuración se muestra en la figura 3.7, en la que se ejecutan dos funciones de configuración, la primera establece al módulo como estación al conectarse a una red WiFi disponible, ejecutando el comando para establecer el nombre de la estación como “Estación” y otro comando para conectarse a la red “SSID”; la segunda función ejecuta el comando para habilitar múltiples canales de comunicación y establece en el canal cero comunicación con un servidor TCP.

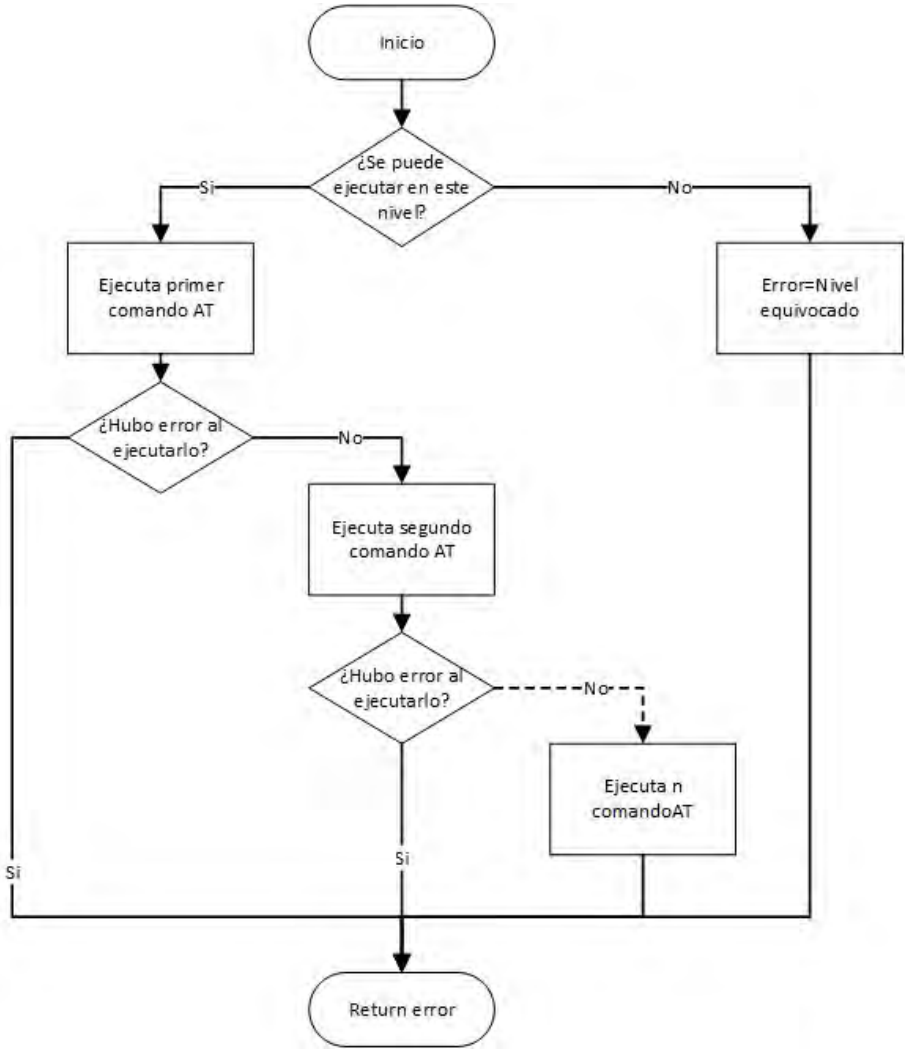


Figura 3.6 *Diagrama de flujo general de funciones de configuración.*

Existen otras funciones de configuración que no involucran ejecución de comandos AT; estas funciones sirven para ajustar parámetros de los recursos que se utilizan en las diferentes aplicaciones, como lo puede ser el ajustar los tamaños de los *buffers* utilizados para almacenar los mensajes recibidos.

Figura 3.7 *Ejemplo de una ejecución de comandos AT en funciones de configuración del módulo.*

De manera general, para que el conjunto PIC-módulo WiFi pueda emitir información a otro dispositivo, requiere que la ejecución del programa se encuentre en el Nivel 3; esto quiere decir que se ha completado con éxito alguna configuración y que el módulo ha establecido conexión con algún dispositivo. Tal como se muestra en la figura 3.8, el PIC emite el mensaje que desea transmitir a través de comandos AT y, posteriormente, el módulo se encarga de enviar dicho mensaje a su destino, que puede ser algún dispositivo conectado a la red WiFi o a la internet.

El envío de mensajes se realiza a través de las respectivas funciones de envío, que ejecutan los comandos correspondientes de cada una de las aplicaciones disponibles. La ejecución de estos comandos es casi igual a la usada en la configuración, la diferencia es que estos comandos se ejecutan comúnmente en dos partes. En la primera parte, se ejecuta el comando AT en el que se especifica la longitud del mensaje (en bytes) y en algunos casos el canal de comunicación; el módulo indicará que está listo para recibir dicho mensaje. En la segunda parte, el PIC escribirá el mensaje en el módulo de comunicación serial y finalmente el módulo responderá que ha enviado el mensaje. En la figura 3.9 se muestra un ejemplo de ejecución de comandos AT de envío de información, en el que se envía “Hola” y “Mensaje” a un servidor TCP; ambos mensajes se envían a través del canal cero, pero cuentan con distintas longitudes.

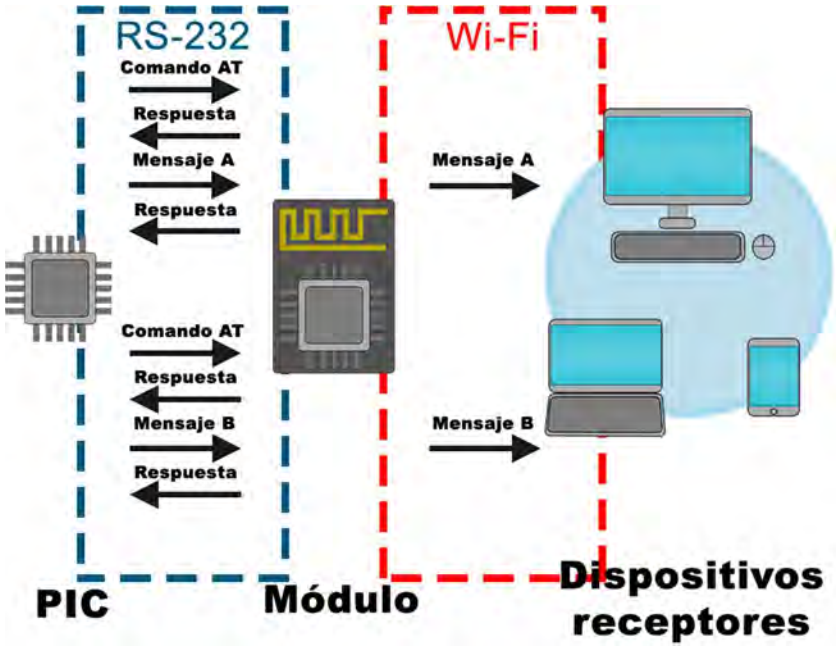


Figura 3.8 *Proceso de envío de mensajes a otros dispositivos.*

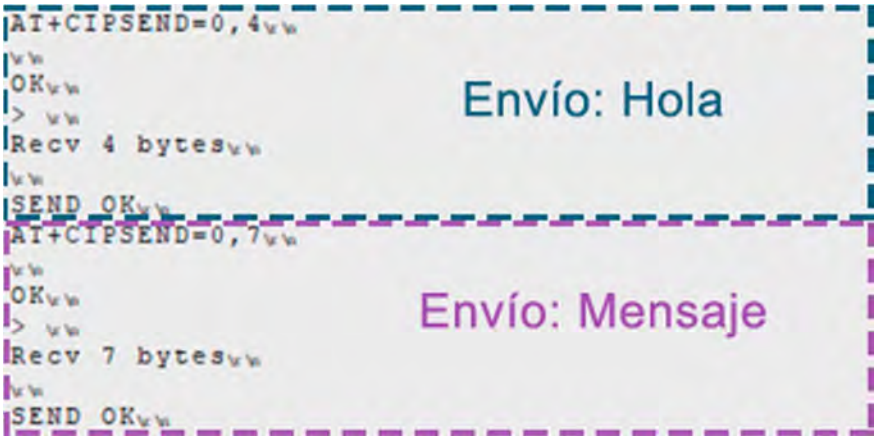


Figura 3.9 *Ejemplo de una ejecución de comandos AT en funciones de envío de mensaje.*

De acuerdo con la aplicación, la biblioteca ESP puede enviar directamente los mensajes sin tener que adaptarlos; estos casos se presentan en las aplicaciones que no usan algún protocolo de aplicación, es decir, que actúan directamente en la capa de transporte. Estas aplicaciones son:

- Cliente UDP
- Cliente TCP
- Cliente SSL
- Servidor TCP

- Servidor SSL
- ESPNOW
- Simple pair

Las aplicaciones en las que la biblioteca ESP otorga un formato a la información antes de transmitirla por medio del serial son aquéllas que utilizan algunos de los protocolos de aplicación que han sido adaptados directamente en la biblioteca ESP, que son:

- Cliente HTTP
- Servidor WEB
- Servidor *Websocket*
- Cliente MQTT

Del mismo modo, en el nivel tres el módulo puede recibir mensajes de aquellos dispositivos con los cuales ha establecido una conexión (ver figura 3.10). El PIC está listo para recibir esta información en cualquier momento puesto que en vez de usar la técnica como *polling* (sondeo), se emplea la interrupción por recepción de datos RDA, por lo que esta interrupción debe estar habilitada siempre que se esté en este nivel; esto implica que las funciones que ejecuta alguna configuración o envío de información siempre vuelven a habilitarla.

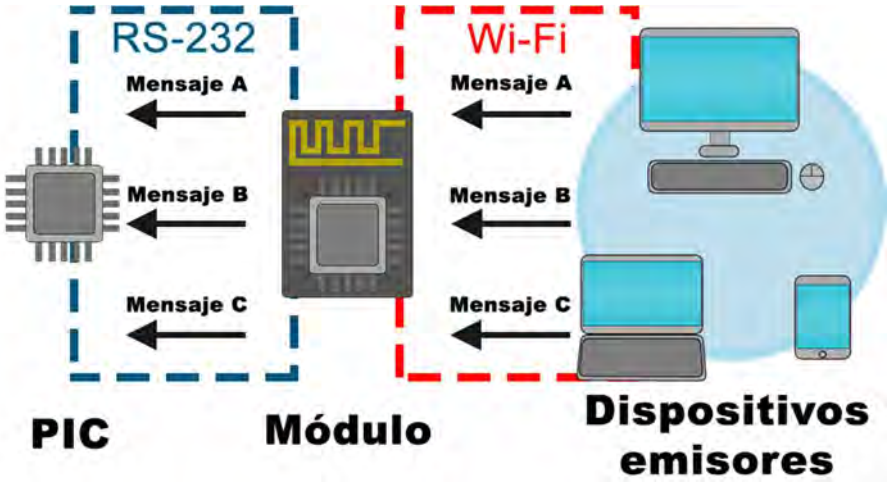


Figura 3.10 *Proceso en el PIC cuando recibe información de otros dispositivos.*

Cuando el módulo retransmite el mensaje recibido hacia el PIC, incluye un identificador, por ejemplo, las aplicaciones que originalmente venían en *firmware* del módulo incluyen el identificador “+IDP” (ver figura 3.11), en tanto para las que surgieron con los nuevos comandos AT, presentan identificadores distintos. Los mensajes entrantes, además de

contener un identificador, pueden incluir la longitud del mensaje y en algunos de ellos el canal de comunicación. Como ya se ha mencionado, en la rutina RDA existe un *buffer* que es utilizado para almacenar temporalmente lo que proviene del módulo.

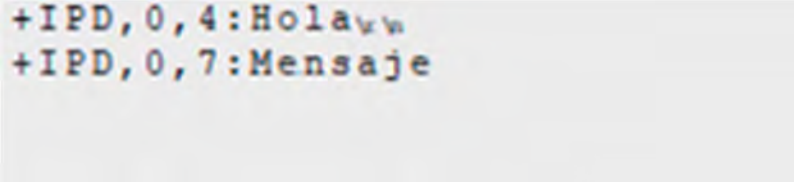


Figura 3.11 *Ejemplo de mensajes entrantes, provenientes de otros dispositivos.*

Al igual que las respuestas del comando, el mensaje es almacenado en el *buffer* y comparado, esperando por algunos de estos identificadores. Cuando se reconoce a alguno de éstos, se registra la longitud del mensaje y su canal si es el caso, esto es importante para que la rutina determine hasta que byte de la respuesta debe ser interpretado como un mensaje y no como parte de la respuesta de algún comando AT. De acuerdo con la aplicación, la información recibida puede ser filtrada y almacenada en un *buffer* o pueda ser procesada como mensaje de algún protocolo de aplicación.

Las aplicaciones en las que los mensajes son directamente almacenados en el *buffer*, son aquéllas que trabajan directamente en la capa de transporte o aquéllas que su protocolo de aplicación está programado en el *firmware* del módulo, éstas son:

- Cliente UDP
- Cliente TCP
- Cliente SSL
- Servidor TCP
- Servidor SSL
- ESPNOW
- Simple pair
- RSSI

Las aplicaciones en las que el mensaje debe ser procesado de acuerdo con un protocolo, son aquéllas que su protocolo de aplicación ha sido adaptado en la biblioteca ESP. Para ello, la rutina RDA, además de hallar el identificador del mensaje, debe identificar el formato del mensaje acorde al protocolo, esto lo realiza comparando lo que llega al *buffer* con algún identificador del protocolo. Dichas aplicaciones son las siguientes:

- Servidor WEB
- Servidor *Websocket*
- Cliente MQTT

Existen algunos protocolos de aplicación como HTTP, MQTT y *Websocket* que cuentan con comandos AT o APIs únicamente en la versión IDF del *firmware*, y que no pueden ser replicados completamente en la versión NONOS. Por tanto, buscando homogeneidad en las aplicaciones para el microcontrolador se han programado éstos junto con otros protocolos como SMTP en el lado del microcontrolador PIC. Es importante señalar que en la biblioteca ESP no se han empleado las especificaciones completas de estos protocolos, ya que algunos de ellos no fueron diseñados para dispositivos con limitada capacidad de cómputo, tal como es el caso de un microcontrolador PIC.

Estos protocolos están involucrados en las tareas de configuración del módulo, proporcionar información que el módulo enviará y procesar información que llegue desde éste, usando el formato que especifique estos protocolos.

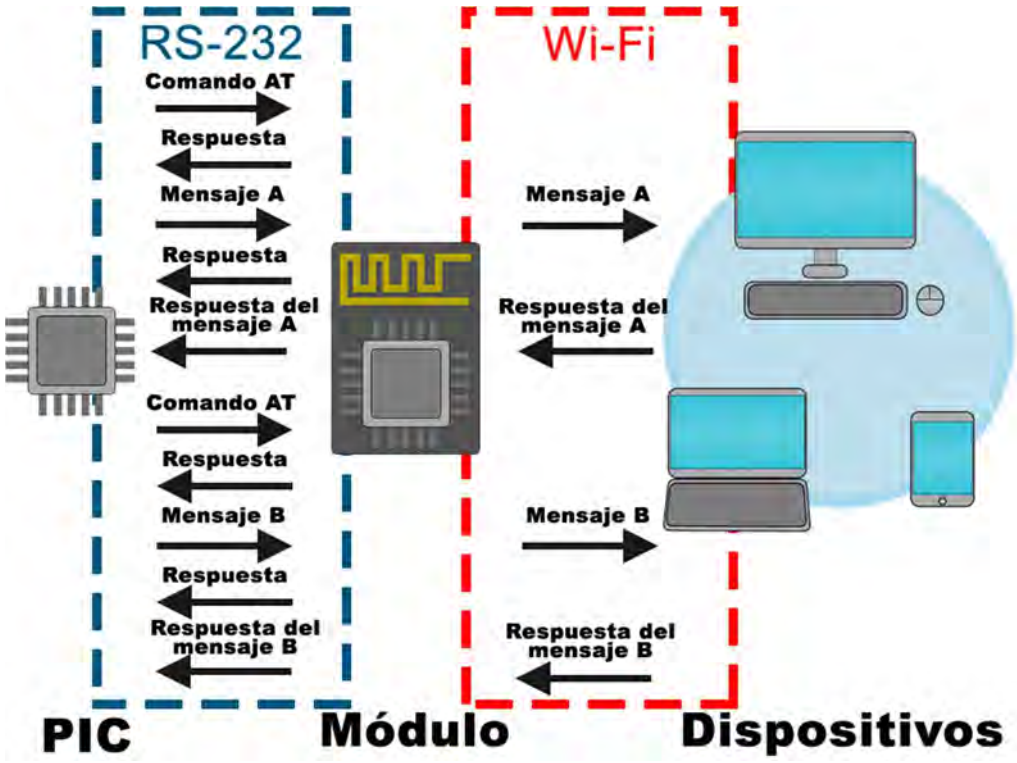
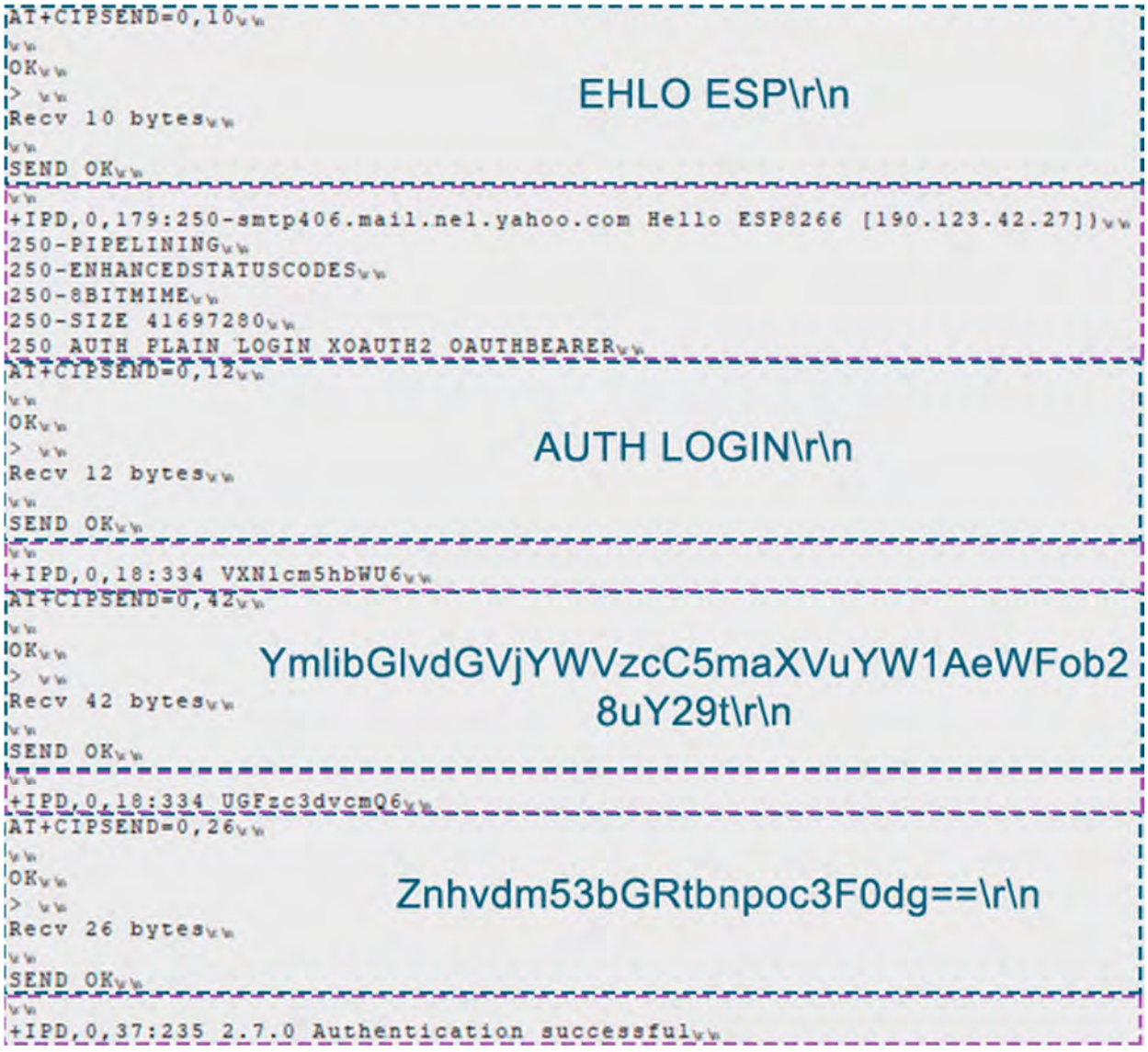


Figura 3.12 *Proceso de envío y recepción secuencial para las aplicaciones como SMTP, cliente HTTP, OAUTH2.0 y MQTT.*

Para algunas de las aplicaciones que utilizan los protocolos de aplicación que han sido adaptados, se han creado funciones que envían y reciben información de algún dispositivo, de manera ordenada y secuencial. Este tipo de funciones pueden ser ejecutadas en el segundo o tercer nivel de ejecución. El procedimiento de intercambio de información se muestra en la figura 3.12, el PIC envía a un dispositivo un mensaje que puede ser

interpretado para el destinatario como una petición o como un comando, en consecuencia, el dispositivo responderá y el PIC debe procesar dicha respuesta que puede contener algún mensaje para el usuario de la biblioteca; este tipo de funciones están disponibles para las siguientes aplicaciones:

- Cliente SMTP, el módulo emite mensajes que son comandos SMTP para enviar correos electrónicos.
- Cliente HTTP, el módulo emite mensajes que son peticiones HTTP.
- OAuth 2.0 para APIS de Google, ejecuta todas las peticiones HTTP para obtener el access *token* (token de acceso) que permite acceder a las APIs de Google.
- MQTT en el envío de algunos mensajes.



- Figura 3.13 *Recepción de mensajes de manera ordenada y secuencial.*

En la figura 3.13 se muestra un ejemplo de envío y recepción de mensajes de manera ordenada y secuencial para una función que emplea un protocolo de aplicación adaptado en la biblioteca ESP. En este ejemplo se muestra un fragmento de la ejecución de comando SMTP a un servidor para enviar un correo electrónico, en el que se observa que por cada comando emitido como mensaje se obtiene una respuesta como mensaje entrante.

La última de las tareas de la biblioteca es realizar peticiones para obtener información del módulo (ver figura 3.14). Las funciones que realizan este tipo de tarea siguen un procedimiento similar a las funciones de configuración, puesto que sólo se encargan de ejecutar comandos AT que en sus respuestas se encuentra la información solicitada. La ejecución de comandos es la misma, sólo que en la rutina de la interrupción RDA, además de saber si se obtuvo la respuesta adecuada, se tiene que almacenar la información recibida. Las funciones que ejecutan este tipo de tarea están presentes en las siguientes aplicaciones:

- Punto de acceso/estación
- DNS
- SNTP
- RSSI
- *Tracking*
- Métodos criptográficos

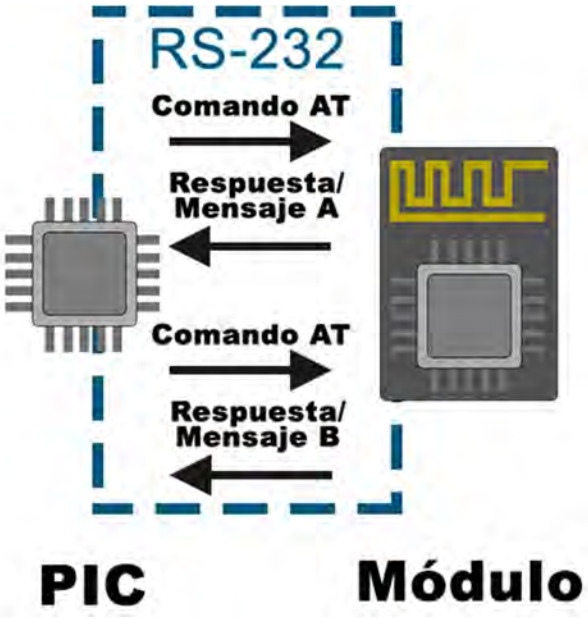


Figura 3.14 *Proceso de solicitud de información al módulo.*

En la figura 3.15 se muestra un ejemplo de ejecución de comandos de funciones que solicitan información al módulo; para este caso se ejecuta dos veces un comando que solicita el valor RSSI al módulo, como se puede observar en la respuesta se encuentra la información solicitada.

Figura 3.15 *Ejemplo de ejecución de petición de información al módulo.*

4.2.3 Características

Para buscar que la biblioteca ESP tuviera flexibilidad para futuras mejoras, se usaron algunas de las herramientas que proporciona el lenguaje C, como enumeraciones y directivas de preprocesador para facilitar la escritura y lectura del código, permitiendo identificar y corregir errores en la programación. A continuación, se describen las características más relevantes del código, así como las características acerca del uso de tipo de datos que son ocupados en la biblioteca ESP.

El sistema de datos en cada compilador es diferente, siendo posible que un mismo alias describa distintos tipos de datos en diferentes compiladores. Ante esta situación, se decidió establecer alias alternativos, los cuales son más homogéneos y facilitan notablemente la escritura del código, además de que evita reemplazar todos los alias predeterminados de CCS, ante un eventual cambio en nuevas versiones del compilador. De este modo, solo es necesario modificar la declaración de los alias alternativos de la biblioteca para adaptarse a nuevas versiones. En la tabla *3-1* se muestran los alias utilizados en la biblioteca.

Así mismo, el compilador CCS usa *little endian* como formato de almacenamiento de datos. Este formato indica que los datos que tengan un tamaño mayor a un byte, almacenan el byte menos significativo en la dirección más baja de memoria, mientras que el byte más significativo se encuentra en la dirección más alta. Es relevante mencionar esta característica porque se empleó de manera significativa en arreglos y uniones en la biblioteca ESP, por lo que un cambio en este formato podría afectar el funcionamiento de algunas funciones de la biblioteca.

Tabla 3-1 Alias de los tipos de datos utilizados por la biblioteca ESP.

Alias para la biblioteca	Alias por defecto	Rango de valores
_bool	short	0 a 1
uint8	unsigned int	0 a 255
sint8	signed int	-128 a 127
uint16	unsigned long	0 a 65535
sint16	signed long	-32768 a 32767
real32	float	-1.5×10^{45} a 3.4×10^{38}
uint32	unsigned long long	0 a 4294967295
sint32	signed long long	-2147483648 a 2147483647

Como ya se ha mencionado, la biblioteca ESP se encarga de gestionar la ejecución de comandos AT, por lo que están presentes en el código de la mayoría de las funciones de la biblioteca. Para evitar que en la escritura del código el comando sea escrito completamente en todas las líneas de código involucradas, se decidió definir cada uno como constantes a través de la directiva de preprocesador #define que, durante su ejecución, los comandos AT son incrustados en el código de la biblioteca. A cada una de estas constantes se les ha asignado un alias breve que permita ajustar con facilidad el comando ante un posible cambio. En las tablas 3-2, 3-3 y 3-4 se señalan los comandos utilizados en la biblioteca ESP, además las versiones de *firmware* y aplicaciones están destinadas.

Tabla 3-2 Definiciones de los comandos AT utilizados por la biblioteca ESP parte 1

Definición	Comando	Versión	Aplicaciones
C0	AT+RESTORE	SDK2.2/3.3/IDF	GENERAL
C1	ATE0	SDK2.2/3.3/IDF	
C2	AT+SYSIOSETCFG	SDK3.3	GPIO
C3	AT+SYSGPIODIR	SDK3.3	
C2	AT+CFGGPIO	IDF	
C4	AT+SYSGPIOWRITE	SDK3.3/IDF	
C5	AT+SYSGPIOREAD	SDK3.3/IDF	
C6	AT+CWMODE_CUR	SDK2.2/3.3	STATION, ACCESS_POINT
C6	AT+CWMODE	IDF	
C7	AT+CWSAP_CUR	SDK2.2/3.3	ACCESS_POINT
C8	AT+CIPAPMAC_CUR	SDK2.2/3.3	
C9	AT+CIPAP_CUR	SDK2.2/3.3	
C7	AT+CWSAP	IDF	
C8	AT+CIPAPMAC	IDF	
C9	AT+CIPAP	IDF	

Tabla 3-3 Definiciones de los comandos AT utilizados por la biblioteca ESP parte 2

Definición	Comando	Versión	Aplicaciones
C10	AT+CWHOSTNAME	SDK2.2/3.3/IDF	STATION
C11	AT+CWLAPOPT	SDK2.2/3.3/IDF	
C12	AT+CWLAP	SDK2.2/3.3/IDF	
C13	AT+CWJAP_CUR	SDK2.2/3.3	
C14	AT+CIPSTAMAC_CUR	SDK2.2/3.3	
C15	AT+CIPSTA_CUR	SDK2.2/3.3	
C13	AT+CWJAP	IDF	
C14	AT+CIPSTAMAC	IDF	
C15	AT+CIPSTA	IDF	
C16	AT+CWQAP	SDK2.2/3.3/IDF	
C17	AT+CIPDNS_CUR	SDK2.2/3.3	DNS
C17	AT+CIPDNS	IDF	
C18	AT+CIPDOMAIN	SDK2.2/3.3/IDF	
C19	AT+CIPSEND	SDK2.2/3.3/IDF	SSL_TCP_UDP_CLIENT_MULTICON,SSL_TCP_UDP_CLIENT_UNICON,HTTP_CLIENT , INT_MQTTCLIENT , SMTP , TCP_SERVER ,WEB_SERVER , TRANSPARENT_CLIENT , GOOGLOAUTH20 , SSL_SERVER
C20	AT+CIPCLOSE	SDK2.2/3.3/IDF	
C21	AT+CIPMUX	SDK2.2/3.3/IDF	
C22	AT+CIPSTART	SDK2.2/3.3/IDF	SSL_TCP_UDP_CLIENT_MULTICON,SSL_TCP_UDP_CLIENT_UNICON,HTTP_CLIENT , INT_MQTTCLIENT , SMTP , TRANSPARENT_CLIENT , GOOGLOAUTH20
C23	AT+SYSMSG	SDK2.2/3.3	TRANSPARENT_CLIENT
C24	AT+CIPMODE	SDK2.2/3.3	
C25	AT+CIPSERVERMAXCONN	SDK2.2/3.3/IDF	INT_SERVER
C26	AT+CIPSERVER	SDK2.2/3.3/IDF	
C27	AT+CIPSTO	SDK2.2/3.3/IDF	
C28	AT+CIPSNTPCFG	SDK2.2/3.3/IDF	SNTP_CLIENT
C29	AT+CIPSNTPTIME	SDK2.2/3.3/IDF	
C30	AT+CWSTARTSMART	SDK2.2/3.3/IDF	SMART_CONFI
C31	AT+CWSTOPSMART	SDK2.2/3.3/IDF	
C32	AT+STALED	SDK2.2/3.3	STATION
C33	AT+APS	SDK2.2/3.3/IDF	
C34	AT+RSSI	SDK/IDF	RSSI
C35	AT+RSSIP	SDK/IDF	
C36	AT+RSSIM	SDK/IDF	
C37	AT+RSSIC	SDK/IDF	
C38	AT+RSSISS	SDK/IDF	
C39	AT+RSSISC	SDK/IDF	
C40	AT+UPG	SDK/IDF	SOFT_UPGRADE_PIC
C41	AT+SV	SDK/IDF	SECOND_TCP_SERVER, WEBSOCKET_SERVER
C42	AT+SVSEND	SDK/IDF	
C43	AT+SVC	SDK/IDF	
C44	AT+NMDNS	SDK/IDF	MDNS
C45	AT+NMDNSST	SDK/IDF	
C46	AT+NMDNSS	SDK/IDF	
C47	AT+NMDNSQ	SDK/IDF	

Tabla 3-4 Definiciones de los comandos AT utilizados por la biblioteca ESP parte 3.

Definición	Comando	Versión	Aplicaciones
C48	AT+SP	SDK	SIMPLE PAIR
C49	AT+SPS	SDK	
C50	AT+SPGET	SDK	
C51	AT+SPGETS	SDK	
C52	AT+SPSET	SDK	
C53	AT+SPANU	SDK	
C54	AT+SPFILTER	SDK	
C55	AT+EN	SDK/IDF	ESPNOW
C56	AT+ENFILTER	SDK/IDF	
C57	AT+ENADD	SDK/IDF	
C58	AT+ENSEND	SDK/IDF	
C59	AT+ENSENDA	SDK/IDF	
C60	AT+ENSMAC	SDK/IDF	
C61	AT+CIPSSLSIZE	SDK	SMTP , SSL_TCP_UDP_CLIENT_UNICON , SSL_TCP_UDP_CLIENT_MULTICON , HTTP CLIENT , GOOGLE OAUTH20 , INT_MQTTCLIENT
C62	AT+SSLSV	SDK	SSL_SERVER
C63	AT+SSLSEND	SDK	
C64	AT+SSLC	SDK	
C65	AT+SNIF	SDK/IDF	TRACKING
C66	AT+PACKET	SDK/IDF	ESTACIÓN
C67	AT+SNIFREG	SDK/IDF	TRACKING
C68	AT+ENC	SDK/IDF	CRYPTOGRAPHIC_FUNC , WEBSOCKET_SERVER
C69	AT+IV	SDK/IDF	CRYPTOGRAPHIC FUNC
C70	AT+VKEY	SDK/IDF	
C71	AT+SKEY	SDK/IDF	
C72	AT+FINGER	SDK	SSL_TCP_UDP_CLIENT_UNICON , SSL_TCP_UDP_CLIENT_MULTICON
C73	AT+SYSADC	SDK/IDF	ADC
C74	AT+UART_CUR	SDK	STATION , GOOGLOAUTH20
C75	AT+DSTA	SDK	ACCESS_POINT
C76	AT+SCPWM AT+PWMTIMER	SDK3.3/IDF	PWM
C77	AT+SPWM AT+PWMCHAN	SDK3.3/IDF	
C78	AT+PER AT+PWMFRE	SDK3.3/IDF	
C79	AT+DUTY AT+PWMDUTY	SDK3.3/IDF	
C80	AT+SPEC	SDK3.3	CLIENTE ESPECIAL
C81	AT+SPECS	SDK3.3	
C82	AT+SPECC	SDK3.3	
C83	AT+SNIFPR	SDK/IDF	TRACKING
C84	AT+SNIFCT	SDK/IDF	
C85	AT+MODE	SDK/IDF	GENERAL
C86	AT+POWER	SDK/IDF	
C87	AT+ADC	IDF ESP32	ADC
C88	AT+SYSADCO	IDF ESP32	
C89	AT+PHA	IDF ESP8266	PWM

Se ha establecido una enumeración para indicar qué respuesta en la rutina de interrupción RDA se identifica cómo respuestas válidas de comandos y mensajes que llegan cuando se usa algún protocolo de aplicación adaptado. Esta enumeración se muestra en la tabla 3-5.

Gran parte de las funciones de la biblioteca ESP retorna un entero de 8 bits, cuyo valor indica el éxito o fracaso de la ejecución de la función. En la tabla 3-6 se muestra la enumeración establecida para estos posibles valores. Así mismo, en la biblioteca se hace uso de *#device PASS_STRINGS=IN_RAM* para permitir el uso de *strings* (cadena de carácteres) como parámetros de funciones, lo que facilita y hace más interactivo el uso de las funciones de la biblioteca ESP, permitiendo incluso que el usuario pueda utilizar esta misma característica en sus propias funciones.

Tabla 3-5 *Enumeración de las respuestas que la biblioteca ESP puede identificar.*

Valor	Respuesta
0	RESP_NULL
1	RESP_OK
2	RESP_GPIO_READ
3	RESP_GET_MAC_AP
4	RESP_GET_IP_AP
5	RESP_APS
6	RESP_GET_MAC_STA
7	RESP_GET_IP_STA
8	RESP_REQUEST_DNS
9	RESP_GET_SNTP
10	RESP_SMART
11	RESP_SEND
12	RESP_TRANS_CLIENT
13	RESP_CONNACK
14	RESP_SUBACK
15	RESP_UNSUBACK
16	RESP_PINGRESP
17	RESP_PUBACK
18	RESP_PUBREC
19	RESP_PUBCOMP
20	RESP_PUBREL
21	RESP_OK_SMTP
22	RESP_220_SMTP
23	RESP_250_SMTP
24	RESP_334_SMTP
25	RESP_235_SMTP
26	RESP_354_SMTP
27	RESP_221_SMTP
28	RESP_RSSI
29	RESP_RSSI_OK
30	RESP_QMDNS
31	RESP_SIMPLE_PAIR_MAC
32	RESP_SIMPLE_PAIR_MESSAGE
33	RESP_SET_ENCRYPT
34	RESP_ENCRYPT
35	RESP_ADC_READ

Tabla 3-6 Enumeración de valores de retorno de las funciones de la biblioteca ESP.

Valor	Retorno	Descripción
0	RET_OK	No existió ningún error en la ejecución de la función.
0	RET_IGNORED	La función ha sido ignorada, ya que no se han cumplido las condiciones para que ésta sea ejecutada.
0	RET_LOW	Indica que el nivel de voltaje presente en el pin es bajo.
1	RET_DETECTED	Indica que se ha detectado el dispositivo con la dirección MAC especificada en la aplicación tracking.
1	RET_ASSIGNED	Indica que se haobtenido un valor RSSI y éste ha sido correctamente asignado a la variable dada.
1	RET_HIGH	Indica que el nivel de voltaje presente en el pin es alto.
15	RET_ERROR	Indica que ha ocurrido un error durante la ejecución de la función, que puede deberse a que el módulo no responda, los parámetros eran incorrectos, entre otros.
16	RET_RESTART_ERROR	Reservada para un uso futuro, que indique reinicio del módulo durante la ejecución de una función
17	RET_LEVEL_WRONG	Indica que se ha llamado una función que requiere una configuración previa para ser llamada.
18	RET_APS_NOTFOUND	Indica que ninguno de los SSID dados fueron encontrados.
19	RET_WRONG_SIZBUF	Indica que el tamaño del *buffer* dado no coincide con el mínimo requerido
20	RET_FAIL_TRANS_DISC	Indica que hubo un fallo al intentar deshabilitar el cliente transparente, por lo que se debe forzar su cierre con la función correspondiente.
21	RET_NOT_CONNECTED	Indica que no se ha logrado establecer conexión con una sesión MQTT.
22	RET_TOPIC_NOT_AVAILABLE	Indica que para el tema especificado no existe mensaje disponible en el *buffer*.
23	RET_MESSAGE_NOT_AVAILABLE	Indica que no existe mensaje disponible en el *buffer*.
24	RET_WRONG_SSID	Indica que el SSID dado no es válido.
25	RET_FATAL_ERROR	Indica que el módulo se ha reiniciado inesperadamente.
26	RET_PORT_ERROR	Indica que el puesto especificado es no válido.
27	RET_BUSY_CHANNEL	Indica que el ID del canal que el cliente TCP o UDP, en multiconexión quiere reclamar, ya ha sido tomado por otro cliente.
28	RET_MAC_ERROR	Indica que la dirección MAC especificada no es válida.
29	RET_INVALID_GPIO	Indica que el GPIO especificado no es válido, ya que internamente es ocupado o está reservado para una función en específico.
30	RET_WRONG_CHANNEL	Indica que el ID del canal que se especificó está fuera del rango de canales válidos.
31	RET_WRONG_KEY_LENGTH	Indica que la longitud de la clave específica no coincide con las requeridas.
32	RET_NO_EXECUTESEND	Indica que no se puede enviar el mensaje en espnow.
33	RET_UNMASKED_CLIENT	Indica que el cliente del *websocket* ha enviado información sin enmascararla, por lo que se procedió a su desconexión.
34	RET_NOT_REQUEST_CONECTION_WS	Indica que no existe conexión pendiente por establecer en *websocket*.
35	RET_BUSY_RSSI	Indica que ya se está ejecutando una tarea RSSI, por lo cual no es posible ejecutar a la que se ha llamado.

Capítulo 5

Resultados y manual de prácticas

5.1 Resultados

La biblioteca ESP diseñada para el compilador CCS puede ser incluida en proyectos de MPLAB IDE v8 y MPLAB X IDE para cualquier microcontrolador PIC que:

- Cuente con al menos un temporizador de 8 o 16 *bits* con interrupción por desbordamiento del temporizador.
- Disponga de un módulo USART con interrupción por recepción de datos.
- Cuente con un tamaño, al menos, de 6 registros de pila.
- Trabaje a una frecuencia que permita alcanzar una velocidad de transmisión de 115.200 baudios para la comunicación serial entre el PIC y el módulo ESP.

Tanto el temporizador como el módulo EUSART junto con sus respectivas interrupciones están destinados completamente al funcionamiento de la biblioteca ESP, por lo que no podrán ser utilizados por el usuario para otros fines. En consecuencia, se debe considerar la utilización de microcontroladores PIC que cuenten con más de un temporizador y/o módulo EUSART para aquellos proyectos que requieran de este tipo de periféricos para desempeñar tareas ajenas a la biblioteca.

Ya que la biblioteca ESP hace uso de la interrupción por recepción de datos y desbordamiento de un temporizador, ésta controla la habilitación general (global) de las interrupciones en el microcontrolador PIC, por lo que el usuario no puede hacer uso de

dicha característica ya que interferiría con el funcionamiento de la biblioteca. Es posible utilizar otras interrupciones, sin embargo, se debe considerar que la biblioteca ha utilizado la directiva del microcontrolador `#priority` para asignarle a la interrupción por recepción de datos la mayor prioridad, por lo que, para utilizar varias interrupciones, el usuario debe modificar el archivo fuente de la biblioteca y establecer la jerarquía de prioridades (si la prioridad es requerida), que para un buen funcionamiento de la biblioteca debe mantener a la interrupción por recepción de datos con la mayor prioridad.

El usuario tiene que considerar que la biblioteca ESP hace uso de otras directivas del microcontrolador como `#device PASS_STRINGS=IN_RAM` y `#zero_ram`. La primera sirve para utilizar *strings* constantes como parámetros de funciones, por lo cual se copia el *string* a la RAM y se proporciona el puntero de la copia a la función (estando ahora disponible para las funciones del usuario), mientras que, la segunda directiva coloca a cero a todos los registros donde se pueden almacenar variables de la memoria de datos del PIC.

En cada proyecto que requiera el uso de la biblioteca ESP se deben incluir los dos archivos de la biblioteca (esp.c y esp.h) los cuales deben ser modificados para ajustar los parámetros de los periféricos utilizados en un proyecto, adecuándose a la diversidad de microcontroladores PIC y de sus componentes. En el archivo fuente se debe incluir y/o modificar lo siguiente:

- Incluir el *driver* del microcontrolador PIC utilizado en el proyecto.
- En el caso de utilizar otras interrupciones, es necesario la modificación de la línea de prioridades de éstas.
- En caso de requerir pines distintos a los que por defecto asigna el UART1 en la comunicación entre PIC y módulo WiFi, se debe modificar la línea que define esta comunicación.

En el archivo de cabecera se debe modificar la directiva `#use delay` para indicar la frecuencia de trabajo del PIC para el oscilador interno o externo, el valor debe generar la tasa de 115.200 baudios requerida en la comunicación serial del *hardware* del uC (para una tasa diferente, se tendría que modificar el *firmware* del módulo). Además, en este archivo se encuentra una lista de definiciones cuyos valores deben ser modificados para ajustar lo siguiente:

- La frecuencia a la que trabaja el uC PIC.
- La velocidad de transferencia y pines para la comunicación del segundo módulo de comunicación seríal virtual creado en la biblioteca para uso del usuario.

- El temporizador del PIC que se destina al funcionamiento de la biblioteca junto con sus valores de predivisor y postdivisor para la interrupción por desbordamiento del temporizador (la biblioteca calcula automáticamente los tiempos de respuesta).

Gracias a las características que disponen los *firmware* de comandos AT junto con modificaciones realizadas en éstos y las adaptaciones de algunos protocolos de aplicación, han permitido a la biblioteca ESP ofrecer hasta 28 aplicaciones, con las que se puede desarrollar una gran cantidad de *software* en el diseño de prototipos que requieran comunicación inalámbrica y/o acceder a servicios o plataformas alojadas en internet.

De estas 28 aplicaciones, existen dos que son conocidas como “aplicaciones base”, las cuales establecen al módulo en alguno de sus dos modos de operación, ya sea como una estación que se conecta a un punto de acceso WiFi existente o como punto de acceso para establecer una nueva red. Siempre que se utilice la biblioteca ESP, es necesario habilitar una de estas dos aplicaciones base, sin embargo, la gran mayoría de las aplicaciones sólo se pueden ejecutar para un modo específico de operación. La habilitación de todas las aplicaciones que requieren conexión a internet ha sido restringida únicamente para cuando el módulo funciona como una estación WiFi.

No todas las aplicaciones de la biblioteca están disponibles para todas las versiones de los *firmware* modificados. Principalmente las aplicaciones que tienen control sobre los periféricos del módulo han sido descartadas para la versión SDK NONOS v2.2.1 (aunque no son las únicas), ya que esta versión fue modificada para los módulos que cuentan con una memoria flash de 8 *Mbits* caracterizados por presentar respecto a *hardware*, una restricción al acceso de muchas de sus entradas/salidas que el SoC dispone. Así mismo, esta misma versión presenta limitaciones como cliente SSL respecto a los *cipher suites* (conjuntos de cifrado), lo que no le permite conectarse a ciertos servidores por este protocolo.

Otras versiones del *firmware* también presentan limitaciones respecto a las aplicaciones desarrolladas; en el caso de la versión del *firmware* basada en ESP-IDF, éstas se deben al hecho de que ciertas APIs utilizadas en SDK NONOS no están disponibles en esta versión. Otras aplicaciones como servidor SSL, son distintas de acuerdo con la versión del *firmware*, puesto que en ESP-IDF ya se incluye esta característica, mientras que para SDK NONOS se realizarón modificaciones para incluirla.

Además de los 28 tipos de aplicación se incluyeron tres herramientas:

- JSON para un manejo limitado de este tipo de información en las aplicaciones de cliente HTTP, servidor web y websocket;

- *STATUS LED* para la aplicación base de estación WiFi en la versión SDK NONOS
- INTERNAL_SHA1 para el cálculo interno de SHA-1 para las conexiones del websocket, usando recursos del microcontrolador PIC en lugar del módulo.

En la tabla 4-1, se muestran las 28 aplicaciones disponibles de la biblioteca ESP junto con las herramientas extra con las que cuentan, en la cual se especifica para cada una éstas en qué versiones de *firmware* pueden ser utilizadas.

Tabla 4-1 *Aplicaciones disponibles para la biblioteca ESP.*

Aplicación	***Firmware***
Aplicación base	
STATION	SDK/IDF
ACCESS_POINT	SDK/IDF
Aplicaciones particulares	
SMART_CONFI	SDK/IDF
GPIO	SDK3.3/IDF
PWM	SDK3.3/IDF
RSSI	SDK/IDF
SNTP_CLIENT	SDK/IDF
DNS	SDK/IDF
MDNS	SDK/IDF
ESPNOW	SDK
SIMPLE_PAIR	SDK/IDF
SSL_TCP_UDP_CLIENT_UNICON	SDK/IDF
SSL_TCP_UDP_CLIENT_MULTICON	SDK/IDF
TRANSPARENT_CLIENT	SDK/IDF
HTTP_CLIENT	SDK/IDF
SPECIAL_CLIENT	SDK3.3
FIRST_SERVER	SDK/IDF
WEB_SERVER	SDK/IDF
WEBSOCKET_SERVER	SDK/IDF
SECOND_TCP_SERVER	SDK/IDF
MQTT_v31	SDK/IDF
MQTT_V311	SDK/IDF
SOFT_UPGRADE_PIC	SDK/IDF
SSL_SERVER	SDK *
SMTP	SDK/IDF
TRACKING	SDK/IDF
CRYPTOGRAPHIC_FUNC	SDK/IDF
ADC	SDK3.3/IDF
GOOGLE_OAUTH20	SDK/IDF
Herramientas	
JSON	SDK/IDF
STATUS_LED	SDK3.3
INTERNAL_SHA1	SDK/IDF

*Servidor SSL en IDF se habilita en FIRST_SERVER al crear el servidor

La habilitación de las 28 aplicaciones junto con las herramientas se realiza en el archivo de cabecera de la biblioteca, en el cual se modifican los valores de una lista de definiciones referentes a éstas. Además, en el archivo de cabecera se ajustan las características del funcionamiento de la biblioteca y parámetros de algunas de las aplicaciones:

- Habilitar WIFI_ERRORS para detener la ejecución del programa del PIC cuando suceda un error relacionado a la ejecución de comandos AT.
- Habilitar DEBUGGER_MOD que indica mediante el nivel de una salida del PIC, si se presentó a la ejecución de comandos AT cuando se habilita la anterior opción.
- Seleccionar la versión del *firmware* del módulo WiFi que se utiliza.
- Para SOFT_UPGRADE_PIC se indica la contraseña para el punto de acceso cuando se realiza la actualización del programa del PIC de manera inalámbrica por un evento de *software*.
- Para GOOGLE_OAUTH20 se indican parámetros de la aplicación.

La biblioteca ESP fue diseñada como un solo módulo de compilación separado, cuyo código fuente se encuentra seccionado mediante directivas de preprocesador para incluir y compilar únicamente el código de las aplicaciones habilitadas y las funciones que comparten éstas, permitiendo un menor consumo de memoria al incrementar el número de aplicaciones simultaneas. Es por esto por lo que la memoria que consume la biblioteca ESP depende de las aplicaciones habilitadas y, en consecuencia, no se pueda definir de manera general la cantidad de memoria que requiere un microcontrolador PIC para utilizar la biblioteca.

Cada vez que se habilite una aplicación, se consume una cantidad de memoria sin importar que sus funciones no sean llamadas en el programa principal del microcontrolador. Lo anterior es producto de la inclusión del módulo de compilación separada de la biblioteca; esa cantidad de memoria no debe ser tomada como la cantidad mínima requerida en un uC PIC al utilizar cierta aplicación, ya que sólo representa la cantidad de memoria que utiliza la definición de las funciones de ésta, no obstante, permite descartar el uso de la biblioteca ESP en ciertos microcontroladores PIC que cuentan con una menor capacidad de almacenamiento. En la tabla 4-2 se señala la cantidad de memoria que consume la definición individual de cada una de las aplicaciones disponibles de la biblioteca ESP cuando las aplicaciones particulares utilizan STATION como base y se destina a la versión SDK 2.2 (a menos que se indique otra versión).

También la memoria que requiere un uC PIC, depende de la versión del *firmware* del módulo y de lo que el usuario programe en la función principal (*main*) del programa del uC PIC. Para prototipos que requieran alojar grandes cantidades de información, como documentos HTML y correos electrónicos, o que requieran el uso simultaneo de las

aplicaciones como SMTP, MQTT, HTTP y Websocket, el consumo de memoria puede incrementar considerablemente, pero este incremento no está directamente relacionado con las funciones de la biblioteca sino con su utilidad.

Se comprobó que una memoria de programa de 8192 palabras y una SRAM de mínimo 512 bytes permite utilizar la biblioteca ESP con flexibilidad para las distintas combinaciones de aplicaciones simultaneas que ofrece la biblioteca ESP. Además, MPLAB permite consultar la cantidad de memoria que un proyecto utilice, por lo que se puede determinar si el microcontrolador seleccionado es suficiente o se requiere de otro con mayor capacidad de memoria.

Tabla 4-2 *Consumo individual de memoria de la definición de las aplicaciones de la biblioteca ESP*

Aplicación	**Memoria de programa (palabras)**	**RAM (B)**
Aplicación base		
STATION	1088	123
ACCESS_POINT	869	123
Aplicaciones particulares		
SMART_CONFI	1186	123
GPIO (SDK 3.3)	1132	123
PWM (SDK 3.3)	1104	123
RSSI	1336	123
SNTP_CLIENT	1172	123
DNS	1159	123
MDNS	1312	123
ESPNOW	1470	123
SIMPLE_PAIR	1352	123
SSL_TCP_UDP_CLIENT_UNICON	1683	123
SSL_TCP_UDP_CLIENT_MULTICON	1742	123
TRANSPARENT_CLIENT	1315	123
HTTP_CLIENT	1984	123
SPECIAL_CLIENT (SDK 3.3)	1255	123
FIRST_SERVER	1661	123
WEB_SERVER	2475	123
WEBSOCKET_SERVER	2698	217
SECOND_TCP_SERVER	1582	123
MQTT_v31	2022	123
MQTT_V311	2021	123
SOFT_UPGRADE_PIC	1106	123
SSL_SERVER	1438	123
SMTP	2074	123
TRACKING	1253	123
CRYPTOGRAPHIC_FUNC	1275	123
ADC (SDK 3.3)	1217	123
GOOGLE_OAUTH20	1678	123
Herramientas		
JSON+HTTP_CLIENT	2239	123
JSON+WEB_SERVER	2747	166
JSON+WEBSOCKET_SERVER	2797	221
STATUS_LED	1106	123
INTERNAL_SHA1+ WEBSOCKET_SERVER	2581	440

La biblioteca ESP permite habilitar varias aplicaciones en un solo proyecto; sin embargo, existen algunas aplicaciones que no pueden ser activadas al mismo tiempo, ya sea porque utilizan un mismo recurso del uC PIC y/o del módulo, o porque se han restringido en búsqueda de una simplicidad de operación. En las tablas 4-3 y 4-4 se señalan aquellas aplicaciones que no pueden ser utilizadas de manera simultánea; si éstas son activadas, al compilar el proyecto se mostrará un error de compilación indicando que existe incompatibilidad. También es importante señalar que el número de aplicaciones simultaneas que se pueden habilitar depende de la memoria del microcontrolador PIC y del módulo WiFi empleado, por ejemplo, para la versión SDK NONOS 2.2.1 que es la más limitada, se recomienda un máximo de 3 aplicaciones.

Tabla 4-3 *Aplicaciones particulares incompatibles marcadas en color rojo parte* 1.

	SMART_CONFI	GPIO	PWM	RSSI	SNTP_CLIENT	DNS	MDNS	ESPNOW	SIMPLPAIR	SSL_TCP_UDP_CLIENT_UNICON	SSL_TCP_UDP_CLIENT_MULTICON	TRANSPARENT_CLIENT	HTTP_CLIENT	SPECIAL_CLIENT	TCP_SERVER	WEB_SERVER
SMART_CONFI																
GPIO																
PWM																
RSSI																
SNTP_CLIENT																
DNS																
MDNS																
ESPNOW																
SIMPLE PAIR																
SSL_TCP_UDP_CLIENT_UNICON																
SSL_TCP_UDP_CLIENT_MULTICON																
TRANSPARENT_CLIENT																
HTTP_CLIENT																
SPECIAL_CLIENT																
TCP_SERVER																
WEB_SERVER																
WEBSOCKET_SERVER																
SECOND_TCP_SERVER																
MQTT_v31																
MQTT_V311																
SOFT_UPGRADPIC																
SSL_SERVER																
SMTP																
TRACKING																
CRYPTOGRAPHIC_FUNC																
ADC																
GOOGLEOAUTH20																

Tabla 4-4 *Aplicaciones incompatibles marcadas en color rojo parte* 2.

	WEBSOCKET_SERVER	SECOND_TCP_SERVER	MQTT_v31	MQTT_V311	SOFT_UPGRADPIC	SSL_SERVER	SMTP	TRACKING	CRYPTOGRAPHIC_FUNC	ADC	GOOGLOAUTH20
SMART_CONFI								■			
GPIO											
PWM											
RSSI											
SNTP_CLIENT								■			
DNS								■			
MDNS								■			
ESPNOW											
SIMPLPAIR								■			
SSL_TCP_UDP_CLIENT_UNICON	■	■	■	■		■	■	■			
SSL_TCP_UDP_CLIENT_MULTICON								■			
TRANSPARENT_CLIENT	■	■	■	■		■	■	■			■
HTTP_CLIENT								■			
SPECIAL_CLIENT	■	■	■	■		■	■	■			■
TCP_SERVER								■			
WEB_SERVER								■			
WEBSOCKET_SERVER		■						■			
SECOND_TCP_SERVER	■							■			
MQTT_v31								■			
MQTT_V311								■			
SOFT_UPGRADPIC											
SSL_SERVER								■			
SMTP								■			
TRACKING	■	■	■	■		■	■				■
CRYPTOGRAPHIC_FUNC											
ADC											
GOOGLOAUTH20								■			

La recomendación general es utilizar hasta tres aplicaciones particulares de manera simultanea, aparte de las relacionadas con aplicaciones base o de control de periféricos del módulo.

Las modificaciones realizadas en los *firmware* de comandos AT contribuyeron para aprovechar mejor los recursos del microcontrolador PIC al liberar a éste de la ejecución de ciertas tareas, así como también los recursos del módulo que eran desaprovechados en las versiones originales del *firmware.* Un ejemplo de ésto es notado en la aplicación de WEBSOCKET_SERVER, la cual ejecuta el algoritmo SHA-1 durante el establecimiento de una conexión. Dicho algoritmo se puede ejecutar en el programa del microcontrolador a través de la herramienta INTERNAL_SHA1 o a través de las funciones criptográficas que fueron habilitadas como comandos AT en las modificaciones del *firmware*. Ejecutar la

función en el microcontrolador PIC tiene un incremento de 1558 palabras y 171 B en la memoria de programa y en la memoria de datos respectivamente, además que la ejecución del algoritmo demora un tiempo mayor que al ejecutar el nuevo comando AT. Otra aplicación en el que se puede observar este efecto es en SPECIAL_CLIENT, la cual filtra la información que se retrasmite al PIC, haciendo más fácil y rápida la identificación y captura de los mensajes.

Hasta ahora los alcances y limitaciones generales de la biblioteca ha sido descritos, sin embargo, para cada una de las aplicaciones éstas están indicadas con mayor precisión en el subtema *5.2 Manual de prácticas.*

5.2 Manual de prácticas

Con la finalidad de ilustrar el uso de las funciones de la biblioteca ESP se optó por la elaboración de un manual de prácticas, el cual a través de éstas permite ilustrar los alcances y limitaciones de cada una de las aplicaciones desarrolladas y brinda la teoría necesaria para su comprensión. Tal y como se señaló en el *Capítulo 3 Descripción de la propuesta*, varios microcontroladores PIC y módulos WiFi son compatibles con el conjunto propuesto; sin embargo, la elección del módulo WiFi resulta ser la de mayor impacto ya que determina la versión del *firmware* de comandos AT que puede incorporar y por tanto a qué características de la biblioteca se pueden acceder (consultar *Capítulo 4 Desarrollo de firmware*). Es por esto que el manual de prácticas se desarrolló considerando tres diferentes módulos WiFi, los cuales resultan ser representativos de las cuatro diferentes versiones del *firmware* de comandos AT que fueron modificadas.

A continuación, se describen las características de cada uno de estos módulos, así como del microcontrolador PIC empleado.

5.2.1 ESP-01

El módulo ESP-01 (ver figura 4.1) es un módulo WiFi fabricado por la empresa *Ai-Thinker*, el cual tiene incorporado el SoC ESP8266. Este módulo opera a 3.3 *V*, aunque soporta voltajes que van desde 3.0 *V* hasta 3.6 *V*, siendo este último el máximo soportado en cualquiera de sus terminales y puede presentar picos de corriente mayores a 300 *mA[77]*. Además, cuenta con un empaquetado tipo DIP (*Dual In-line Package*) de 24.7*14.4*11.0 *mm* que incorpora 8 diferentes pines, los cuales están identificados en la figura 4.1 y se describen en la tabla 4-5,para brindar acceso a las interfaces UART y GPIO(x2), también cuenta con una antena impresa en su PCB y dos LED indicadores, tanto para la alimentación como para el canal de transmisión TX [73, 77].

Tabla 4-5 *Descripción de pines ESP-01[47, 73, 77].*

Pin	Función
GND	*Ground*
GPIO 2	GPIO, internamente *Pull-Up*
GPIO 0	GPIO, internamente *Pull-Up*
RX	Canal de recepción de datos del puerto serial que se conecta al canal TX de otro dispositivo
TX	Canal de transmisión de datos del puerto serial que se conecta al canal RX de otro dispositivo
CH_PD o CH_EN	Pin de habilitación del chip. Este pin enciende o apaga el SoC ESP8266. Para activarlo debe estar conectado a 3.3 *V*. Cuando está desactivado (conectado a GND) existe un bajo consumo de corriente.
RESET	Pin de reinicio. Para reiniciar el módulo debe pasar a estado bajo (GND)
VCC	Pin de alimentación a 3.3 *V* (Máximo 3.6 *V*)

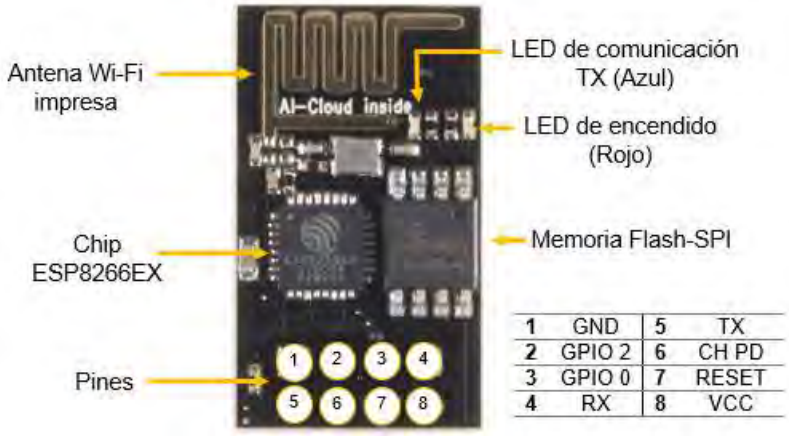


Figura 4.1 *Módulo ESP-01: Modificado de [78].*

Generalmente este módulo incorpora una memoria SPI Flash de 8Mbit, razón por la que fue seleccionado para utilizar el *firmware* modificado de comandos AT que está basado en el SDK NONOS v2.2.1. Para más información sobre este módulo consultar [77]. Actualmente, existe una nueva versión del módulo llamada ESP-01S, la cual presenta una reconfiguración del *hardware* sobre la tarjeta del módulo, conservando en esencia las mismas características del módulo ESP-01. Dentro de los cambios más notables en esta nueva versión es la aparición de un solo LED indicador para el GPIO 2, además de ya ser opcional la conexión a VCC de los pines CH_PD y RST [73].

5.2.2 ESP-07S

El módulo ESP-07S (ver figura 4.2) es un módulo WiFi fabricado por la empresa *Ai-Thinker*, el cual tiene incorporado el SoC ESP8266 y opera normalmente a 3.3 *V.* Soporta voltajes que van desde 3.0 *V* hasta 3.6 *V*, siendo este último el máximo en cualquiera de sus terminales y puede presentar picos de corriente mayores a 500 *mA.* Este módulo presenta un empaquetado SMD (*Surface Mounting Device*) de 17*16*3 *mm* que brinda acceso a 16 terminales del SoC que incorpora, las cuales están identificadas en la figura 4.2 y descritas en la tabla 4-6 [47, 79]. Este empaquetado brinda acceso a las interfaces UART, GPIO(x9), PWM y ADC. Además, permite la utilización de una antena externa a través de una interfaz IPEX [79].

Generalmente este módulo incorpora una memoria SPI flash de 32Mbits, razón por la que fue seleccionado para utilizar el *firmware* modificado de comandos AT que está basado en el SDK NONOS v3.0.3. Para más información sobre este módulo consultar [79]. Si bien la versión ESP-IDF del *firmware* de comandos AT del ESP8266 está originalmente diseñada para los módulos de Espressif Systems, las pruebas realizadas han mostrado que este módulo también es capaz de soportarla.

Tabla 4-6 *Descripción de pines ESP-07S[47, 79].*

Pin	Función
RST	Pin de reinicio. Para reiniciar el módulo debe pasar a estado bajo (GND)
ADC	Convertidor analógico a digital cuyo rango de voltaje de entrada es de 0-1*V,* el rango de valores es 0-1024.
EN	Pin de habilitación del chip. Este pin enciende o apaga el SoC ESP8266. Para activarlo debe estar conectado a 3.3 *V.* Cuando está desactivado (conectado a GND) existe un bajo consumo de corriente.
IO16	Conectar con el pin RST para despertar de *Deep Sleep*
IO14	GPIO14; HSPI_CLK; PWM2
IO12	GPIO12; HSPI_MISO; PWM0
IO13	GPIO13; HSPI_MOSI; UART0_CTS
VCC	Pin de alimentación a 3.3 *V* (máximo 3.6 *V*)
GND	*Ground* (tierra)
IO15	GPIO15; MTDO; HSPICS; UART0_RTS; PWM1
IO2	GPIO2; UART1_TXD
IO0	GPIO0; HSPI_MISO; I2SI_DATA
IO4	GPIO14; PWM3
IO5	GPIO5; IR_R
RX	Canal de recepción de datos del puerto serial que se conecta al canal TX de otro dispositivo; GPIO3
TX	Canal de transmisión de datos del puerto serial que se conecta al canal RX de otro dispositivo; GPIO1

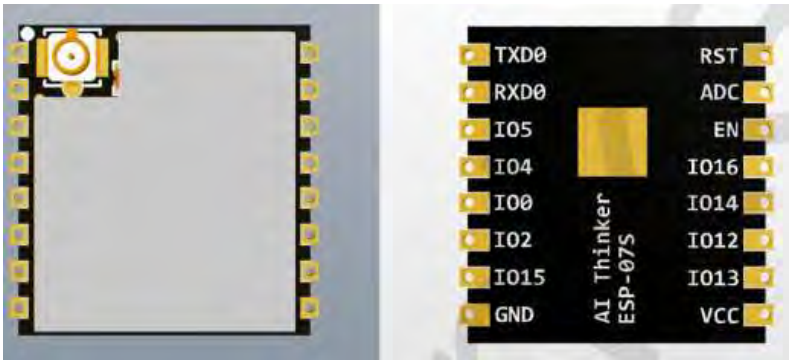


Figura 4.2 *Módulo ESP-07S[79].*

5.2.3 ESP32-DevKitC V4

El ESP32_DevKitC (ver figura 4.3) es una tarjeta de desarrollo de Espressif Systems construida generalmente a partir del módulo WiFi ESP32-WROOM-32 que a su vez integra un SoC ESP32; sin embargo, puede tener incorporado cualquiera de los siguientes módulos WiFi:

- ESP32-WROOM-32
- ESP32-WROOM-32D
- ESP32-WROOM-32U
- ESP32-SOLO-1
- ESP32-WROVER
- ESP32-WROVER-B
- ESP32-WROVER-I
- ESP32-WROVER-B (IPEX)[57].

Figura 4.3 *ESP32_DevKitC V4. Modificado de [57].*

Esta tarjeta de desarrollo de 54.4x27.9 *mm* trabaja a 5 *V* o 3.3 *V* y presenta características como:

- Una interfaz de programación USB-UART que provee tasas de transferencia de hasta 3 *Mbps* y una fuente de alimentación para la tarjeta.
- *Headers* macho (conectores tipo macho) en los pines.
- Botón pulsador para el reinicio y activación del modo *Firmware Download*
- Un LED indicador de alimentación a 5 V [57].

Las razones que llevaron a la selección de esta tarjeta de desarrollo son:

- Implementa un módulo que integra un SoC ESP32, lo cual permitió probar el trabajo desarrollado para los SoC de este tipo.
- Esta tarjeta de desarrollo puede utilizar el *firmware* de comandos AT que fue modificado y que está basado en ESP-IDF ESP32.
- Permite comprobar que las tarjetas de desarrollo basadas en un módulo WiFi compatible con el conjunto microcontrolador-módulo WiFi propuesto, también pueden ser utilizadas con un micrcoontrolador PIC.

5.2.4 PIC 16LF1939

El microcontrolador que se seleccionó para realizar las pruebas de la biblioteca ESP y para el manual de prácticas es el PIC 16LF1939 fabricado por la empresa *Microchip Technology Inc*, el cual pertenece a la familia de microcontroladores 16F193X que está constituida por seis diferentes microcontroladores, cada uno con una versión de bajo consumo energético, y es también el que presenta mejores características [80] , tal y como se puede observar en la tabla *4-7*.

El PIC 16LF1939 es un microcontrolador de 8 bits catalogado como un microcontrolador PIC de gama media mejorada debido al CPU que incorpora con un repertorio de 49 instrucciones; además, cuenta con una pila de 16 niveles, un módulo de detección capacitiva, un módulo controlador de LCD estática o multiplexada, un módulo EUSART (compatible con RS-232, RS-485 y LIN) y tiene la capacidad de implementar interrupciones que se activan debido al cambio de estado en alguna terminal del puerto B, por captura, por comparación, al desbordamiento de un temporizador o por recepción de datos RS-232, entre otras más[80].

La diferencia entre el 16F1939 y el 16LF1939 , es que este último es una versión de bajo consumo del primero, cuyo voltaje de operación va desde 1.8 *V* hasta 3.6 *V*[81]. Las características más importantes de este microcontrolador se resumen en la tabla *4-8*. Al ser un microcontrolador de 8 bits, permite que sea más fácil su implementación al ser su arquitectura relativamente sencilla.

Tabla 4-7 Características de la familia uC PIC 16L/F193X: Adaptado de [80].

Microcontrolador PIC	Memoria Flash de Programa (palabras)	EEPROM de datos (bytes)	SRAM (bytes)	I/O	Convertidor A/D 10-bit (canales)	Módulo de detección capacitiva	Comparadores	Temporizadores 8/16-bit	EUSART	I2C/PSI	ECCP	CCP	LCD (Com/Seg/Total)
16F1933 16LF1933	4096	256	256	25	11	8	2	4/1	Si	Si	3	2	4/16/60*
16F1934 16LF1934	4096	256	256	36	14	16	2	4/1	Si	Si	3	2	4/24/96
16F1936 16LF1936	8192	256	512	25	11	8	2	4/1	Si	Si	3	2	4/16/60*
16F1937 16LF1937	8192	256	512	36	14	16	2	4/1	Si	Si	3	2	4/24/96
16F1938 16LF1938	16384	256	1024	25	11	8	2	4/1	Si	Si	3	2	4/16/60*
16F1939 16LF1939	16384	256	1024	36	14	16	2	4/1	Si	Si	3	2	4/24/96

**COM3 y SEG15 comparten físicamente el mismo pin, por lo tanto, SEG15 no está disponible cuando se usa el multiplexado de pantallas 1:4.*

Tabla 4-8 Características del microcontrolador 16LF1939: Adaptado de [81]

Característica	**Valor**
Memoria flash (memoria de programa)	28 kB
Velocidad de CPU (MIPS/DMIPS)8	8
SRAM	1024 B
EEPROM/HEF de datos	256 B
Periféricos de comunicación digital	1-UART, 1-SPI,1-I2C1-MSSP(SPI/I2)
Periféricos de captura/comparación/ PWM	5 pines de captura (entrada), 2 CCP, 3 ECCP
Temporizadores	4x8-bit (TMR0/TMR2/TMR4/TMR6) 1x16-bit (TMR1)
Entradas ADC	14 canales, 10-bit con voltaje de referencia
Número de comparadores	2 (con selector de voltaje de referencia)
LDC segmentadas	96
Rango de temperatura	-40 a +125 (°C)
Oscilador Interno	32 MHz
Source/sink current I/O (Entrada/salida de corriente tipo fuente/sumidero)	25 mA

5.2.5 *Software* externo empleado

Para el desarrollo de algunas de las prácticas se utilizaron herramientas de terceros entre las que se encuentra HTerm, el cual es un programa que emula una terminal para la comunicación en serie y puede ser descargado de forma gratuita desde el siguiente enlace:

http://www.der-hammer.info/pages/terminal.html

También se hizo uso del programa *Packet Sender,* el cual es un programa que permite enviar y recibir paquetes de datos a través de la red mediante protocolos como TCP, UDP y SSL, ya sea como cliente o como servidor. Este programa puede ser descargado de forma gratuita desde el siguiente enlace:

https://packetsender.com/

Práctica 0 Biblioteca ESP: uso de la biblioteca ESP en un proyecto

Introducción

La biblioteca ESP permite a los microcontroladores PIC acceder a redes inalámbricas WiFi y a servicios en internet mediante el control de algún módulo que incorpore el SoC ESP8266 o los ESP32. Esta biblioteca fue diseñada para *CCS Compiler* como un módulo de compilación separada, que está compuesta por dos archivos: esp.c que contiene el código fuente de las funciones de cada una de las aplicaciones disponibles; y esp.h que es el archivo de cabecera donde el usuario ajusta y habilita las características de la biblioteca.

Cada vez que se utilice la biblioteca ESP en un proyecto creado en la IDE MPLAB, se debe incluir tanto el archivo de cabecera como el código fuente. En el archivo fuente (esp.c) se tiene que incluir el *driver* del microcontrolador PIC utilizado y ajustar las características de la biblioteca y del microcontrolador en el archivo de cabecera (esp.h).

En el archivo de cabecera se ajustan los siguientes aspectos:

- La directiva `#use delay` para configurar la frecuencia de trabajo del microcontrolador PIC con algún valor que pueda generar la tasa 115.200 baudios requerida en el serial del *hardware* del uC, ya sea para un oscilador interno o externo.
- Los periféricos del microcontrolador, que ajustan la comunicación serial y el temporizador que utiliza la biblioteca y la frecuencia de trabajo.
- El modo de operación de la biblioteca, que define como la biblioteca actúa ante la presencia de algún error.
- La selección de la versión del *firmware* del módulo utilizado.
- La habilitación de las aplicaciones que requiere en el proyecto.

Cada uno de los ajustes se realiza cambiando el valor de las definiciones (#define) correspondientes. Los valores pueden ser TRUE/FALSE o un algún valor numérico.

Las definiciones involucradas en el UART creado por *software* para el uso del usuario están señaladas en la tabla P0.1. En éstas se ajusta la tasa de baudios junto con los pines destinados para este fin.

Tabla P0.1 *Definiciones relacionadas con la comunicación serial virtual.*

Definición	Valor
BAUD_PIC	57600, 56000, 38400,19200
RX_PIC	Etiqueta del pin RX
TX_PIC	Etiqueta del pin TX

Las definiciones relacionadas con el temporizador utilizado por la biblioteca y con los tiempos de respuesta se muestran en la tabla P0.2. En éstas, se indica la frecuencia de trabajo con la que el microntrolador PIC trabaja, se ajusta si el archivo de cabecera del microcontrolador usa los alias TMR_DIV_BY_## para referirse a los valores del predivisor del temporizador seleccionado o se refiere a estos valores con los alias T(número de *timer*)_DIV_##, si es el segundo caso entonces TIMER_TMR debe estar deshabilitado (*FALSE*). Además, se tiene que establecer el número del temporizador que se designa para el funcionamiento de la biblioteca, junto los valores disponibles de predivisor y de postdivisor para el temporizador seleccionado, estableciendo los valores más grandes posibles.

Después de estos ajustes, se encuentra la sección de definiciones relacionadas con la ejecución de la biblioteca ante la presencia de un error derivado del reinicio inesperado del módulo WiFi o de algún comando AT. Estas definiciones se encuentran en la tabla P0.3. Cuando WiFi ERRORS está deshabilita el uC PIC sigue ejecutando su programa aun cuando surja un error, en caso de que sea por reinicio inesperado únicamente se bloquearán las funciones relacionadas con el módulo. Si esta definición es habitada la ejecución del programa se bloqueará (estará en un *loop* infinitamente) teniendo la opción de habilitar una salida (DEBUGGER_MOD) como un indicador de que este bloqueo se ha dado (LtoH).

Tabla P0.2 *Definiciones relacionadas con el temporizador.*

Definición	Valor
WORK_FREQUENCY	Valor numérico
TIMER_TMR	TRUE/FALSE
TIMER	1, 2, 3, 4, 5, 6, 7, 8, 9
POSTSCALER	Valor numérico
PRESCALER	1, 8, 64, 256

Tabla P0.3 *Definiciones relacionadas con la ejecución la biblioteca ante la presencia de un reinicio inesperado o fallo en algún comando AT.*

Definición	Valor
WIFI_ERRORS	TRUE/FALSE
DEBUGGER_MOD	TRUE/FALSE
DEBUGGER_PIN	Etiqueta de un pin

Las definiciones que ajustan la versión del *firmware* del módulo utilizado se observan en la tabla P0.4, de las cuales se habilita sólo una de las cuatro opciones.

Tabla P0.4 *Definiciones relacionadas con la versión de firmware del módulo.*

Definición	Valor
SDK_V2	TRUE/FALSE
SDK_V3	TRUE/FALSE
IDF ESP32	TRUE/FALSE
IDF ESP8266	TRUE/FALSE

Finalmente, las aplicaciones que se requieren en un proyecto son habilitadas en la lista de aplicaciones. Se puede en habilitar una o varias de éstas, incluyendo las herramientas (ver tabla P0.5). La cantidad de aplicaciones habilitadas al mismo tiempo depende principalmente de la memoria del uC PIC y módulo empleado, así como la compatibilidad de las aplicaciones entre sí. Siempre deben estar habilitadas E_STATION o E_ACCESS_POINT.

Tabla P0.5 *Definiciones de las aplicaciones y herramientas disponibles.*

Aplicación	**Valor**
Aplicación base	
E_STATION	TRUE/FALSE
E_ACCESS_POINT	TRUE/FALSE
Aplicaciones particulares	
E_SMART_CONFI	TRUE/FALSE
E_GPIO	TRUE/FALSE
E_PWM	TRUE/FALSE
E_RSSI	TRUE/FALSE
E_SNTP_CLIENT	TRUE/FALSE
E_DNS	TRUE/FALSE
E_MDNS	TRUE/FALSE
E_ESPNOW	TRUE/FALSE
E_SIMPLE_PAIR	TRUE/FALSE
E_SSL_TCP_UDP_CLIENT_UNICON	TRUE/FALSE
E_SSL_TCP_UDP_CLIENT_MULTICON	TRUE/FALSE
E_TRANSPARENT_CLIENT	TRUE/FALSE
E_HTTP_CLIENT	TRUE/FALSE
E_SPECIAL_CLIENT	TRUE/FALSE
E_FIRST_SERVER	TRUE/FALSE
E_WEB_SERVER	TRUE/FALSE
E_WEBSOCKET_SERVER	TRUE/FALSE
E_SECOND_TCP_SERVER	TRUE/FALSE
E_MQTT_v31	TRUE/FALSE
E_MQTT_V311	TRUE/FALSE
E_SOFT_UPGRADE_PIC	TRUE/FALSE
E_SSL_SERVER	TRUE/FALSE
E_SMTP	TRUE/FALSE
E_TRACKING	TRUE/FALSE
E_CRYPTOGRAPHIC_FUNC	TRUE/FALSE
E_ADC	TRUE/FALSE
E_GOOGLE_OAUTH20	TRUE/FALSE
Herramientas	
E_JSON	TRUE/FALSE
E_STATUS_LED	TRUE/FALSE
E_INTERNAL_SHA1	TRUE/FALSE

Objetivo

Crear un proyecto en MPLAB que use la biblioteca ESP

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF

Descripción

Se creará un proyecto en MPLAB IDE v8.90 (replicable para MPLAB X IDE), en el que se mostrarán los pasos a seguir para utilizar la biblioteca ESP en este entorno de desarrollo.

El primer paso será crear un proyecto en MPLAB utilizando el *Project wizard*. En el primer paso, se especifica el microcontrolador PIC que se utilizará, que en esta práctica corresponde al PIC16LF1939. En el segundo paso se seleccionará CCS Compiler como *language toolsuite*. Para el tercer paso se indicará el directorio del proyecto y en cuarto paso se agregarán al proyecto los archivos de la biblioteca (ver figura P0.1), los cuales pueden ser descargados en:

https://bit.ly/2BsLtJt

Una vez creado el proyecto, en la carpeta de archivos fuentes se agregará un nuevo archivo con terminación (.c), que será en el que se coloque el programa principal del microcontrolador PIC.

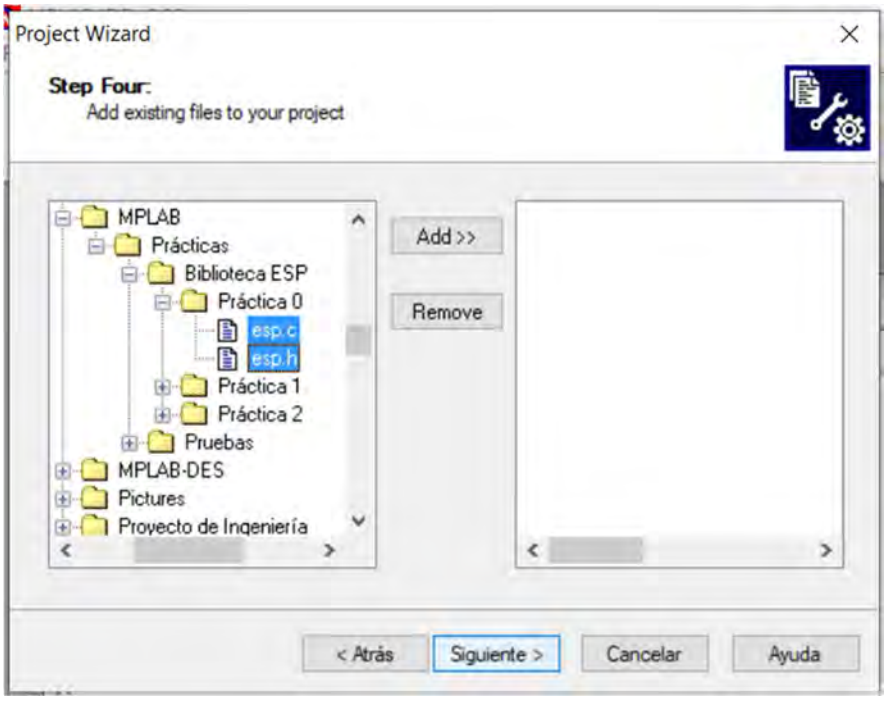


Figura P0.1 *Inclusión de los archivos de la biblioteca ESP.*

El proyecto quedará compuesto por tres archivos, tal como se muestra en la figura P0.3.

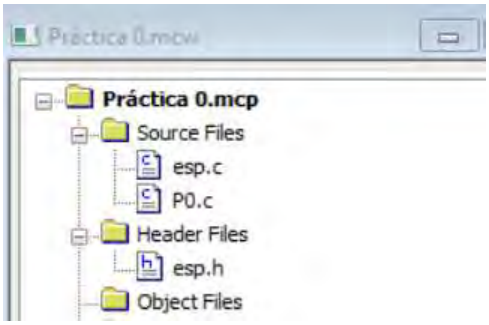


Figura P0.3 *Composición de un proyecto que utilice la biblioteca ESP.*

Se abrirá el archivo fuente de la biblioteca y en la primera línea de éste se incluye el *driver* del microcontrolador utilizado en el proyecto, que en esta práctica corresponde a "#include <16lf1939.h>" tal como se muestra en la figura P0 2. Se cierra y se guardan los cambios de este archivo.

Posteriormente se abrirá el archivo de cabecera y se ajustarán los parámetros relacionados con los periféricos del microcontrolador. Se seleccionará una frecuencia de reloj suficiente para alcanzar 115.200 *baudios* que requiere la comunicación del PIC y el módulo ESP (sin ningún prefijo como MHz). El *baudrate* del serial que podrá utilizar el usuario y que sirve para comunicar al PIC con una terminal u otro periférico. Para este ejemplo se dejará con el valor por defecto 57600 y los pines C1 y C0.

Para el temporizador que requiere la biblioteca ESP, en este caso, se seleccionará el temporizador número cuatro del PIC16LF1939 y ya que las etiquetas del predivisor de este temporizador es del tipo T4_DIV_, se deshabilita la opción de TMR. En este ejemplo se seleccionará un valor de 16 y 64 como postdivisor y predivisor respectivamente (ver figura P0.4).

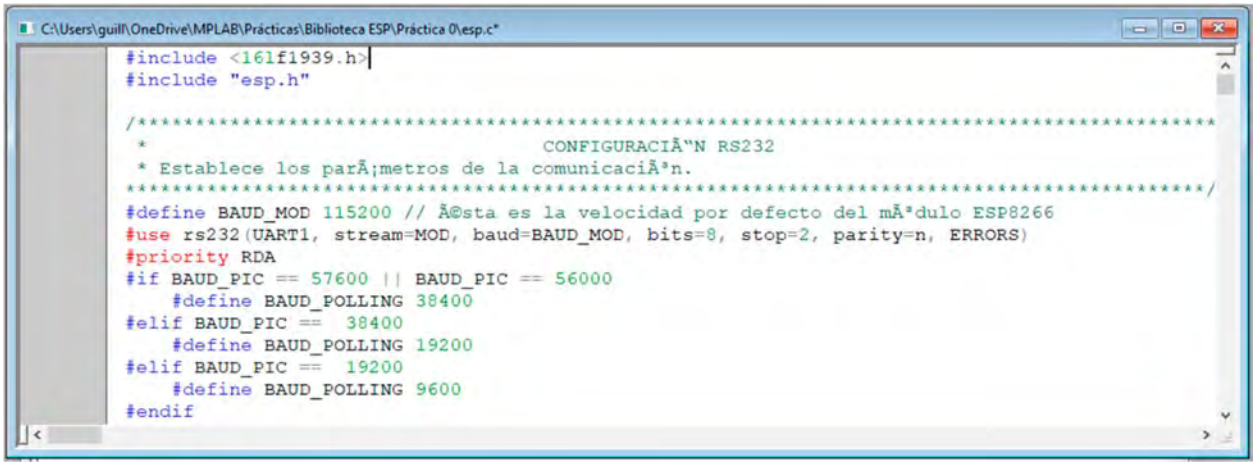


Figura P0 2 *Inclusión del driver del microcontrolador PIC utilizado en el proyecto.*

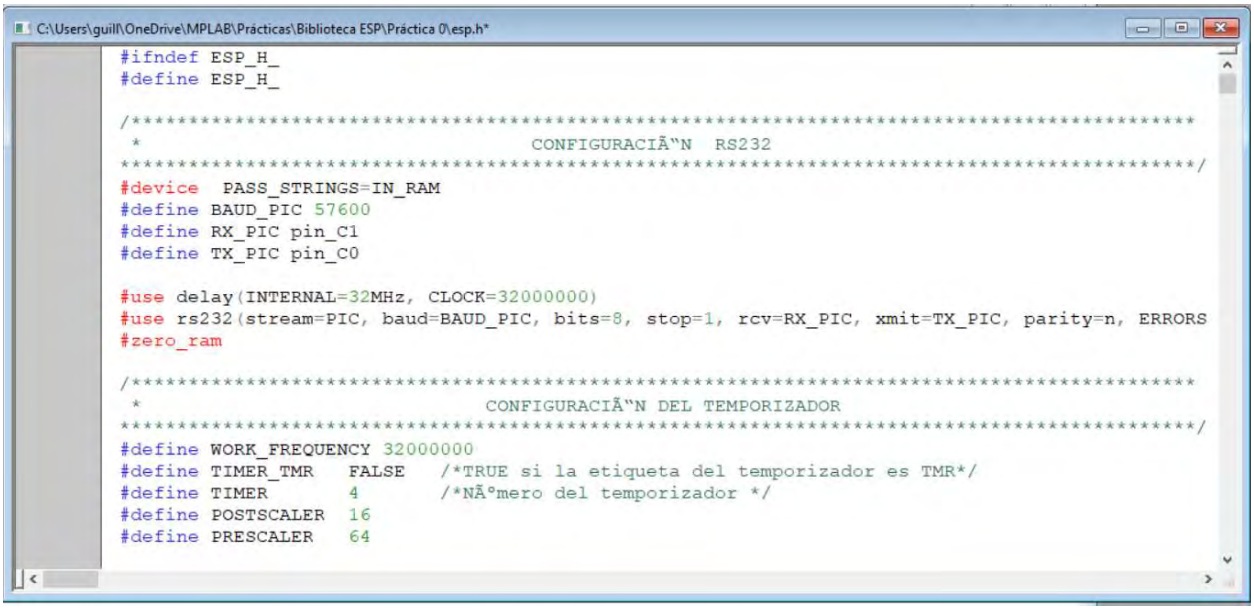


Figura P0.4 *Ajuste relacionados con los periféricos del microcontrolador PIC.*

Despues de configurar los parámetros de los periféricos, se ajustará el comportamiento de la biblioteca. Para la definición de errores, ésta se deshabilita para que cuando suceda algún error relacionado con la biblioteca, el programa del PIC siga ejecutándose, por lo que no es necesario habilitar el pin del *debugger* que indicaría si se ha producido un error en la biblioteca. En esta práctica se utiliza el ESP-01 que tiene el *firmware* SDK en la versión 2.2.1, por lo que se habilita su respectiva opción (ver figura P0.5).

Después de todos los parámetros configurados, en la lista de aplicaciones se habilitará cualquiera de las

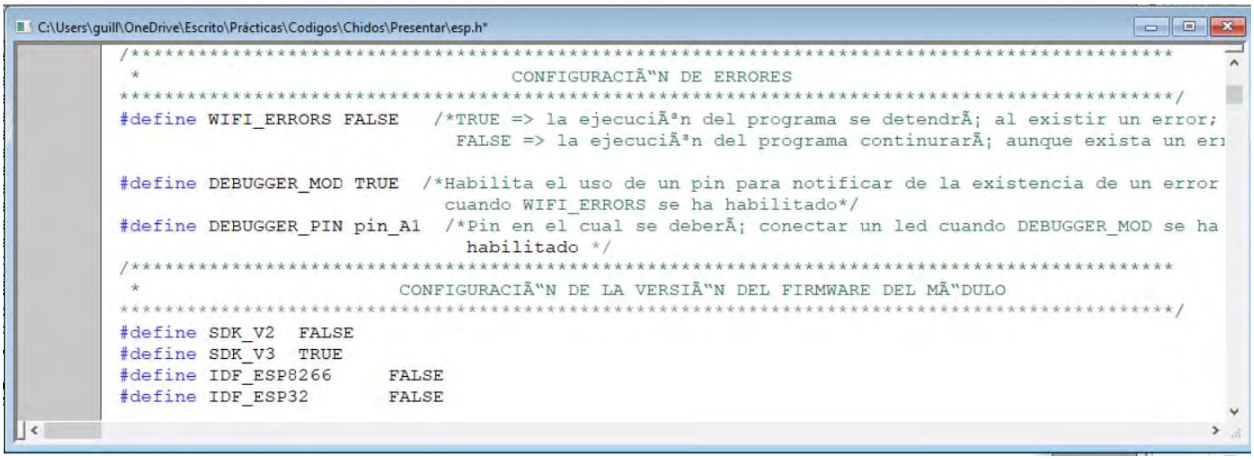


Figura P0.5 *Ajuste con el funcionamiento de la biblioteca ESP y la versión del módulo ESP utilizado.*

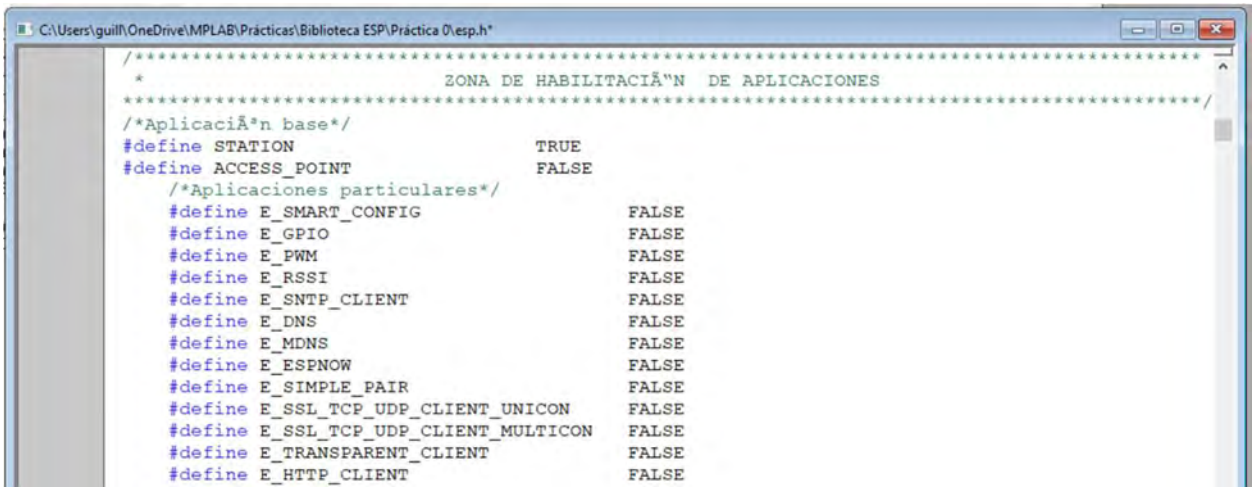


Figura P0.6 *Habilitación de las aplicaciones que se requieren.*

aplicaciones que se requieran, aunque siempre se deberá habilitar alguna de las aplicaciones base (ver figura P0.6).

Con lo anterior se ha completado la configuración de la biblioteca ESP, ahora solo restará escribir el código del programa principal. En la primera línea se incluirá el *driver* del microcontrolador PIC utilizado, en la segunda se incluirá la biblioteca ESP y se coloca en la función principal la inicialización del módulo (ver figura P0.6).

Finalmente se compilará todo el proyecto (*Build all*) y se obtendrá como resultado lo que se muestra en la figura P0.9, en la que se observa dos unidades de compilación, la primera pertenece a la sección del código utilizado de la biblioteca ESP y el segundo del programa principal del microcontrolador PIC.

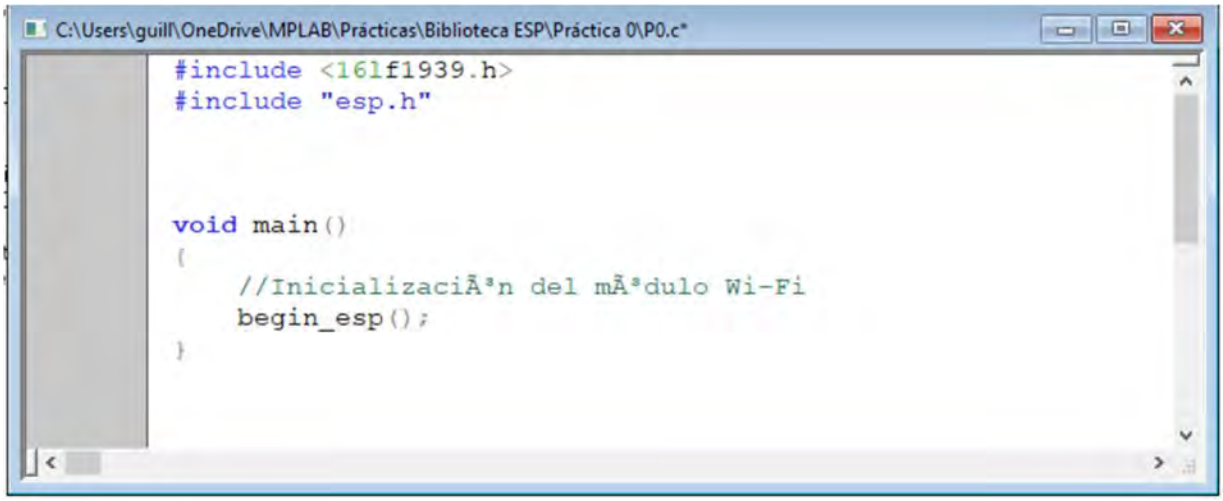


Figura P0.7 *Programa principal del PIC en el que se inicializa el módulo ESP.*

Figura P0.9 *Compilación del proyecto de la práctica 0.*

Para esta actividad se requiere de algún terminal serial para visualizar que la comunicación serial entre PIC y módulo es correcta. En el terminal se seleccionará el COM que fue asignado al adaptador TTL, se seleccionará una tasa de 57600, se colocará que la cantidad bits de un dato es 8 (1 *byte*), un solo *bit* de parada y sin *bit* de paridad.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P0.10.

Resultados

Al ejecutar el programa del microcontrolador PIC, se mostrará en el terminal el mensaje de “Módulo esp listo para utilizarse” que indica que la inicialización del módulo es correcta (ver figura P0 8).

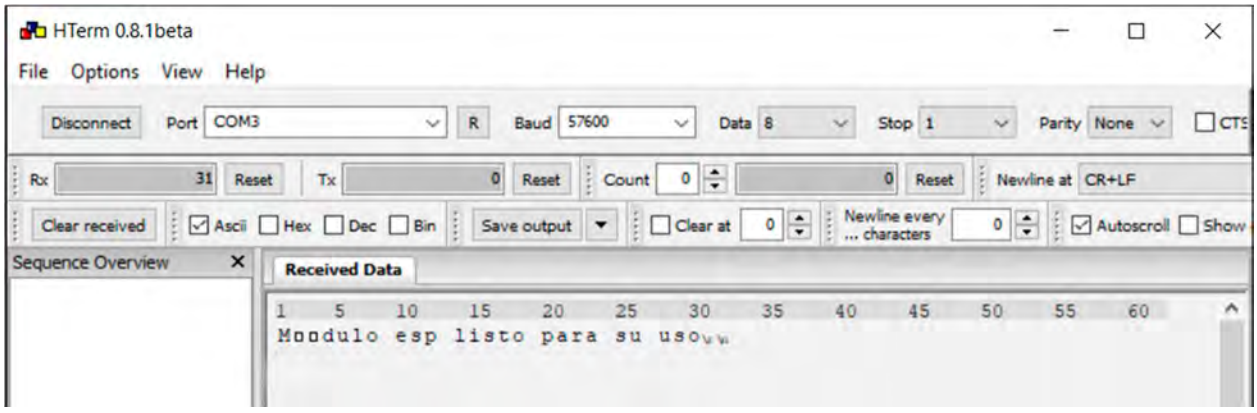


Figura P0 8 *Mensaje de inicialización correcta del módulo ESP.*

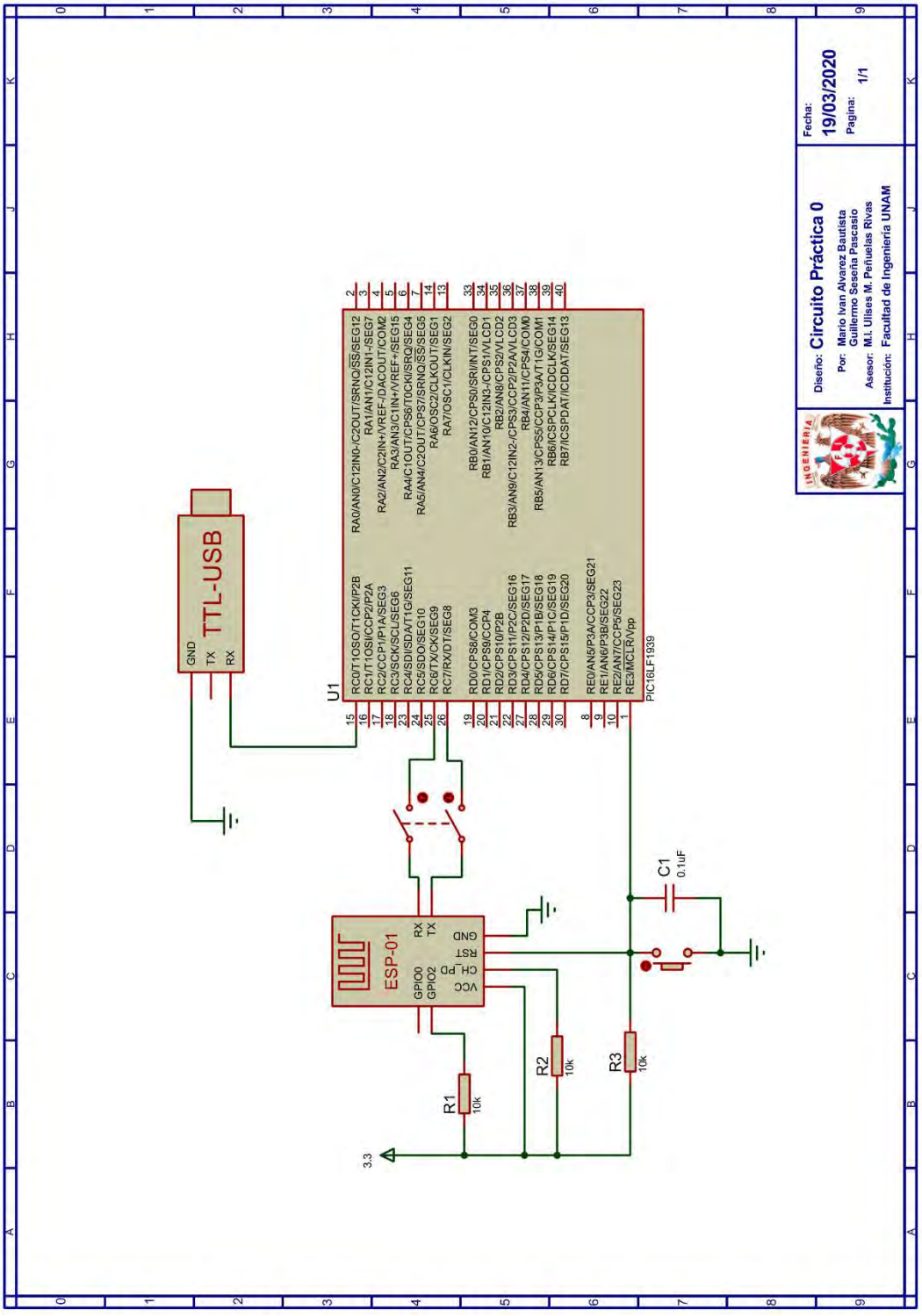


Figura P0.10 *Circuito de la práctica 0.*

Nota

- Como la biblioteca hace uso de la directiva *device PASS_STRINGS = IN_RAM,* el usuario podrá utilizar *strings* como parámetros de funciones.

Práctica 1-A Estación: conexión a un punto de acceso de un módulo ESP

Introducción

Una estación es un dispositivo que cuenta con un adaptador WiFi que le permite conectarse a una red para proveer y/o utilizar servicios [1]. Los módulos WiFi que incorporan el SoC ESP8266 o los ESP32 pueden funcionar como estaciones y están diseñados para operar en modo infraestructura, es decir, que se deben conectar a un punto de acceso, el cual gestionará las conexiones y transmitirá todos los paquetes [1-3]. Al mismo tiempo, éstos están configurados como estaciones estáticas, por lo que no es posible comunicarse entre redes creadas por diferentes puntos de acceso [1-3].

La estación está identificada tanto por una dirección MAC como por una dirección IP, esta última generalmente es asignada por el punto de acceso [4]. Para conectarse a un punto de acceso se debe estar dentro del área de cobertura, conocer su SSID (*Service Set IDentifier*) y contraseña [1].

La biblioteca ESP permite realizar las siguientes acciones cuando el módulo funciona como una estación:

- Conectarse a un punto de acceso.
- Obtener o establecer la dirección IP asignada al módulo como estación.
- Obtener o establecer la dirección MAC del módulo como estación.
- Desconectarse de un punto de acceso.

Objetivo

Mostrar como conectarse a un punto de acceso.

Firmware de comandos AT

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

Se conectará el módulo WiFi a un punto de acceso. Tras la conexión se deberá mostrar en una terminal la dirección IP que le fue asignada al módulo WiFi.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P1-A.2, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P1-A.3.

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC, el primer mensaje que se mostrará en la terminal corresponderá al que notifica del éxito o fracaso de la conexión del módulo al punto de acceso. De ser exitosa la conexión, el siguiente mensaje corresponderá a la dirección IP que le fue asignada al módulo WiFi (ver figura P1-A.1)

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3emfvgI

Notas

- Se recomienda usar el programa Hterm para simular la terminal.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.

```
Moodulo esp listo para su uso\r\n
Conexioon exitosa\r\n
La direccion IP asignada es:192.168.43.243\r\n
```

Figura P1-A.1 *Mensaje de conexión.*

```
#include <16lf1939.h>
#include "esp.h"

int err=0;
char IP[16]={0};
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    err=begin_station("ESP","P1_A","123456789");
    if(err==0)
    {
        fprintf(PIC,"Conexión exitosa\r\n");
        //Obtención de la dirección IP
        get_ip_station(IP,sizeof(IP));
        fprintf(PIC,"La dirección IP asignada es:%s\r\n",IP);
    }
    else
        fprintf(PIC,"No fue posible conectarse al AP\r\n");
    while(1)
    {
    }
}
```

Figura P1-A.2 *Código de la práctica 1-A*

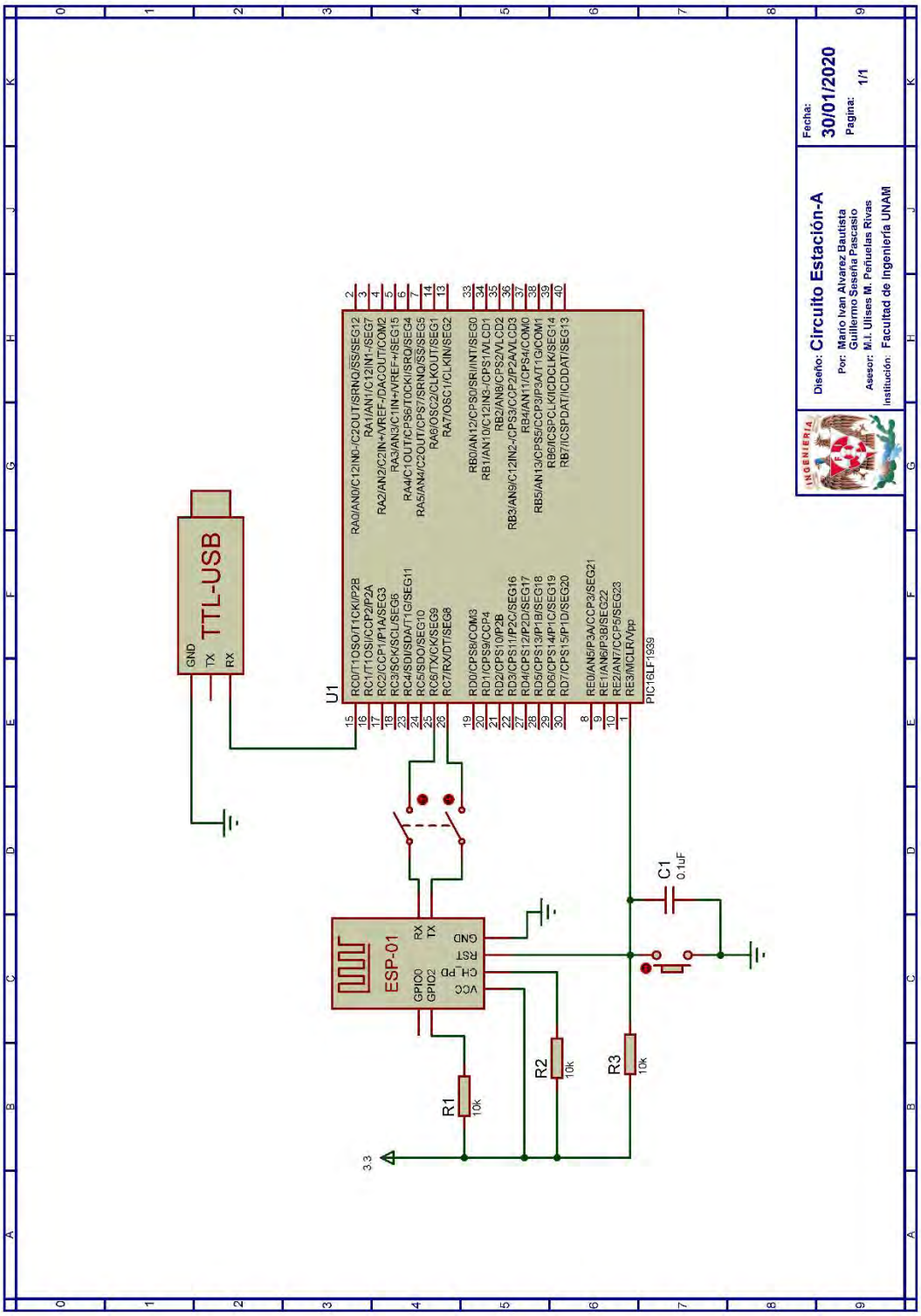


Figura P1-A.3 *Circuito de la práctica 1-A.*

Referencias

1. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
2. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
3. Espressif Systems, *ESP8266 Non-OS SDK API Reference Version 3.0.1*. 2019.
4. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.

Práctica 1-B Estación: conexión a un punto de acceso de un módulo ESP y uso de la herramienta estatus LED

Introducción

Una estación es un dispositivo que cuenta con un adaptador WiFi que le permite conectarse a una red para proveer y/o utilizar servicios [1]. Los módulos WiFi que incorporan el SoC ESP8266 o los ESP32 pueden funcionar como estaciones y están diseñados para operar en modo infraestructura, es decir, que se deben conectar a un punto de acceso, el cual gestionará las conexiones y transmitirá todos los paquetes [1-3]. Al mismo tiempo, éstos están configurados como estaciones estáticas, por lo que no es posible comunicarse entre redes creadas por diferentes puntos de acceso [1-3].

La estación está identificada tanto por una dirección MAC como por una dirección IP, esta última generalmente es asignada por el punto de acceso [4]. Para conectarse a un punto de acceso se debe estar dentro del área de cobertura, conocer su SSID (*Service Set IDentifier*) y contraseña [1].

La biblioteca ESP permite realizar las siguientes acciones cuando el módulo funciona como una estación:

- Conectarse a un punto de acceso
- Obtener o establecer la dirección IP asignada al módulo como estación.
- Obtener o establecer la dirección MAC del módulo como estación.
- Desconectarse de un punto de acceso.
- Establecer un GPIO del módulo para estatus LED.

También está disponible la herramienta estatus LED que utiliza un GPIO del módulo WiFi, al cual se le conecta un LED, para notificar si el módulo WiFi está conectado a un punto de acceso o no. Si no está conectado el LED parpadeará, en caso contrario dejará de parpadear. Esta herramienta resulta útil cuando no se tiene una interfaz para verificar el estado de la conexión.

Objetivo

Mostrar como usar la herramienta estatus LED y reconocer su utilidad.

Firmware de comandos AT

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3, por lo que en esta práctica se empleará el módulo ESP-07S y únicamente SDK_V3 debe ser TRUE en esp.h.

Habilitación de aplicaciones

STATION y STATUS_LED

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 teléfono inteligente
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 mA

Descripción

Se deberá crear un punto de acceso en un teléfono inteligente. Una vez creado se habilitará la herramienta estatus LED y se deberá conectar el módulo WiFi a dicho punto de acceso. Cuando el teléfono haya detectado la conexión de una estación, se deberá eliminar el punto de acceso. En todo este proceso se debe observar el comportamiento del LED.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P1-B.1, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El cálculo del resistor para el uso de un LED:

$$R = \frac{V_{pin} - V_{LED}}{I_{LED}} = \frac{3.3V - 2.2V}{20mA} = 55\Omega$$

Aunque se utilizará un resistor de 56Ω ya que éste es un valor comercial. El circuito para llevar a cabo esta práctica se muestra en la figura P1-B.2.

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
   //Inicialización del módulo Wi-Fi
   begin_esp();
   //Habilitación del GPIO para el estatus LED
   enable_status_led(12);
   //Conexión al punto de acceso indicado
   begin_station("ESP","SSID","123456789");
   while(1)
   {
   }
}
```

Figura P1-B.1 *Código de la práctica 1-B.*

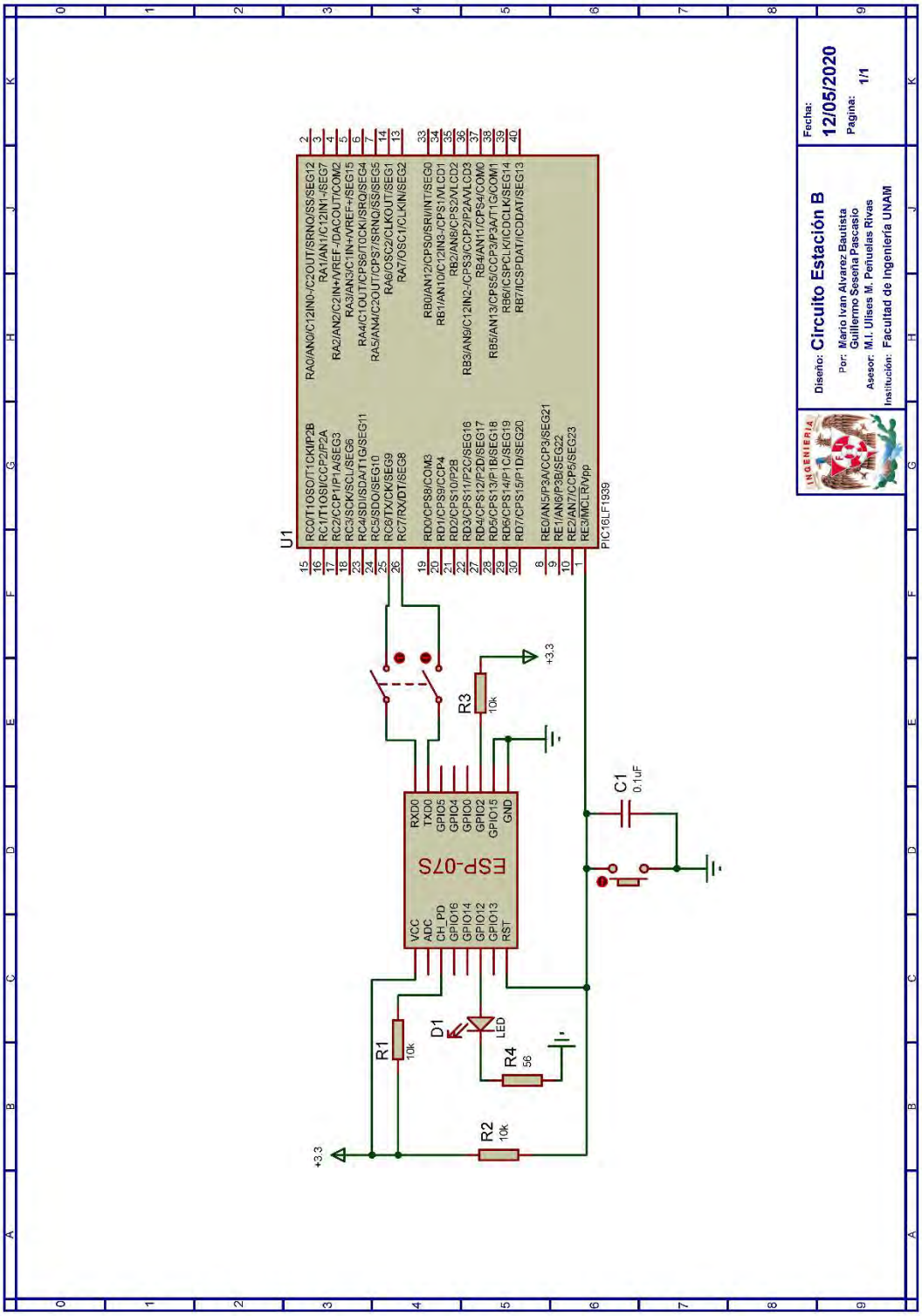


Figura P1-B.2 *Circuito de la práctica 1-B*.

Resultados

Cuando el LED comience a parpadear significará que el módulo está llevando a cabo el proceso de conexión al punto de acceso especificado. Si el LED deja de parpadear, el módulo se habrá conectado exitosamente, en caso contrario, la conexión habrá fallado, esto puede deberse a que el punto de acceso no fue encontrado o a que el SSID y/o contraseña dados eran incorrectos

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2NbhHLU

Notas

- La herramienta estatus LED siempre utiliza un GPIO del módulo WiFi.

Referencias

1. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
2. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
3. Espressif Systems, *ESP8266 Non-OS SDK API Reference Version 3.0.1*. 2019.
4. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.

Práctica 1-C Estación: conexión a un punto de acceso con ingreso de parámetros desde una terminal

Introducción

Una estación es un dispositivo que cuenta con un adaptador WiFi que le permite conectarse a una red para proveer y/o utilizar servicios [1]. Los módulos WiFi que incorporan el SoC ESP8266 o los ESP32 pueden funcionar como estaciones y están diseñados para operar en modo infraestructura, es decir, que se deben conectar a un punto de acceso, el cual gestionará las conexiones y transmitirá todos los paquetes [1-3]. Al mismo tiempo, éstos están configurados como estaciones estáticas, por lo que no es posible comunicarse entre redes creadas por diferentes puntos de acceso [1-3].

La estación está identificada tanto por una dirección MAC como por una dirección IP, esta última generalmente es asignada por el punto de acceso [4]. Para conectarse a un punto de acceso se debe estar dentro del área de cobertura, conocer su SSID (*Service Set IDentifier*) y contraseña [1].

La biblioteca ESP permite realizar las siguientes acciones cuando el módulo funciona como una estación:

- Conectarse a un punto de acceso
- Obtener o establecer la dirección IP asignada al módulo como estación.
- Obtener o establecer la dirección MAC del módulo como estación.
- Desconectarse de un punto de acceso.

Objetivo

Mostrar como conectarse a un punto de acceso desde una interfaz y reconocer su utilidad.

Firmware de comandos AT

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST

- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

Se utilizará el módulo WiFi como estación. Al iniciar la ejecución del programa del microcontrolador PIC, en la terminal se observará una lista con los nombres de los puntos de acceso cercanos al módulo. Se deberán seguir las instrucciones que se indiquen en la terminal para conectarse a alguno de los puntos de acceso mostrados previamente. Posteriormente, se deberá reclamar una dirección IP personalizada, e imprimir la dirección IP asignada para corroborar el cambio de la dirección IP.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P1-C.1 en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P1-C.2

```
#include <16lf1939.h>
#include "esp.h"

int err=0;
char IP[16]={0};
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión a un AP a través de una interfaz.
    err=scan_ap_station("ESP");
    if(err==0)
    {
        //Se reclama una dirección IP
        set_ip_station("192.168.1.70");
        //Obtiene la dirección IP asignada al módulo Wi-Fi
        get_ip_station(IP,sizeof(IP));
        fprintf(PIC,"Conexión exitosa\r\n");
        fprintf(PIC,"La dirección IP asignada es:%s\r\n",IP);
    }
    else
        fprintf(PIC,"No fue posible conectarse al AP\r\n");
    while(1)
    {
    }
}
```

Figura P1-C.1 *Código de la práctica 1-C*.

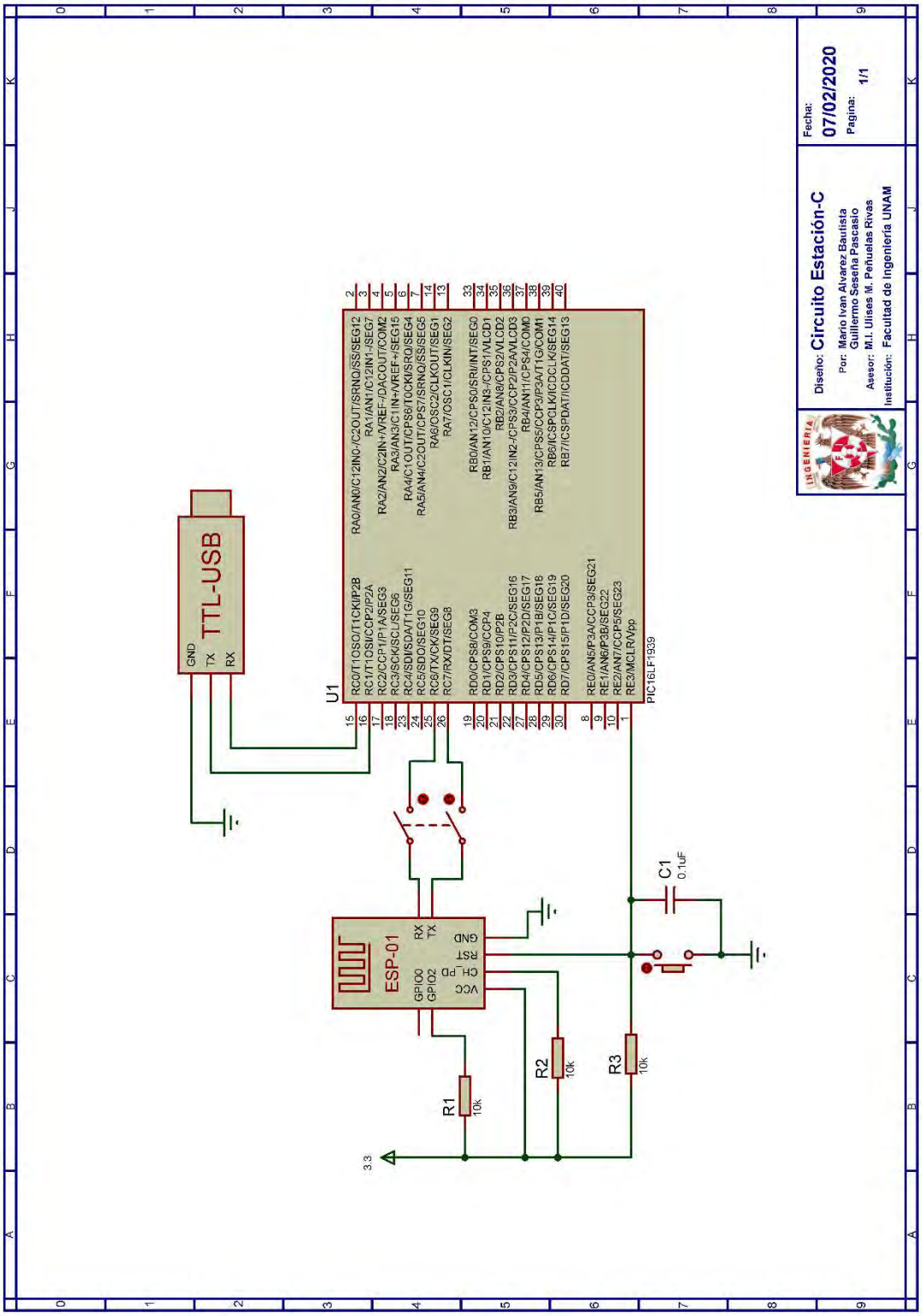


Figura P1-C.2 *Circuito de la práctica 1-C.*

Resultados

La ubicación de la herramienta que permite la creación de un punto de acceso en un teléfono inteligente varía en cada fabricante, sin embargo, ésta suele ser llamada “punto de acceso portátil” y generalmente está ubicada dentro de redes inalámbricas en configuración.

Al iniciar la ejecución del programa del microcontrolador PIC, el primer mensaje que se mostrará en la terminal será una lista de nombres de puntos de acceso cercanos al módulo WiFi (ver figura P1-C.3).

```
Moodulo esp listo para su uso\r\n
("infinitum movil")\r\n
("PROYECTOS_CIA")\r\n
("NodosMCU")\r\n
("INVITADOS_CIA")\r\n
("IMPRESORAS_CIA")\r\n
("Roootech")\r\n
("Mechatronics Research Group")\r\n
("MECANICA_CIA")\r\n
("INDUSTRIAL_CIA")\r\n
("ESTUDIANTES_CIA")\r\n
("MECATRONICA_CIA")\r\n
("ENCUESTAS_CIA")\r\n
("P1_C")\r\n
("ESTUDIANTES_CIA")\r\n
("MECATRONICA_CIA")\r\n
("ENCUESTAS_CIA")\r\n
("BIOMEDICOS_CIA")\r\n
("PROYECTOS_CIA")\r\n
```

Figura P1-C.3 *Puntos de acceso cercanos.*

El siguiente mensaje que se mostrará será la petición al usuario para que ingrese el SSID del punto de acceso y una vez ingresado éste, se mostrará un mensaje pidiendo al usuario que ingrese la contraseña de dicho punto de acceso (ver figura P1-C.4).

```
\r\nIngresa ssid\r\n
Ingresa password\r\n
```

Figura P1-C.4 Solicitud de ingreso de parámetros del AP.

El siguiente mensaje indicará si la conexión fue exitosa o no y en caso de ser exitosa, incluirá la dirección IP asignada al módulo WiFi que deberá ser la que se ha reclamado (ver figura P1-C.5).

```
Conexioon exitosa\r\n
La direccion IP asignada es:192.168.1.70
```

Figura P1-C.5 Dirección IP asignada al módulo.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3dhJg11

Notas

- Se recomienda usar el programa Hterm para simular la terminal.
- Cuando se reclama una dirección IP, únicamente se puede personalizar la parte correspondiente a la identificación del host o el cambio no se realizará. Para más detalles sobre direcciones IP consultar [2].
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los

establecidos por defecto para el segundo serial por *software* del uC PIC.

- Esta forma de conexión permite la conexión a diferentes puntos de acceso sin tener que cambiar el código del uC PIC.

Referencias

1. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
2. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
3. Espressif Systems, *ESP8266 Non-OS SDK API Reference Version 3.0.1*. 2019.
4. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.

Práctica 1-D Estación: conexión a un punto de acceso con la mayor intensidad de señal

Introducción

Una estación es un dispositivo que cuenta con un adaptador WiFi que le permite conectarse a una red para proveer y/o utilizar servicios [1]. Los módulos WiFi que incorporan el SoC ESP8266 o los ESP32 pueden funcionar como estaciones y están diseñados para operar en modo infraestructura, es decir, que se deben conectar a un punto de acceso, el cual gestionará las conexiones y transmitirá todos los paquetes [1-3]. Al mismo tiempo, éstos están configurados como estaciones estáticas, por lo que no es posible comunicarse entre redes creadas por diferentes puntos de acceso [1-3].

La estación está identificada tanto por una dirección MAC como por una dirección IP, esta última generalmente es asignada por el punto de acceso [4]. Para conectarse a un punto de acceso se debe estar dentro del área de cobertura, conocer su SSID (*Service Set IDentifier*) y contraseña [1].

La biblioteca ESP permite realizar las siguientes acciones cuando el módulo funciona como una estación:

- Conectarse a un punto de acceso, en este caso estableciendo hasta tres puntos de acceso con sus respectivas contraseñas y se conectará al de mayor intensidad de señal.
- Obtener o establecer la dirección IP asignada al módulo como estación.
- Obtener o establecer la dirección MAC del módulo como estación.
- Desconectarse de un punto de acceso.
- Definir un GPIO del módulo para estatus LED

Objetivo

Ilustrar el proceso para conectarse a un punto de acceso que presente la mayor intensidad de señal.

Firmware de comandos AT

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION

Materiales

- 2 teléfonos inteligentes
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

Se utilizarán dos teléfonos inteligentes para crear en cada uno un punto de acceso, llamados SSID1 y SSID2. Un teléfono se colocará a 30 cm del módulo WiFi, mientras que el otro se colocará a dos metros de éste. Finalmente se verificará en los teléfonos a que punto de acceso se ha conectado el módulo WiFi.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P1-D.1, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P1-D.2.

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso con la mayor intensidad de señal, para
    // lo cual se especifican los SSID y contraseñas de los APs
    begin_strongestap_station("ESP01","SSID1","123456789","SSID2","012345678");
    while(1)
    {
    }
}
```

Figura P1-D.1 *Código de la práctica 1-D.*

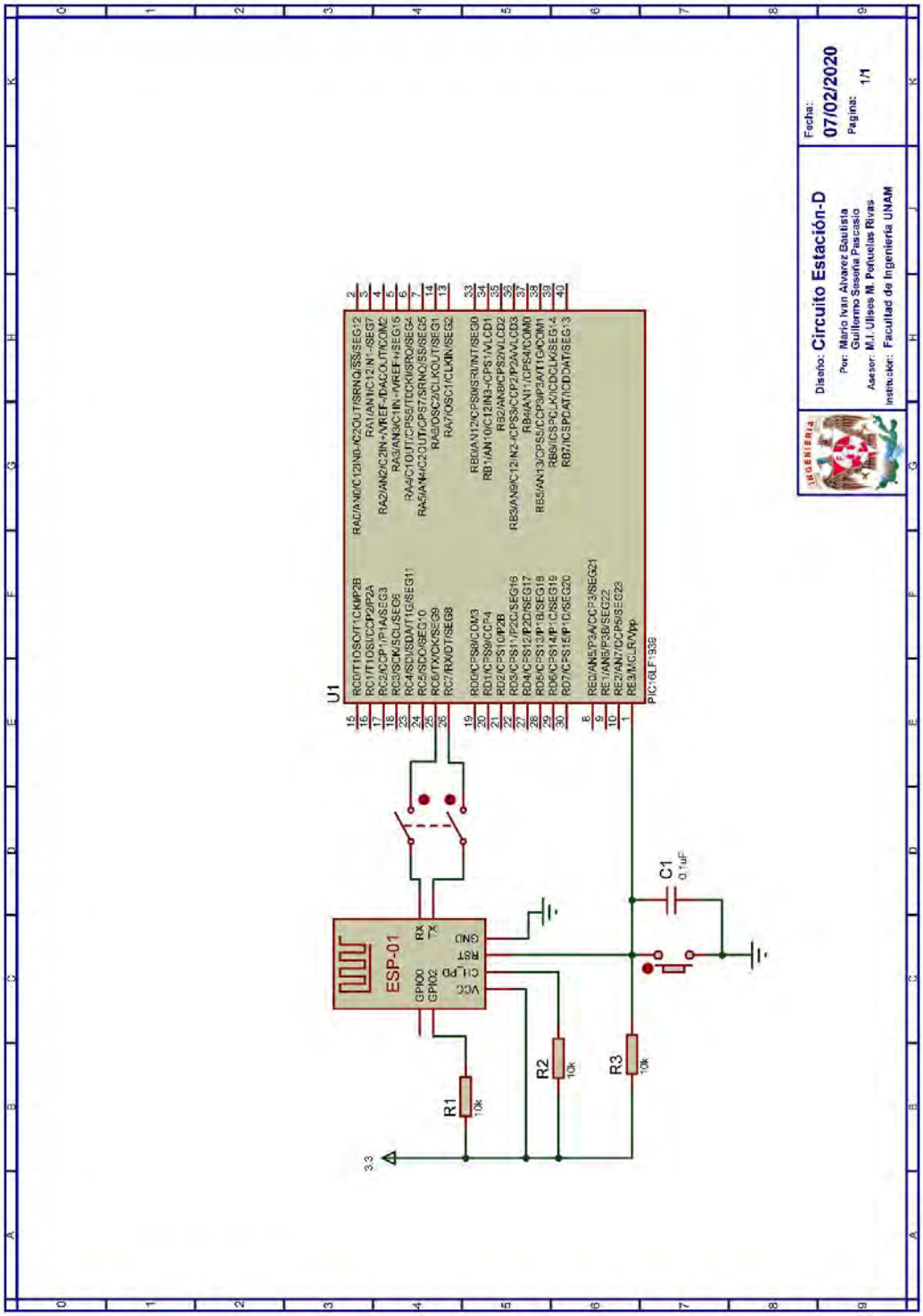


Figura P1-D.2 *Circuito de la práctica 1-D.*

Resultados

La creación de un punto de acceso en un teléfono inteligente varía en cada fabricante, sin embargo, ésta suele ser llamada “punto de acceso portátil” y generalmente está ubicada dentro de redes inalámbricas en configuración.

Previo a la ejecución del código se debe verificar la creación de los puntos de acceso de los teléfonos inteligentes SSID1 y SSID 2 (ver figura P1-D.3).

Figura P1-D.3 *Redes WiFi creadas para la práctica*.

Tras la ejecución del código del microcontrolador, el módulo se conectará al punto de acceso más cercano, puesto que será éste el que presente la mayor intensidad de señal. En el teléfono inteligente que haya creado el punto de acceso al cual se conectó el módulo, se indicará que está conectado un dispositivo, mientras que en el otro teléfono inteligente no se tendrá registro de ningún dispositivo conectado.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/37HHbdk

Notas

- La intensidad de señal de un punto de acceso puede verse afectada por diversos factores como lo puede ser la presencia de otras redes WiFi, objetos entre la estación y el emisor, el tipo de antena del punto de acceso, entre otros.

Referencias

1. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente*. Quinta ed. 2010, España: Pearson eduaction.
2. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
3. Espressif Systems, *ESP8266 Non-OS SDK API Reference Version 3.0.1*. 2019.
4. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.

Práctica 2 Smart config

Introducción

La tecnología denominada *Smart Config* que está presente en los módulos que incorporan el SoC ESP8266 o los ESP32, utiliza el protocolo ESP-TOUCH desarrollado por Espressif Systems para configurar estos módulos de manera remota para conectarse a un punto de acceso. Este proceso está ideado para sistemas que no cuentan con una interfaz y es realizado de manera simple a través de una aplicación en un teléfono inteligente (ver figura P2.1) [1, 2].

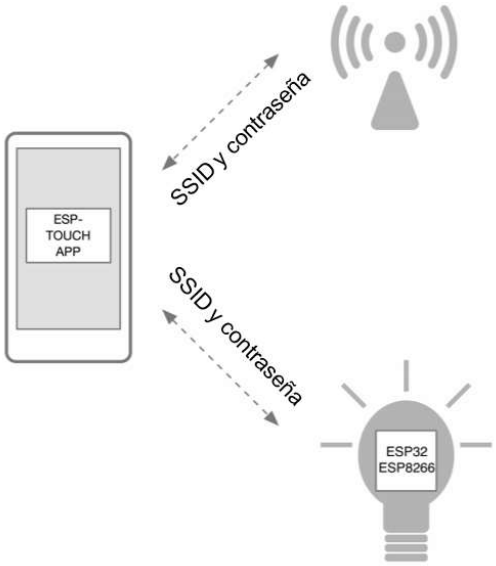


Figura P2.1 *Esquema ESP-TOUCH[1].*

Para esto, el teléfono inteligente enviará una serie de paquetes UDP (ver figura P2.2) con el SSID y la contraseña codificados hacia el punto de acceso, el módulo WiFi que incorpore alguno de los SoC antes mencionados, podrá alcanzar estos paquetes y por tanto, obtener la información que requiere para conectarse al punto de acceso. El proceso para la utilización de *Smart Config* se describe a continuación:

1. Habilitar el *Smart Config* en el módulo WiFi mediante la biblioteca ESP.
2. Conectar el teléfono inteligente al punto de acceso.
3. Abrir la aplicación ESP-TOUCH instalada en el teléfono inteligente.
4. Ingresar la aplicación el SSID y la contraseña, este último en caso de la conexión este cifrada, para conectarse al punto de acceso.

El tiempo que tardará el módulo WiFi en conectarse al punto de acceso dependerá de la distancia a la que se encuentren todos los elementos involucrados. Si el proceso de conexión falló, la aplicación mostrará un mensaje alertando de la falla. En cambio, si la conexión resultó exitosa la aplicación mostrará un mensaje notificando del éxito de la conexión junto con la dirección IP que fue asignada al módulo WiFi [1].

La aplicación que se ejecuta en el teléfono inteligente es ESP-TOUCH y está disponible tanto para Android como para iOS, los códigos de éstas pueden ser descargados en [2] dentro de

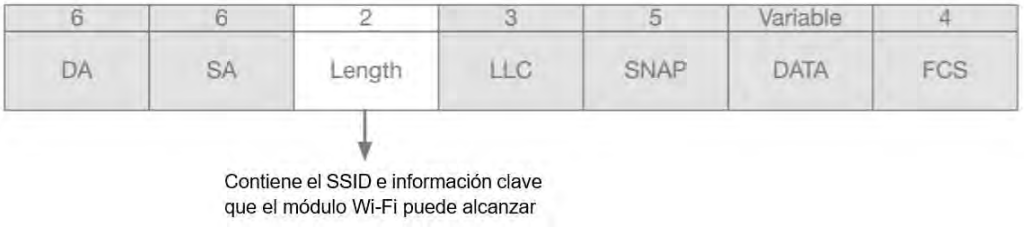


Figura P2.2 *Estructura del paquete de datos[1].*

Resources en APKS, permitiendo la personalización de dichas aplicaciones.

Con *Smart Config* la biblioteca ESP permite que el módulo sea conectado a cualquier punto de acceso sin tener que cambiar el código del uC PIC.

Objetivo

Mostrar como utilizar *Smart Config* y reconocer su utilidad.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

STATION y E_SMART_CONFIG.

Materiales

- 1 adaptador USB-TTL
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 mA
- Aplicación ESP-TOUCH en un teléfono inteligente

Descripción

Al ejecutarse el código del microcontrolador PIC, en la terminal aparecerá el mensaje “ejecute aplicación” para indicar el momento en que se ha activado *Smart Config* en el módulo WiFi. Tras este mensaje se procederá a conectar el módulo WiFi a través de la aplicación ESP-TOUCH que es proporcionada por Espressif Systems.

```
#include <16lf1939.h>
#include "esp.h"

int  conn=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    fprintf(PIC,"Inicie aplicación\r\n");
    //Habilita Smart config
    conn=run_smart_station("ESP");
    if(conn==0)
    {
        //Enciende el LED si se conectó al AP
        output_bit(PIN_A5,TRUE);
    }
    while(1)
    {
    }
}
```

Figura P2.3 *Código de la práctica 2.*

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P2.3.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P2.6.

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC se deberá esperar hasta que aparezca el mensaje “Inicie aplicación”, momento en el que se deberá conectar el teléfono inteligente al punto de acceso al cual se desea conectar el módulo WiFi. Posteriormente, se deberá abrir la aplicación ESP-TOUCH e ingresar la contraseña del punto de acceso, el número de los dispositivos a los que se les compartirá la información y seleccionar *broadcast*, tal y como se muestra en la figura P2.5.

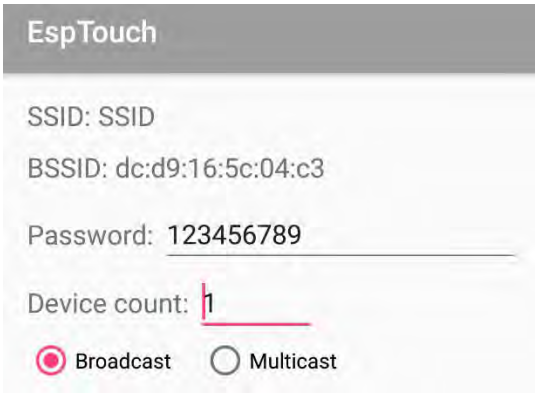


Figura P2.5 *Establecimiento de parámetros en ESP-TOUCH*

Después, se deberá presionar el botón *CONFIRM* para iniciar el proceso de conexión del módulo WiFi. Si la conexión fue exitosa, en la aplicación se observará el mensaje mostrado en la figura P2.4

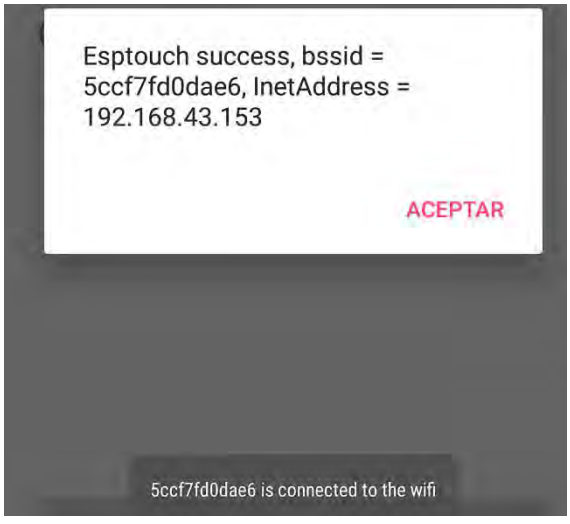


Figura P2.4 *Mensaje de conexión exitosa.*

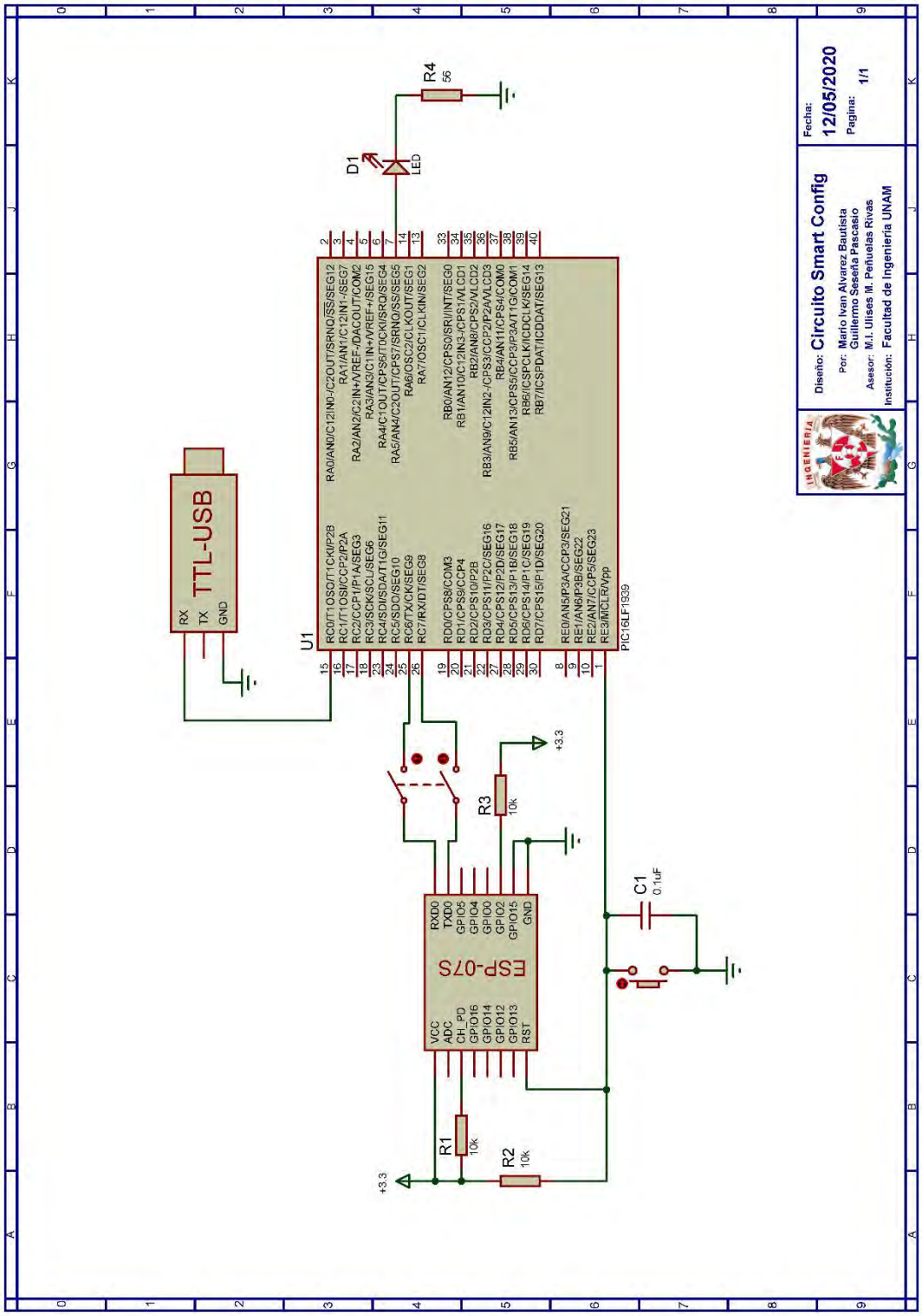


Figura P2.6 *Circuito de la práctica 2.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/30ZzZIc

Notas

- Se puede reemplazar el uso de la terminal y el adaptador USB-TLL por un LED, para indicar el momento en que se ha activado *Smart Config* en el módulo WiFi y se deba habilitar la aplicación.

Referencias

1. Espressif Systems, *ESP-TOUCH User Guide version 2.0*. 2018.
2. Espressif Systems. *ESP-Touch*. [Citado 2020 Marzo]; Disponible en:https://www.espressif.com/en/products/software/esp-touch/resources.

Práctica 3 Servidor TCP

Introducción

Cualquier dispositivo conectado a internet requiere de un identificador único para comunicarse con otros dispositivos, dicho identificador se llama dirección IP, que comúnmente es representada con valores decimales divididos en octetos por puntos. La limitada cantidad de direcciones IP disponibles en internet no es lo suficientemente numerosa para satisfacer la enorme cantidad de dispositivos conectados, por lo que los Proveedores de Servicios de internet (*nternet Service Provider, IPS*), proporcionan subredes que mayormente son redes LAN (*Local Area Network*), y a cada una de éstas le asigna una dirección IP para tráfico en internet, que es conocida como IP publica, mientras que, los dispositivos conectados a la subred se les otorga una dirección IP única (dentro de la red) nombrada como IP privada. Para que los dispositivos locales puedan comunicarse a internet es necesario que la información pase de direcciones IP privadas a la IP publica compartida y viceversa, para dicha tarea se encarga la Traducción de Dirección de Red (NAT-*Network Address Translation*) [1].

Cada dispositivo de la subred se comunica con otros usando puertos, los cuales son extensiones de las direcciones IP, permitiendo que un solo dispositivo pueda disponer de diferentes aplicaciones a la vez, estos puertos se representan como valores enteros de 16 bits. Cuando un dispositivo de la subred envíe información lo realiza enviando uno o varios paquetes, en cada paquete se especifica el puerto de origen y el de destino. Si el destinatario del paquete se encuentra en la misma red, el enrutador lo retransmite directamente, de no ser así, retransmite al enrutador del IPS, pero antes tiene que pasar por la caja NAT (ver figura P3.1), en donde el paquete es modificado para que sea retransmitido usando la IP pública compartida, además el puerto origen es sustituido por un valor (puerto público)

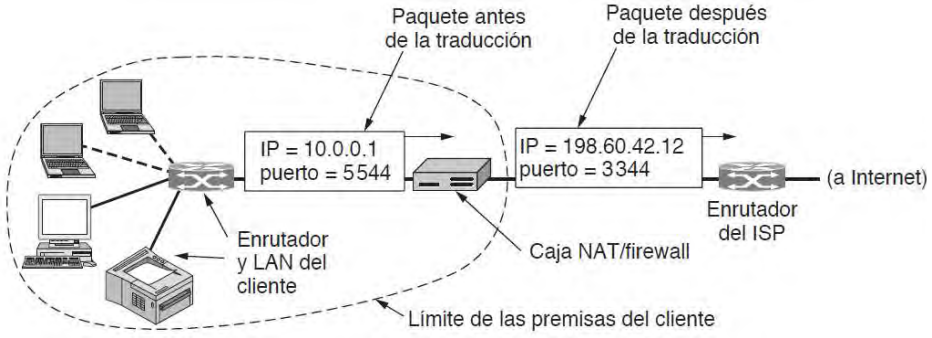


Figura P3.1 *Colocación y funcionamiento de una caja NAT [1].*

disponible de la tabla de traducción de entradas. En dicha tabla se almacena la dirección IP privada y puerto de origen, de tal modo que, si el dispositivo destinatario envía una respuesta, el o los paquetes entrantes especificaran el puerto de destino público, el cual es utilizado como índice en la tabla de traducción, y así el paquete entrante es retransmitido a la IP privada y puerto destino correspondiente.

La NAT es útil para que un dispositivo local pueda enviar información a uno remoto, sin embargo, un remoto no puede comunicarse con él de la subred hasta que el dispositivo local haya comenzado la interacción, esta es una gran debilidad de la NAT. No obstante, algunas cajas NAT permiten asignar un puerto (índice) en el que los paquetes entrantes a la IP pública compartida sean retransmitidos a una dirección IP privada con un puerto específico (de un dispositivo local). De esta forma, el dispositivo remoto puede comunicarse con el local mediante paquetes dirigidos a la IP publica compartida con puerto de destino igual al índice establecido en la caja NAT.

En internet existen dos principales formas en que los paquetes son enviados, una es mediante el protocolo TCP (Protocolo de control de transmisión o *Transmission Control Protocol*) y la otra es a través del protocolo UDP (Protocolo de datagramas de usuario o *User Datagram Protocol*). Ambos son protocolos de transporte que sirven para enviar información entre aplicaciones sin que éstas tengan que lidiar con la parte física de la transmisión. El protocolo TCP es el más utilizado, ya que proporciona un servicio confiable, en que la información enviada es secciona en pequeños paquetes, los cuales son reconstruidos de forma ordenada por el destinatario, garantizando así, la integridad de la información que se transmite [1].

Otra característica importante de TCP es que ofrece un servicio basado en la conexión, para comprender esto es necesario mencionar que en internet existe un modelo ampliamente utilizado llamado cliente-servidor, modelo en el que los dispositivos desempeñan un rol específico para intercambiar información.

Los servidores son dispositivos (computadoras) con gran capacidad computacional en los que se aloja información, mientras que los clientes son dispositivos más simples, que acceden a la información que disponen los servidores. Aunque el intercambio de información entre cliente y servidor puede ser bidireccional, usualmente se efectúa del servidor hacia el cliente a través de respuestas. Los clientes solicitan a un servidor, información o efectuar alguna acción, al recibir dicha solicitud el servidor genera una respuesta para el cliente. El intercambio

de información entre cliente y servidor sólo es posible hasta que ambos hayan establecido una conexión, en la cual hay un previo intercambio de paquetes, en el que se puede realizar una negociación de parámetros de comunicación o simplemente es para asegurar que ambos dispositivos están dispuestos a intercambiar información. La conexión siempre es inicializada por los clientes, mientras que los servidores están a la espera de atender dichas conexiones [1].

Los módulos ESP, pueden ser establecidos como servidores usando el protocolo TCP, evidentemente no se pueden equiparar con servidores de uso profesional, aun así, ofrece características muy interesantes. Aquellas disponibles en el *firmware* de comandos son:

- Crear un servidor TCP para un puerto seleccionado, cuando el módulo funcione como estación o punto de acceso.
- Especificar el tiempo de espera máximo (*timeout*) en que un cliente puede estar inactivo antes de ser desconectado por el servidor.
- Soportar varias conexiones simultaneas. Desde la perspectiva del módulo cada una de las conexiones es asignada a un canal de comunicación. Cada que un cliente se conecte se le asigna un canal de comunicación disponible, canal a través del cual pueden intercambiar información. Hasta que ocurra la desconexión el canal será liberado para una nueva conexión.
- Recepción de mensajes provenientes de clientes TCP, los cuales son extraídos directamente de la capa de transporte sin ningún formato.
- Envío de mensajes a los clientes conectados, especificando el canal en el cual se envía la información.
- Cerrar canales de comunicación, que permite desconectar clientes.
- Eliminar el servidor creado.
- Con las modificaciones del *firmware*, se puede establecer dos servidores TCP o mejor dicho habilitar dos puertos TCP, para cada uno de ellos se dispone de un máximo de dos conexiones simultaneas en la versión SDK NONOS, mientras que en la versión ESP-IDF el segundo servidor sólo permite una conexión.

Estas características han sido aprovechadas por la biblioteca ESP, permitiendo al usuario utilizar al módulo como un servidor TCP, para intercambiar información con uno o varios clientes. Los mensajes son extraídos sin ningún formato, dejando que el usuario

establezca las reglas o algún protocolo de aplicación si así lo requiere. La biblioteca ESP permite realizar las siguientes acciones:

- Establecer un servidor TCP, especificando el número máximo de conexiones simultaneas y el *timeout*.
- Habilitar uno o dos servidores (2 conexiones máximas para cada puerto).
- Establecer un *buffer* para cada puerto utilizado (tomar en cuenta que los paquetes recibidos son guardados como *strings* por ello al mensaje recibido se le inserta el terminador \0, si se requiere recibir información binaria el usuario tiene que establecer una forma para reconocer en donde termina cada mensaje recibido, por ejemplo, establecer que los dos primeros bytes indiquen la longitud del paquete recibido)
- Comparar los *buffers* con algún *string*
- Enviar *strings* alojados en la ROM del PIC (máximo 2048 bytes)
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 2048 bytes)
- Conocer que canal está siendo utilizado por un cliente.
- Cerrar la conexión de algún cliente.
- Eliminar uno o los dos servidores.

Objetivo

Conocer las características disponibles para el módulo cuando es establecido como servidor TCP.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_FIRST_SERVER para todas las actividades a excepción de la tercera.
- ACCESS_POINT y E_FIRST_SERVER para la tercera actividad.

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ

- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF
- 2 LED de 2.2V y 20 *mA*
- 2 resistores de 56 Ω

Descripción

Esta práctica está dividida en cuatro actividades:

- Acceso local
- Acceso remoto
- Como punto de acceso
- Segundo servidor

I Acceso local

El módulo será establecido como un servidor TCP en el puerto 8080, permitiendo un máximo de 4 conexiones a la vez. Dicho servidor ofrecerá el servicio de control de un LED e indicará el estado actual de dicho LED mediante la pulsación de un botón.

Para esta primera parte el módulo funcionará como una estación y se accederá al servidor montado en él desde un cliente local, para ello se conectará una computadora a la misma red que a la que está conectado el módulo, y se crea un cliente en el programa *packet sender* especificando la dirección IP privada (local) que le fue asignada al módulo y que es mostrada en una terminal.

Cuando un cliente establezca conexión con el servidor, recibirá el mensaje "Hola cliente TCP". Para controlar el estado del LED, el cliente tendrá que enviar los mensajes "ON" y "OFF" para el encendido y apagado respectivamente, cada que se envíe cualquiera de estos mensajes, el servidor responderá indicando que la acción se ha realizado correctamente. No obstante, el servidor podrá enviar información sin que el cliente lo solicite al pulsar el botón asociado al microcontrolador del servidor, acción que indicará el estado actual del LED a todos los clientes conectados en ese momento.

Además de responder con *strings* el servidor podrá enviar información almacenada en arreglos de bytes, por lo que cuando el cliente envíe “Bytes” obtendrá una respuesta con 3 bytes.

Los clientes podrán desconectarse cerrando la ventana de la conexión en *packet sender* o enviando el mensaje "Cerrar", y podrán eliminar el servidor al enviar el mensaje "Eliminar".

II Acceso remoto

En la segunda parte se realizará exactamente las mismas acciones que en la anterior actividad, la diferencia radica en que se accederá desde un cliente remoto, para esto el módulo se tiene que conectar a un punto de acceso conectado a internet. Además, para la subred a la que está conectado el módulo se le debe configurar la caja NAT para

acceder remotamente al módulo. Lo más común es que los IPS proporcionen dispositivos llamados módems que sirven para modular y desmodular la señal proveniente del IPS, que también funcionan como enrutadores. Estos dispositivos cuentan con documentación que especifica cómo acceder a su configuración, generalmente se realiza accediendo a la dirección IP privada del módem en un navegador web desde un dispositivo conectado a la misma red, en donde se solicitan credenciales para acceder a la configuración.

Dentro de la configuración se deberá buscar el apartado de la NAT y acceder a la opción de mapeo de puertos o apertura de puertos, esto dependerá específicamente del modelo de modem, por lo que es necesario consultarlo con el ISP que se usa. Aquí se muestra un ejemplo para un modem Arcadyan VRV8019AW22, en que se accede a la opción de mapeo de puertos, que como se muestra en la figura P3.2 se habilita una aplicación de tipo acceso remoto, ingresando la IP privada asignada al módulo junto con el puerto LAN que corresponde al puerto del servidor creado y finalmente indicando el puerto remoto que para este ejemplo se estableció en el puerto 8080, a cual los clientes remotos podrán acceder al módulo usando la IP publica compartida. Para este modelo también es necesario habilitar la opción de zona desmilitarizada tal como se muestra en la figura P3.3.

III Como punto de acceso

En esta tercera parte se repetirá las mismas acciones que en la primera actividad, pero con la diferencia de que el módulo funcionará como un punto de acceso abierto. Esta actividad está pensada para situaciones en las que no se pueda acceder a un punto de acceso

Figura P3.2 *Configuración de mapeo de puertos para el módulo ESP.*

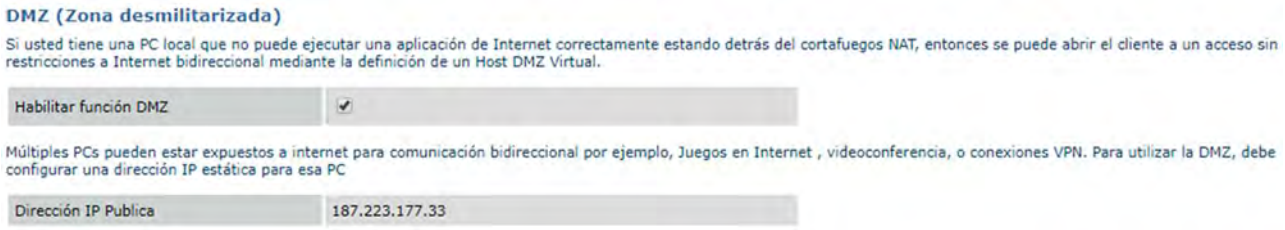


Figura P3.3 *Habilitación de zona desmilitarizada.*

WiFi. Los clientes que se deseen conectar al servidor deberán ser creados desde una computadora o dispositivo conectado a la red que se crea en el módulo ESP.

IV Segundo servidor

Finalmente, esta última actividad demostrará el funcionamiento del segundo servidor (segundo puerto TCP), para lo cual se tendrá el mismo funcionamiento de las actividades anteriores, pero ahora se controlan dos LED. El primero será controlado por el primer puerto (8080), mientras que el segundo se controlará en el segundo puerto (8081). Para crear clientes en el segundo puerto se realizará de la misma manera que para el primero, solo que se especificará el puerto 8081. Los clientes del segundo puerto también podrán desconectarse al enviar "Cerrar" y eliminar el segundo servidor enviando el mensaje "Eliminar" al puerto correcto.

Código

El código que deberá ejecutar el microcontrolador PIC para las primeras dos actividades es el mostrado en la figura P3.9 y la figura P3.10, mientras que, para la tercera actividad se sustituye la línea:

```
begin_station("Estacion","SSID",
"password");
```

por:

```
begin_ap(0, "SSID");
```

Finalmente, para la cuarta parte se debe ejecutar el código mostrado en la figura P3.11, figura P3.12 y figura P3.13, que como se puede apreciar se agregó código para el segundo puerto TCP, que tiene el mismo funcionamiento que el primero, sin embargo, cambia la sintaxis del código.

Para todas las actividades a excepción de la tercera, se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P3.14.

Resultados

Independientemente de la actividad, al ejecutarse el código del microcontrolador, lo primero que se observará en la terminal será la dirección IP privada que le fue asignada al módulo, ya sea como estación o como punto de acceso.

La computadora deberá conectarse a la misma red a la que el módulo está conectado, a excepción de la segunda actividad en que se accede remotamente desde una red diferente. Utilizando el programa *packet sender* (consultar https://packetsender.com/documentation) se creará uno o varios clientes especificando la dirección IP privada y el puerto 8080 o el puerto 8081 para la tercera actividad, con excepción en la segunda actividad, en la que se especifica la IP publica compartida asignada a la red junto con el puerto público.

Los clientes podrán encender o apagar un LED, cada vez que realice cualquiera de estas opciones obtendrán una respuesta del servidor tal y como se muestra en la figura P3.5. Al presionar el botón se informará a los clientes conectados, sobre el estado actual del LED (ver figura P3.4).

Además, los clientes podrán solicitar información binaria (arreglo bytes), enviando el mensaje “Bytes”, como la respuesta contiene caracteres no imprimibles, éstos no se podrán visualizar en la ventana de la conexión, sino que se mostrarán en la ventana principal del programa tal como se muestra en la figura P3.6.

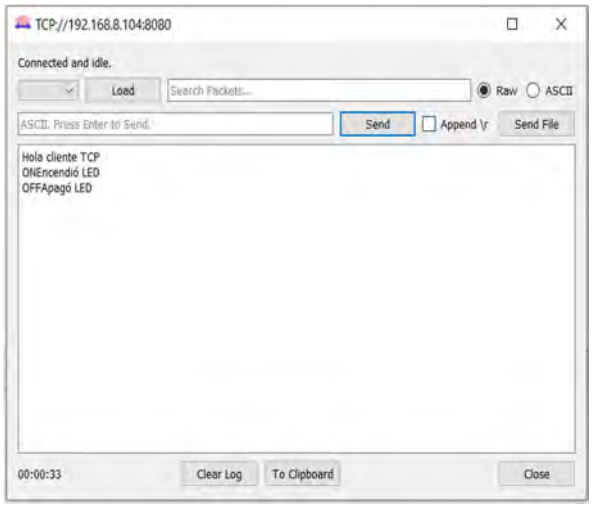


Figura P3.5 *Cliente TCP controlando estado de un LED.*

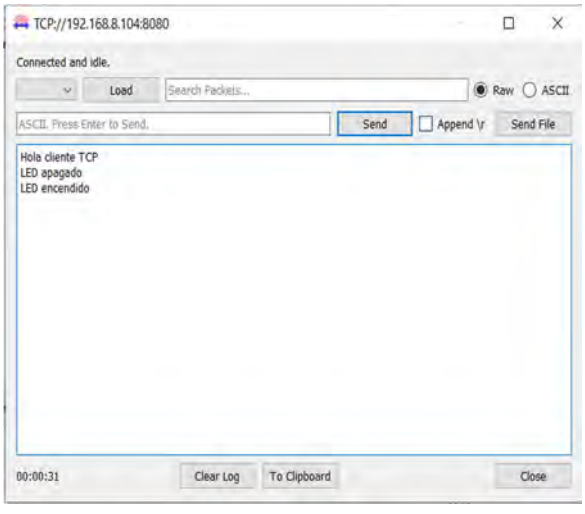


Figura P3.4 *Cliente TCP recibiendo el estado de un LED.*

	Time	From IP	From Port	To IP	To Port	Method	Error	ASCII	
1	19:18:35.014	192.168.8.104	8080	You	62226	TCP		\00\01\02	00 01 02
2	19:18:34.678	You	62226	192....	8080	TCP		Bytes	42 79 74 65 73

Figura P3.6 *Cliente TCP recibiendo información en bytes.*

Los clientes podrán desconectarse al enviar “Cerrar” (ver figura P3.7) dejando libre el canal de comunicación para una nueva conexión. También los clientes pueden eliminar el puerto al que se conectan al enviar “Eliminar” (ver figura P3.8), al eliminarlo ya no se podrá crear clientes para dicho puerto.

En el siguiente enlace está la carpeta que contiene los códigos y el video del funcionamiento de la práctica:

https://bit.ly/30YF1oz

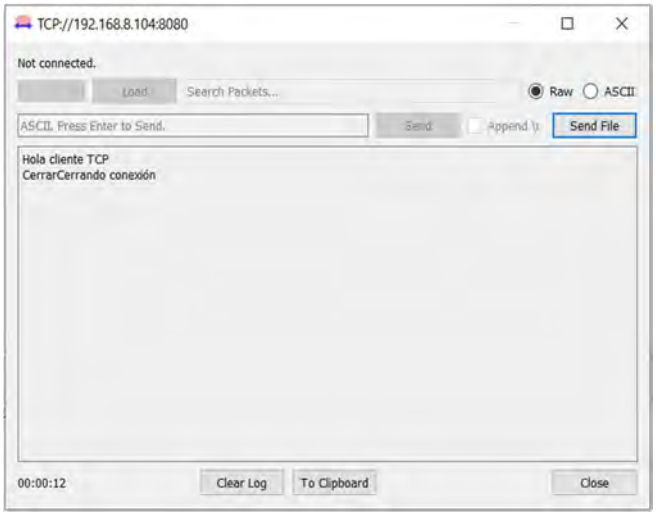


Figura P3.7 Cliente TCP desconectándose.

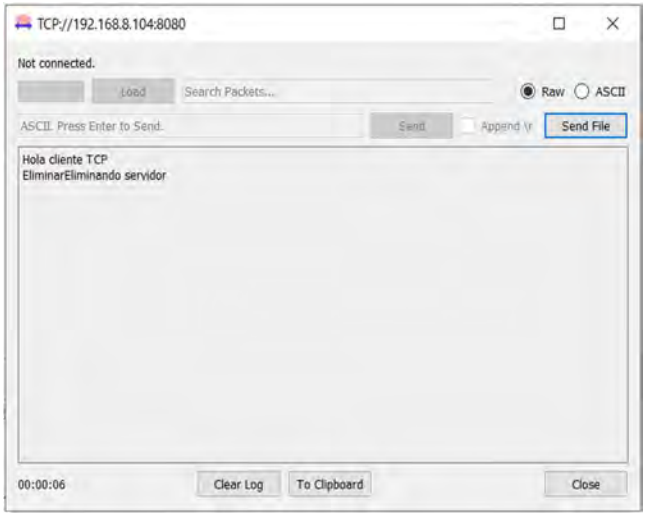


Figura P3.8 Cliente TCP eliminando el servidor.

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

int buffer[21]={0};
int banderas_saludo=0;
const char saludo[]="Hola cliente TCP\r\n";
int bytes[]={0x00, 0x01, 0x02};

void main()
{
    /*Establece al PIN B7 como salida y al resto del puerto B
    como entradas*/
    set_tris_b(0b01111111);
    //Establece en cero todo el puerto B
    output_b(0);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtención de la dirección IP de la estación
    get_ip_station(buffer,sizeof(buffer));
    //Imprime en la terminal la dirección IP
    fprintf(PIC,"Dirección IP:%s\r\n",buffer);
    /*Establece buffer para recibir mensajes provenientes
    de clientes TCP. En la posición 20 se almacena el
    canal de donde proviene el mensaje*/
    set_buffer_server1(buffer,sizeof(buffer));
    /*Crea servidor TCP estableciendo el puerto 8080,
    permitiendo un máximo de 4 conexiones a la vez y un
    timeout de 4000 s*/
    create_server1(8080,4,4000);
    while(1)
    {
        //Envía saludo a los clientes que se conecten
        for(int i=0; i<4; i++)
        {
            if(bit_test(banderas_saludo,i)==0 && client_connected(i)==1)
            {
                bit_set(banderas_saludo,i);
                send_server1_const(i,saludo);
            }
            else if(client_connected(i)==0)
            {
                bit_clear(banderas_saludo,i);
            }
        }

        //Enciende LED si se recibe "ON"
        if(compare_S1buffer("ON",1))
        {
            output_bit(PIN_B7,1);
            send_server1(buffer[20],"Encendió LED\r\n");
        }
```

Figura P3.9 *Primera parte del código de la práctica 3 para la primera y segunda actividad.*

```
//Apaga LED si se recibe "OFF"
    else if(compare_S1buffer("OFF",1))
    {
        output_bit(PIN_B7,0);
        send_server1(buffer[20],"Apagó LED\r\n");
    }
    //Al presionar botón para indicar estado actual del LED
    if(1==input_state(PIN_B6))
    {
        if(input_state(PIN_B7))
        {
            /*Indica solo a los clientes activos que el LED
            está encendido*/
            send_server1(0,"LED encendido\r\n");
            send_server1(1,"LED encendido\r\n");
            send_server1(2,"LED encendido\r\n");
            send_server1(3,"LED encendido\r\n");
        }
        else
        {
            /*Indica solo a los clientes activos que el LED
            está apagado*/
            send_server1(0,"LED apagado\r\n");
            send_server1(1,"LED apagado\r\n");
            send_server1(2,"LED apagado\r\n");
            send_server1(3,"LED apagado\r\n");
        }
         while(1==input_state(PIN_B6))
        {
            //Se mantiene hasta soltar el botón
        }
    }
    //Envía un arreglo de enteros de 8 bits (bytes)
    if(compare_S1buffer("Bytes",1))
    {
        send_bytes_server1(buffer[20],bytes,sizeof(bytes));
    }
    //Cierra la conexión al cliente que envía "Cerrar"
    if(compare_S1buffer("Cerrar",1))
    {
        send_server1(buffer[20],"Cerrando conexión\r\n");
        close_client_server1(buffer[20]);
    }
    //Elimina el servidor si se recibe "Eliminar"
    if (compare_S1buffer("Eliminar",1))
    {
        send_server1(buffer[20],"Eliminando servidor\r\n");
        delete_server1();
    }
}

}
```

Figura P3.10 *Segunda parte del código de la práctica 3 para la primera y segunda actividad.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

int s1_buffer[21]={0};
int s2_buffer[21]={0};
int banderas_saludo=0;
const char saludo[]="Hola cliente TCP\r\n";
int bytes[]={0x00, 0x01, 0x02};

void main()
{
    /*Establece al PIN B7 y PIN B5 como salida y
    al resto del puerto B como entradas*/
    set_tris_b(0b01011111);
    //Establece en cero todo el puerto B
    output_b(0);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtención de la dirección IP de la estación
    get_ip_station(s1_buffer,sizeof(s1_buffer));
    //Imprime en la terminal la dirección IP
    fprintf(PIC,"Dirección IP:%s\r\n",s1_buffer);
    /*Establece buffer para recibir mensajes provenientes
    de clientes del primer servidor. En la posición 20 se
    almacena el canal de donde proviene el mensaje*/
    set_buffer_server1(s1_buffer,sizeof(s1_buffer));
    /*Establece buffer para recibir mensajes provenientes
    de clientes del segundo servidor. En la posición 20 se
    almacena el canal de donde proviene el mensaje*/
    set_buffer_server2(s2_buffer,sizeof(s2_buffer));
    /*Crea el primer servidor TCP estableciendo el puerto 8080,
    permitiendo un máximo de 2 conexiones a la vez y un
    timeout de 4000 s*/
    create_server1(8080,2,4000);
    /*Crea el segundo servidor TCP estableciendo el puerto 8081,
    permitiendo un máximo de 2 conexiones a la vez y un
    timeout de 4000 s*/
    create_server2(8081,2,4000);
```

Figura P3.11 *Primera parte del código de la práctica 3 para la cuarta actividad.*

```
while(1)
{
    //Envía saludo a los clientes que se conecten
    for(int i=0; i<2; i++)
    {
        if(bit_test(banderas_saludo,i)==0 && client_connected(i)==1)
        {
            bit_set(banderas_saludo,i);
            send_server1_const(i,saludo);
        }
        else if(client_connected(i)==0)
        {
            bit_clear(banderas_saludo,i);
        }
    }
    for(int j=2; j<4; j++)
    {
        if(bit_test(banderas_saludo,j)==0 && client_connected(j)==1)
        {
            bit_set(banderas_saludo,j);
            send_server2_const(j,saludo);
        }
        else if(client_connected(j)==0)
        {
            bit_clear(banderas_saludo,j);
        }
    }

    //Enciende LED si se recibe "ON" para el primer servidor
    if(compare_S1buffer("ON",1))
    {
        output_bit(PIN_B7,1);
        send_server1(s1_buffer[20],"Encendió LED\r\n");
    }
    //Apaga LED si se recibe "OFF" para el primer servidor
    else if(compare_S1buffer("OFF",1))
    {
        output_bit(PIN_B7,0);
        send_server1(s1_buffer[20],"Apagó LED\r\n");
    }

    //Enciende LED si se recibe "ON" para el segundo servidor
    if(compare_S2buffer("ON",1))
    {
        output_bit(PIN_B5,1);
        send_server2(s2_buffer[20],"Encendió LED\r\n");
    }
    //Apaga LED si se recibe "OFF" para el segundo servidor
    else if(compare_S2buffer("OFF",1))
    {
        output_bit(PIN_B5,0);
        send_server2(s2_buffer[20],"Apagó LED\r\n");
    }
    //Al presionar botón para indicar estado actual de los LEDs
    if(1==input_state(PIN_B6))
    {
        //Para el primer servidor
         if(input_state(PIN_B7))
        {
            send_server1(0,"LED encendido\r\n");
            send_server1(1,"LED encendido\r\n");

        }
```

Figura P3.12 *Segunda parte del código de la práctica 3 para la cuarta actividad.*

```
            else
            {
                send_server1(0,"LED apagado\r\n");
                send_server1(1,"LED apagado\r\n");

            }
            //Para el segundo servidor
            if(input_state(PIN_B5))
            {
                send_server2(2,"LED encendido\r\n");
                send_server2(3,"LED encendido\r\n");
            }
            else
            {
                send_server2(2,"LED apagado\r\n");
                send_server2(3,"LED apagado\r\n");
            }
             while(1==input_state(PIN_B6))
            {
                //Se mantiene hasta soltar el botón
            }
        }
        /*Envía un arreglo de enteros de 8 bits (bytes)
        para el primer servidor*/
        if(compare_S1buffer("Bytes",1))
        {
            send_bytes_server1(s1_buffer[20],bytes,sizeof(bytes));
        }
        /*Envía un arreglo de enteros de 8 bits (bytes)
        para el segundo servidor*/
        if(compare_S2buffer("Bytes",1))
        {
            send_bytes_server2(s2_buffer[20],bytes,sizeof(bytes));
        }
        /*Cierra la conexión al cliente que envía "Cerrar"
        para el primer servidor*/
        if(compare_S1buffer("Cerrar",1))
        {
            send_server1(s1_buffer[20],"Cerrando conexión\r\n");
            close_client_server1(s1_buffer[20]);
        }
        /*Elimina el servidor si se recibe "Eliminar"
        para el primer servidor*/
        if (compare_S1buffer("Eliminar",1))
        {
            delete_server1();
        }
        /*Cierra la conexión al cliente que envía "Cerrar"
        para el segundo servidor*/
        if(compare_S2buffer("Cerrar",1))
        {
            send_server2(s2_buffer[20],"Cerrando conexión\r\n");
            close_client_server2(s2_buffer[20]);
        }
        /*Elimina el servidor si se recibe "Eliminar"
        para el segundo servidor*/
        if (compare_S2buffer("Eliminar",1))
        {
            delete_server2();
        }
    }
}
```

Figura P3.13 *Tercera parte del código de la práctica 3 para la cuarta actividad.*

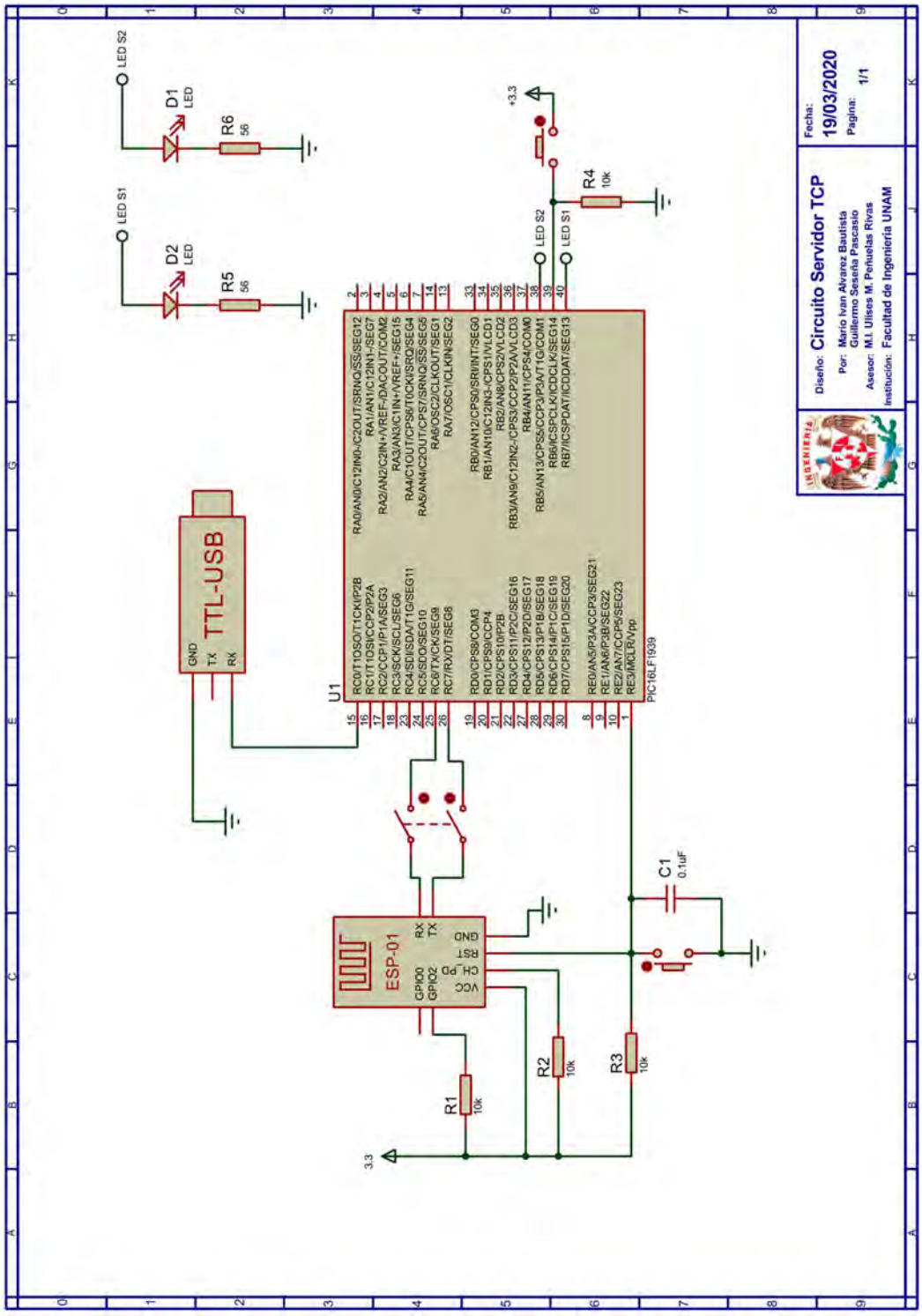


Figura P3.14 *Circuito de la práctica 3.*

Notas

- Se recomienda usar el programa Hterm para simular la terminal.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 4 Servidor SSL

Introducción

La necesidad por conexiones seguras surgió cuando las empresas requirieron intercambiar información confidencial para realizar acciones como transacciones financieras o pagos a través de la navegación web. Una de las soluciones para atender a dicha necesidad fue el protocolo SSL (*Secure Sockets Layer,* Capa de sockets seguros) desarrollado por Netscape Communications Corp. Posteriormente la versión 3.0 de este protocolo sería tomada como base para desarrollar un estándar de conexiones seguras llamado TLS (*Transport Layer Security,* Seguridad de la apa de transporte) [1]. TLS es un protocolo ampliamente utilizado en la navegación web mediante HTTPS (Protocolo seguro de transferencia de hipertexto, *Secure Hypertext Transfer Protocol)*, pero puede ser utilizado para cualquier aplicación que utilice TCP [1].

SSL/TLS trabajan encima de la capa de transporte utilizando el protocolo TCP y están diseñados para proporcionar tres servicios:

- Confidencialidad, todos los datos enviados son cifrados.
- Autenticación del servidor o de ambas partes mediante un certificado digital firmado por alguna autoridad de certificación. La autenticación puede ser opcional.
- Integridad del mensaje, todos los mensajes cifrados incluyen un resumen del mensaje para comprobar si han sido manipulados o falsificados[2-4].

Después que un cliente establece conexión con un servidor SSL/TLS, ambos llevan un proceso de negociación en que acuerdan los métodos de encriptación que se utilizarán para intercambiar información. Cuando se aceptan las condiciones de la comunicación, el servidor envía un certificado que el cliente tiene que verificar con la autoridad de certificación para conocer si el servidor es realmente quien dice ser. La autentificación no sólo es para el servidor, sino que también el servidor puede verificar la identidad del cliente. Después de la verificación del certificado, cliente y servidor intercambian las claves que cada uno utilizará para cifrar y descifrar la información que se intercambiará [1, 5].

El módulo ESP puede ser establecido como un servidor SSL/TSL que cuenta con un certificado autofirmado. Para las versiones IDF el servidor SSL/TLS es establecido habilitando la aplicación del servidor TCP (E_FIRST_SERVER), ajustando uno de los parámetros durante su creación para convertirlo en servidor SSL, mientras que para las versiones SDK, en las cuales no estaba incluido en

el *firmware* original, se ha agregado como una aplicación separada (E_SSL_SERVER).

El servidor SSL en IDF cuenta con las mismas características que el servidor TCP que fue mostrado en la práctica anterior. Para la versión SDK el módulo es establecido como un servidor SSL que permite realizar las siguientes acciones:

- Establecer un servidor SSL con un máximo de una conexión simultánea y con un *timeout* ajustable.
- Establecer un *buffer* de información proveniente del cliente (tomar en cuenta que los paquetes recibidos son guardados como *strings,* por ello al mensaje recibido se le inserta el terminador \0, si se requiere recibir información binaria el usuario tiene que establecer una forma para reconocer en donde termina cada mensaje recibido, por ejemplo, establecer que los dos primeros bytes indiquen la longitud del paquete recibido)
- Comparar los *buffers* con algún *string*
- Enviar *strings* alojados en la ROM del PIC (máximo 2048 bytes)
- Enviar *strings* alojados en la RAM del PIC (máximo 2048 bytes)
- Eliminar el servidor

Objetivo

Conocer las características para el módulo cuando es establecido como servidor SSL.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_SSL_SERVER

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 LED de 2.2V y 20 *mA*
- 1 resistencias de 56 Ω

Descripción

El microcontrolador será establecido como un servidor SSL en el puerto 3030, en el cual se ofrecerá el servicio de control de un LED. A su vez, se indicará el estado actual de dicho LED, mediante la pulsación de un botón.

Se utilizará una computadora que deberá estar conectada a la misma red a la que está conectado el módulo. Con ella se creará, mediante el programa *packet sender*, un cliente especificando la dirección IP privada, el puerto 3030 y el protocolo SSL.

Para controlar el estado del LED, el cliente tendrá que enviar los mensajes "ON" y "OFF" para el encendido y apagado respectivamente. Cada que vez se envíe cualquiera de estos mensajes, el servidor responderá indicando que la acción se ha realizado correctamente. También al pulsar el botón el servidor enviará el estado actual del LED al cliente conectado en ese momento.

Código

El código que se deberá ejecutar el microcontrolador PIC es el mostrado en la figura P4.4, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P4.5.

Resultados

Al ejecutarse el código del microcontrolador, se mostrará en la terminal la dirección IP privada que le fue asignada.

Una vez que el cliente se conecte al servidor, podrá encender o apagar un LED y cada vez que realice cualquiera de estas acciones tendrá que obtener una respuesta del servidor tal y como se muestra en la figura P4.1. Al presionar el botón asociado al microcontrolador del servidor se deberá recibir un mensaje que indique sobre el estado actual del LED (ver figura P4.3).

Cuando desde el cliente se envíe el mensaje “Eliminar” (ver figura P4.2), el cliente será desconectado inmediatamente y no se podrá volver a conectar.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2NeSjFi

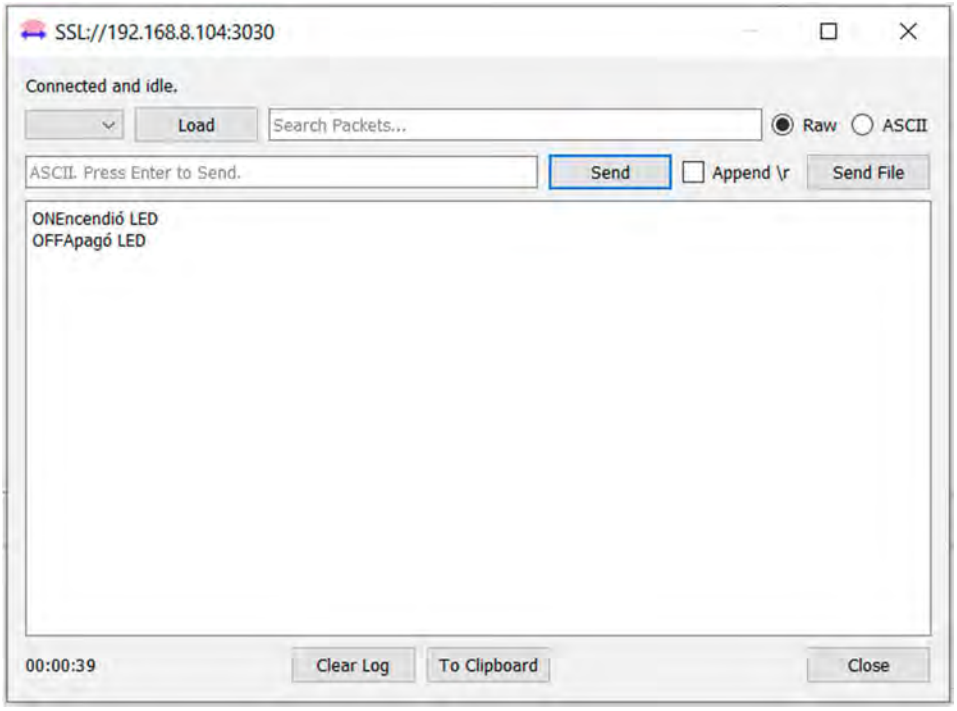


Figura P4.1 *Cliente SSL controlando el estado de un LED.*

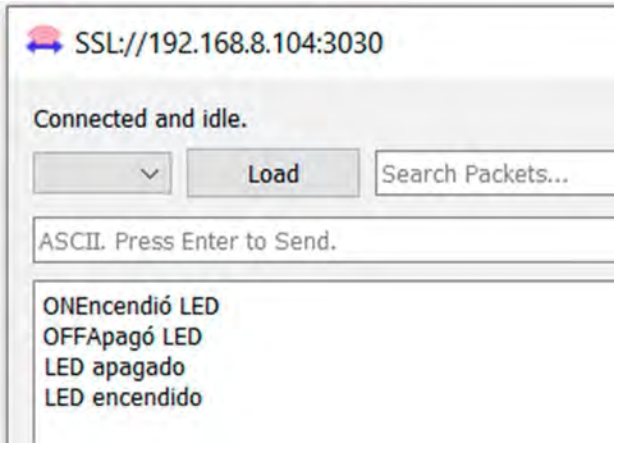


Figura P4.3 *Cliente SSL recibiendo el estado de un LED.*

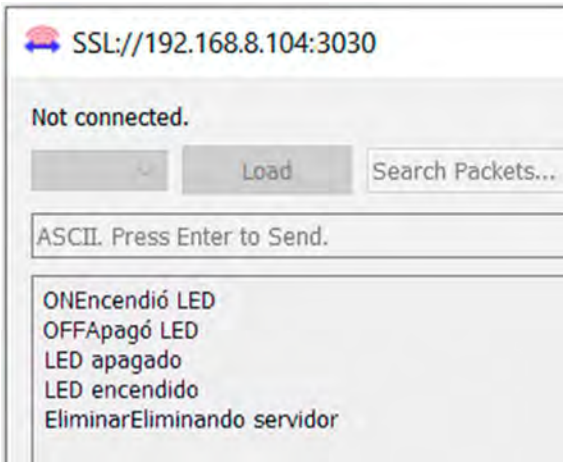


Figura P4.2 *Cliente SSL eliminado el servidor.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

int buffer[21]={0};

void main()
{
    /*Establece al PIN B7 como salida y al resto del puerto B
    como entradas*/
    set_tris_b(0b01111111);
    //Establece en cero todo el puerto B
    output_b(0);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtención de la dirección IP de la estación
    get_ip_station(buffer,sizeof(buffer));
    //Imprime en la terminal la dirección IP
    fprintf(PIC,"Dirección IP:%s\r\n",buffer);
    /*Establece buffer para recibir mensajes provenientes
    de clientes SSL*/
    set_buffer_ssl_server(buffer,sizeof(buffer));
    /*Crea servidor SSL estableciendo el puerto 3030,
     y un timeout de 4000 s*/
    create_ssl_server(3030,4000);
    while(1)
    {
        //Enciende LED si se recibe "ON"
        if(compare_SSLbuffer("ON",1))
        {
            output_bit(PIN_B7,1);
            send_server_ssl("Encendió LED\r\n");
        }
        //Apaga LED si se recibe "OFF"
        else if(compare_SSLbuffer("OFF",1))
        {
            output_bit(PIN_B7,0);
            send_server_ssl("Apagó LED\r\n");
        }
        //Al presionar botón para indicar estado actual del LED
        if(1==input_state(PIN_B6))
        {
            //Indica a cliente activos que el LED está encendido
             if(input_state(PIN_B7))
            {
                send_server_ssl("LED encendido\r\n");
            }
            //Indica a cliente activos que el LED está apagado
            else
            {
                send_server_ssl("LED apagado\r\n");
            }
             while(1==input_state(PIN_B6))
            {
                //Se mantiene hasta soltar el botón
            }
        }
        //Elimina el servidor si se recibe "Eliminar"
        if (compare_SSLbuffer("Eliminar",1))
        {
            send_server_ssl("Eliminando servidor\r\n");
            delete_ssl_server();
        }
    }

}
```

Figura P4.4 *Código de la práctica 4.*

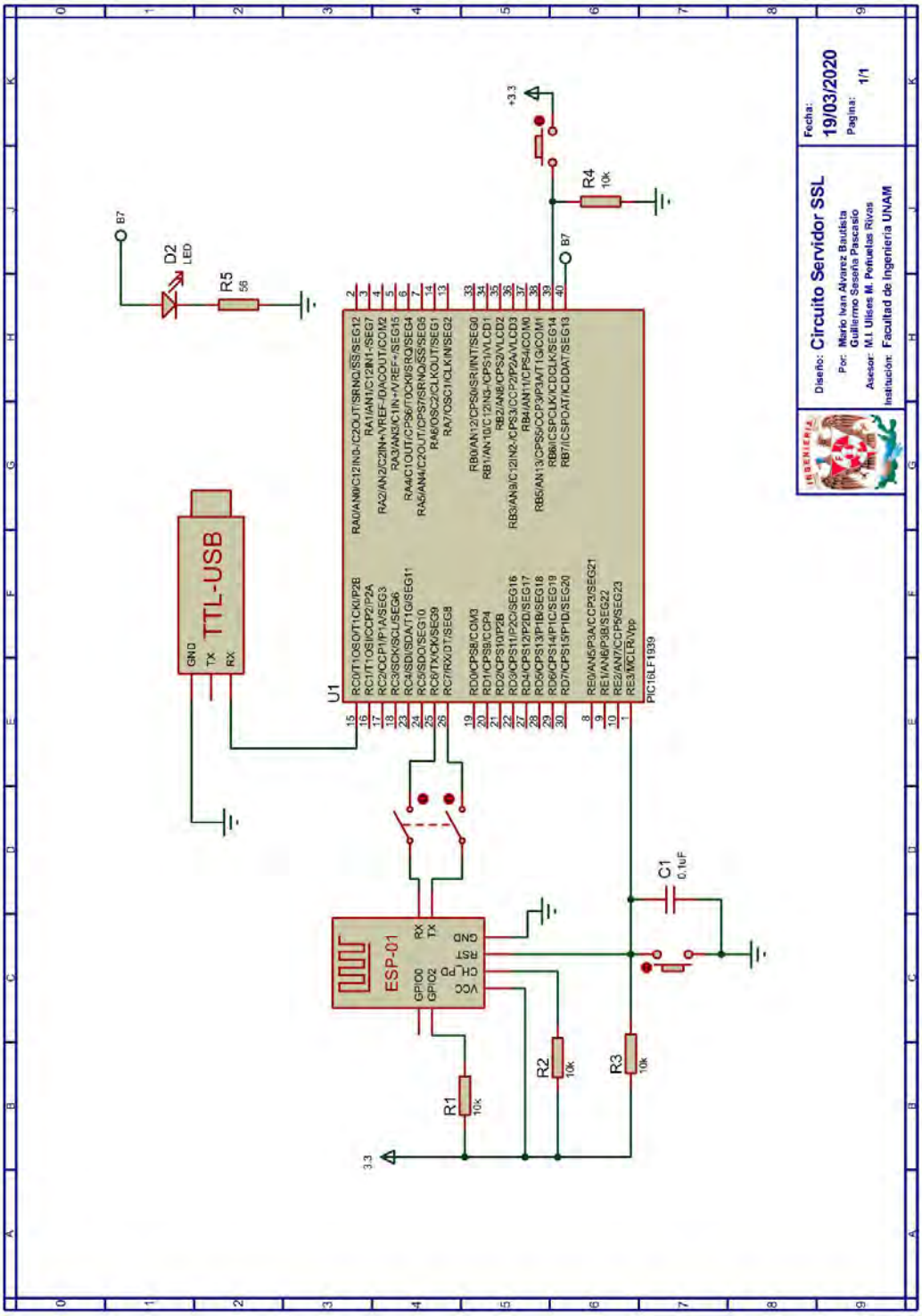


Figura P4.5 *Circuito de la práctica 4.*

Notas

- Se recomienda usar el programa Hterm para simular la terminal.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- En *packet sender* se recomienda habilitar “ignorar los errores SSL” derivados de certificados autofirmados, expirados entre otros.

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.
2. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
3. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
4. Werstén, B., *Implementing the Transport Layer Security Protocol for Embedded Systems.* 2007.
5. Kurose, J.F. and K.W. Ross, *Computer networking : A top-down approach*. 6th ed. 2013.

Práctica 5 Punto de acceso

Introducción

El punto de acceso (*Access Point*, AP) o también conocido como estación base, es el responsable de coordinar, enviar y recibir los paquetes datos desde y hacia los diferentes dispositivos inalámbricos, conocidos como estaciones, estos dispositivos deberán estar dentro del área de cobertura del punto de acceso [3]. A lo anterior se le conoce como una arquitectura BSS (*Basic Service Set*) (ver figura P5.1).

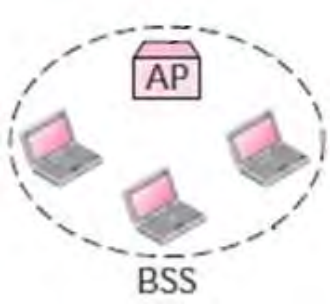


Figura P5.1 *AP en arquitectura BSS. Modificado de [1].*

Aunque los puntos de acceso se pueden conectar a un dispositivo de interconexión como un *router* para conectarse a otras redes (AP funciona como un retransmisor) [3], en el módulo WiFi esto no es posible. Cuando se crea un punto de acceso se asigna a éste un identificador de servicio de una o dos palabras (*Service Set Identifier*, SSID), un número de canal (ver tabla P5.1) y en algunos casos, una contraseña para autenticar al dispositivo que intenta conectarse al punto de acceso [3].

Tabla P5.1 *Canales de frecuencia[2].*

Canal	Frecuencia (MHz)	Canal	Frecuencia (MHz)
1	2412	8	2447
2	2417	9	2452
3	2422	10	2457
4	2427	11	2462
5	2432	12	2467
6	2437	13	2472
7	2442	14	2484

En este caso, los módulos WiFi que integran el SoC ESP8266 o alguno de los SoC ESP32 implementan el conjunto de protocolos TCP/IP, los estándares 802.11 b/g/n y una interfaz virtual WiFi, con su propia dirección MAC, para crear un *softAP,* es decir, un punto de acceso habilitado por *software*, para realizar operaciones en una arquitectura BSS [2, 4]. El número máximo de dispositivos que pueden conectarse al punto de acceso es de 5 y permite que éste sea abierto o utilice WPA PSK, WPA2 PSK y WPA WPA2 PSK como método de cifrado [5].

El punto de acceso creado por el módulo WiFi asignará las direcciones IP a las estaciones que se conecten mediante un servidor DHCP (*Dynamic Host*

Configuration Protocol o Protocolo dinámico de identificación de *host*) [5].

La biblioteca ESP permite realizar las siguientes acciones cuando el módulo funciona como un punto de acceso:

- Crear un punto de acceso.
- Desconectar a una estación del punto de acceso, conociendo previamente la dirección MAC de éste.
- Obtener o establecer la dirección IP del punto de acceso.
- Obtener o establecer la dirección MAC del punto de acceso.

Objetivo

Mostrar cómo crear un punto de acceso y reconocer su utilidad para aplicaciones ajenas.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará la tarjeta de desarrollo ESP32-DevKitC.

Habilitación de aplicaciones

ACCESS_POINT.

Materiales

- 1 microcontrolador PIC
- 1 teléfono inteligente
- 1 computadora
- 1 módulo WiFi
- 1 interruptor DPST
- 2 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

Se creará un punto de acceso abierto con el nombre ESP_FI. Este punto de acceso será utilizado para manejar un teléfono inteligente como segunda pantalla de una computadora. Para esto se deberá instalar en el teléfono inteligente la aplicación *space desk* desarrollada por datronicsoft, la cual está disponible en la *play store* (e instalar el *sofware* asociado a éste también en la computadora).

Tanto el teléfono inteligente como la computadora estarán conectados al punto de acceso creado por el módulo.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P5.2, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P5.4.

Resultados

Al ejecutar el programa del microcontrolador PIC y haberse creado el punto de acceso, debe aparecer el nombre de éste en la lista de redes tanto en la computadora como en el teléfono inteligente (ver figura P5.3) y se deberán conectar ambos dispositivos a dicho punto de acceso. Después se deberá abrir la aplicación y seguir las instrucciones que ahí se mencionan.

Figura P5.3 *Punto de acceso creado.*

Si todo se ha hecho de manera correcta, en la pantalla del teléfono inteligente se observará la imagen del monitor de la computadora.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2ARrIeP

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //creación de un punto de acceso
    begin_ap(0,"ESP_FI",NULL);
    while(1)
    {
    }
}
```

Figura P5.2 *Código de la práctica 5.*

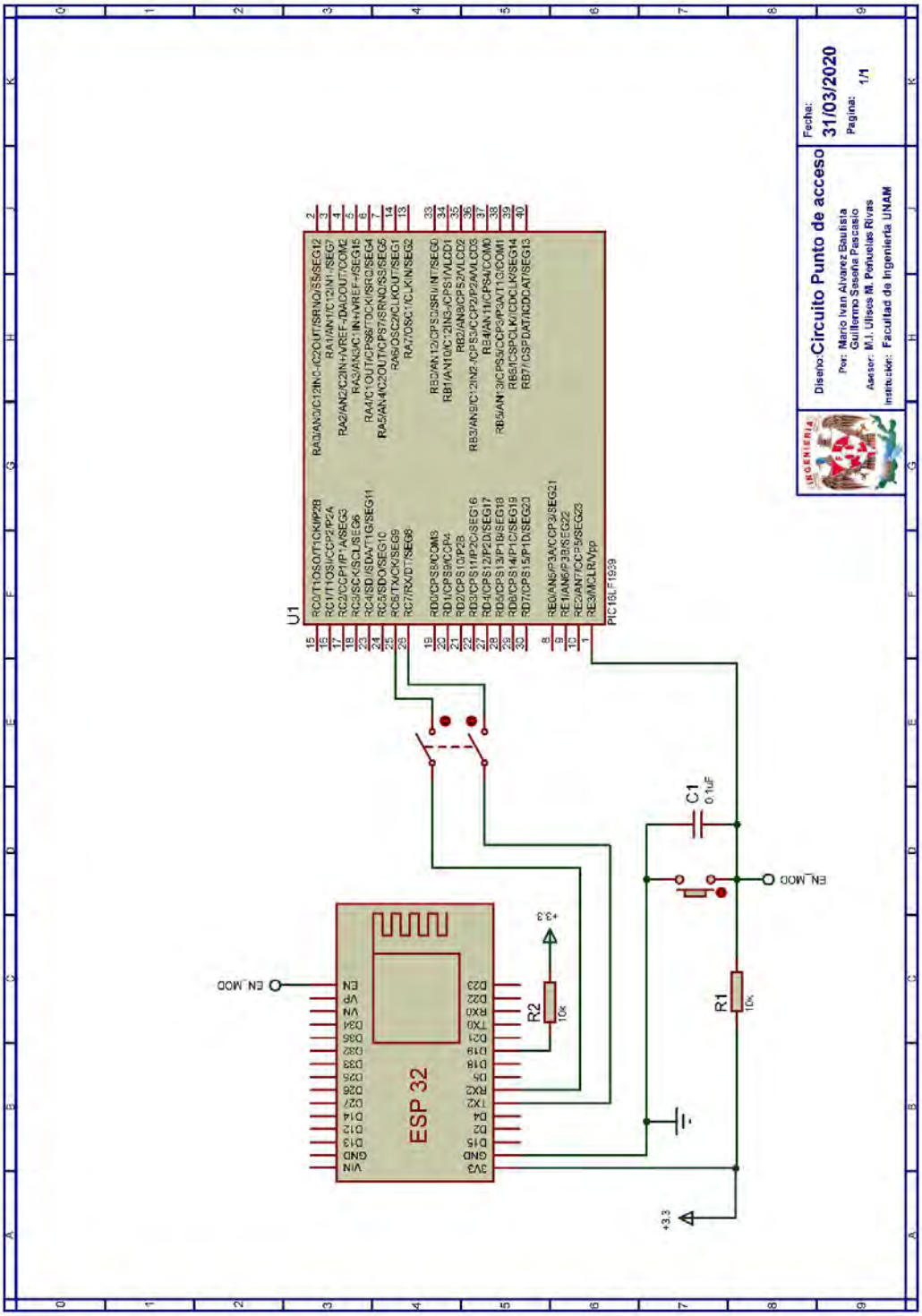


Figura P5.4 *Circuito de práctica 5.*

Nota

- Se deberá instalar la aplicación *space desk* en el teléfono inteligente y en la computadora se debe descargar el *software* de ésta desde https://spacedesk.net/ y seguir las instrucciones que ahí se describen.

Referencias

1. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
2. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.
3. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
4. Espresssif Systems, *ESP32 Series DataSheet Version 3.2*. 2019.
5. Espressif Systems, *ESP8266 AT Instruction Set Version 3.0.2*. 2019.

Práctica 6 DNS

Introducción

Todos los dispositivos conectados a internet son identificados de manera única y jerárquica a través de una dirección IP. Esta forma de identificación es efectiva entre máquinas, sin embargo, para la navegación por internet, a los humanos nos resulta impráctico recordar las direcciones IP de los servidores. Ante esta situación surgió el Sistema de nombres de dominio (*Domain Name System,* DNS) que asocia nombres legibles únicos con direcciones IP, con el fin de facilitar la navegación por internet [1].

DNS ha desarrollado un sistema jerárquico de nombres basado en dominios, cuya base de datos se encuentra distribuida en servidores colocados estratégicamente en diversas partes del mundo. Los dominios están divididos por niveles, en el primer nivel se encuentran los genéricos y los dominios de países, para éstos últimos se le ha asignado a cada país un dominio. A partir de los dominios de primer nivel se dividen en subdominios y éstos a su vez en más subdominios tal como se muestra en la figura P6.1 [2].

DNS es una aplicación que trabaja encima del protocolo de transporte UDP y está basado en el modelo de solicitud-respuesta. De manera simplificada, para que un dispositivo traduzca un nombre de dominio a una dirección IP debe ejecutarse como un cliente DNS, el cual realiza una petición a un servidor DNS local pasando como parámetro el nombre del dominio, el servidor local recibirá la solicitud y realiza un proceso iterativo de solicitudes a otros servidores DNS hasta obtener la dirección IP solicitada, no obstante, para reducir el tráfico DNS los servidores DNS e incluso el cliente cuentan con un caché para almacenar las direcciones IP asociados a dominios ya solicitados [1].

Los módulos ESP pueden ejecutarse como un cliente DNS ejecutando comandos para solicitar la dirección IP de algún dominio al servidor

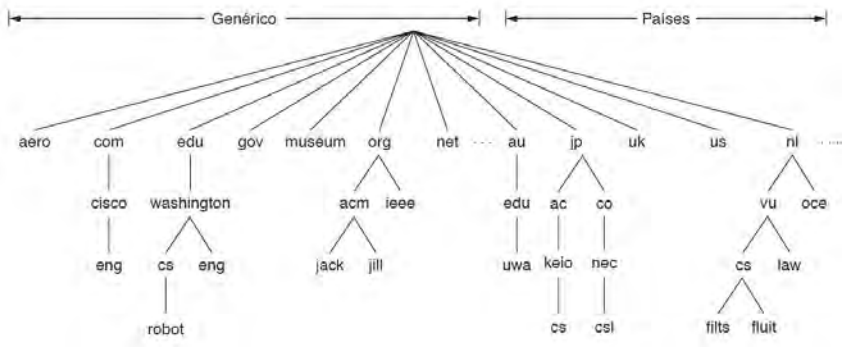


Figura P6.1 *Porción del espacio de nombres de dominio de internet [2].*

208.67.222.222. Además, para comunicarse con algún puerto TCP o UDP remoto, el módulo realiza la obtención de la IP si es que se le ha especificado algún nombre de dominio.

Objetivo

Obtener la dirección IP de algunos servidores usando DNS.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_DNS

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

En esta práctica el módulo se conectará a un punto de acceso con conexión a internet y solicitará la dirección IP de dos servidores usando nombres de dominios, dicha dirección se mostrará en una terminal.

Código

El código que deberá ejecutar para esta práctica se muestra en la figura P6.4, en el que se debe ingresar nombre y contraseña de un AP con acceso a internet.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P6.5.

Resultados

Al ejecutarse el código del microcontrolador, en la terminal se mostrará la dirección IP pública del servidor unam.mx y de ingenieria.unam.mx. (ver figura P6.2) Para comprobar que sean correctas se

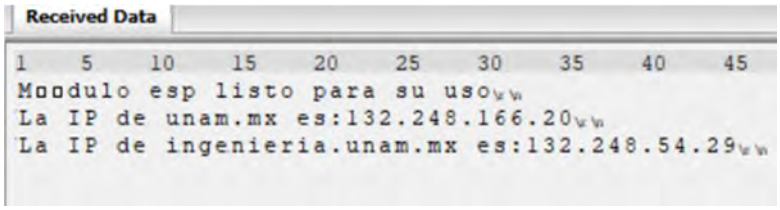


Figura P6.2 *Direcciones IP de los dos servidores consultados con el módulo.*

realiza un ping desde el Símbolo de Sistema de Windows (ver figura P6.3).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3efCY3e

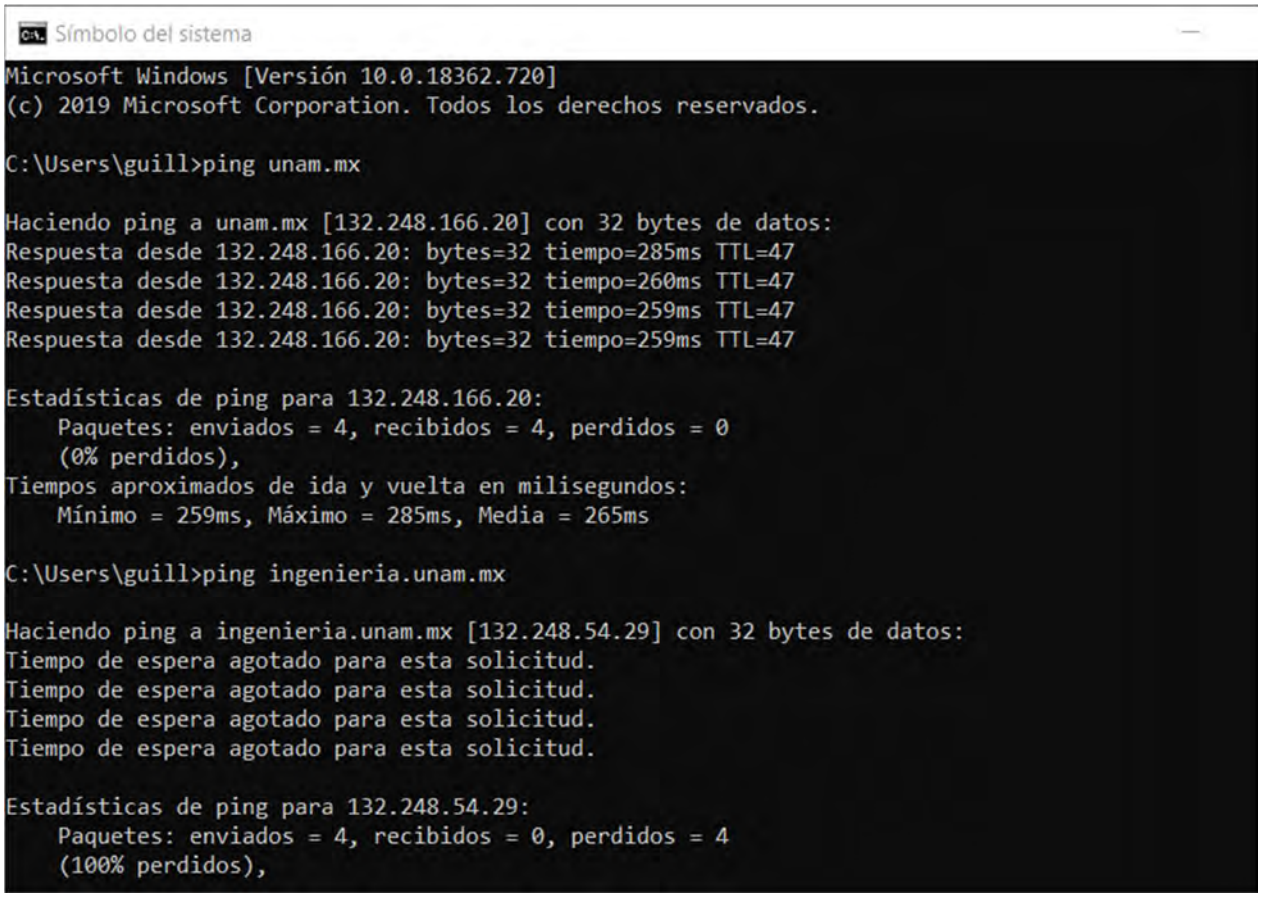


Figura P6.3 *Ping para comprobación de las direcciones IP de los servidores.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char ip[21]={0};

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Habilita DNS
    enable_dns();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtiene la dirección IP de unam.mx
    request_dns("unam.mx",ip,sizeof(ip));
    fprintf(PIC,"La IP de unam.mx es:%s\r\n",ip);
    delay_ms(1000);
    //Obtiene la dirección IP de ingenieria.unam.mx
    request_dns("ingenieria.unam.mx",ip,sizeof(ip));
    fprintf(PIC,"La IP de ingenieria.unam.mx es:%s\r\n",ip);
    delay_ms(1000);
}
```

Figura P6.4 *Código de la práctica 6.*

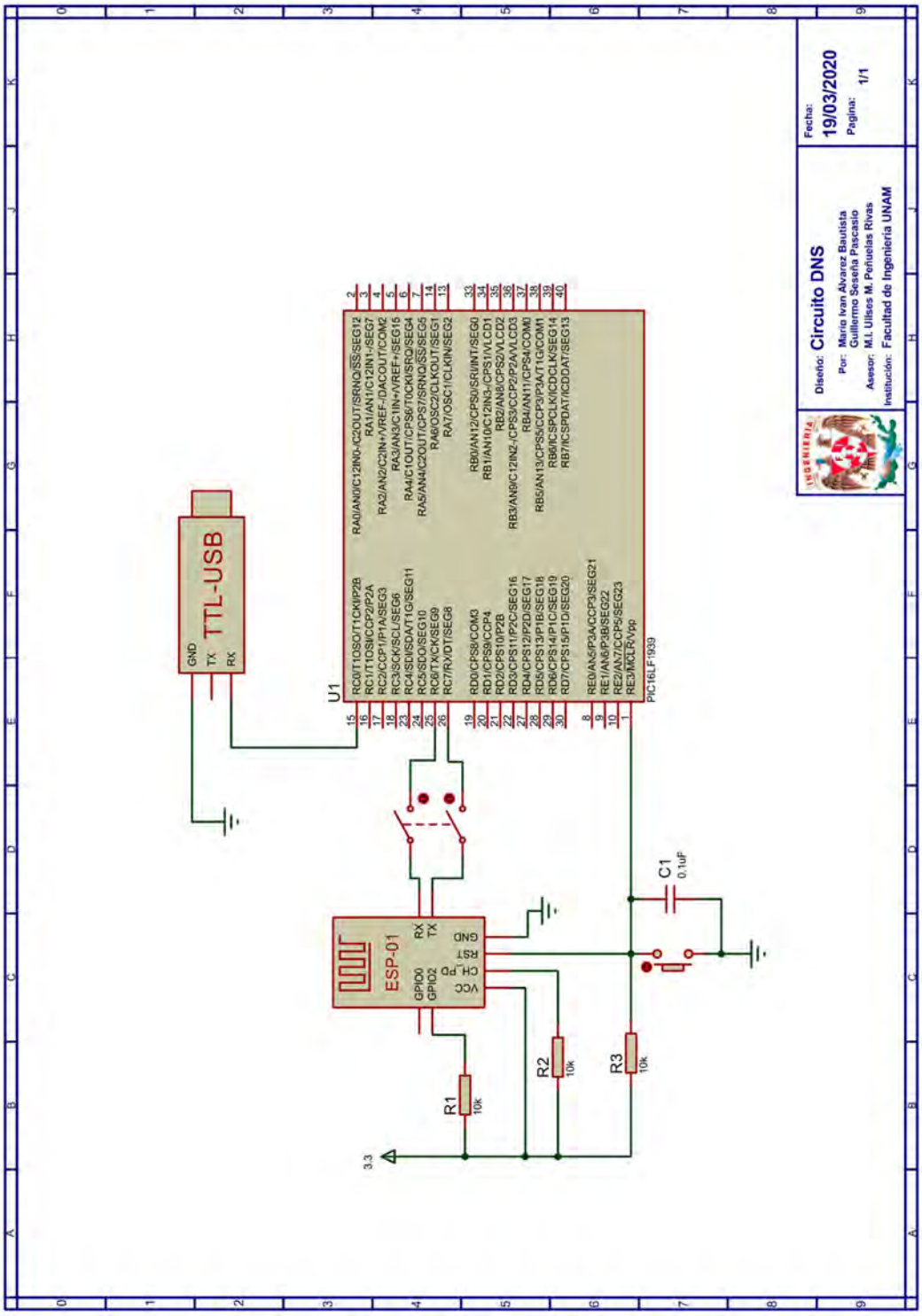


Figura P6.5 *Circuito de la práctica 6.*

Notas

- Se recomienda usar el programa Hterm para simular la terminal.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.

Referencias

1. Kurose, J.F. and K.W. Ross, *Computer networking : A top-down approach*. 6th ed. 2013.
2. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 7 Cliente TCP

Introducción

El Protocolo de control de transmisión (*Transmission Control Protocol*,TCP) es el protocolo de transporte más popular de internet, el cual provee de un servicio seguro orientado a la conexión, en que la información enviada es secciona en pequeños paquetes, los cuales son reconstruidos de forma ordenada por el destinatario, garantizando así, la integridad de la información que se transmite [1].

Los módulos ESP no sólo pueden ser utilizados como servidores TCP, sino que también pueden ser empleados como clientes usando dicho protocolo. Los módulos disponen de varios canales de comunicación, en cada uno de ellos se puede establecer una conexión con algún servidor. Además, los módulos pueden intercambiar mensajes con los servidores sin emplear algún protocolo de aplicación, si no, directamente de la capa de transporte.

La biblioteca ESP permite utilizar el canal por defecto del módulo (uni-conexión) o varios canales (multi-conexión) y así disponer de varios clientes a la vez que no necesariamente tienen que usar el protocolo TCP, sino que pueden funcionar con varios clientes UDP y/o un cliente SSL al mismo tiempo, incluso se puede habilitar el o los servidores del módulo siempre y cuando no utilicen los mismos canales.

Del mismo modo que el servidor TCP, la biblioteca ESP extrae y envía mensajes directamente de como el módulo lo obtiene de la capa de transporte sin ningún formato, dejando al cliente la libertad para definir las reglas del intercambio de información. La biblioteca ESP permite realizar las siguientes acciones:

- Establecer conexión con uno o más servidores TCP, especificando el puerto remoto de cada uno de los servidores
- Establecer un *buffer* compartido para los clientes (tomar en cuenta que los paquetes recibidos son guardados como *strings,* por ello al mensaje recibido se le inserta el terminador \0, si se requiere recibir información binaria el usuario tiene que establecer una forma para reconocer en donde termina cada mensaje recibido, por ejemplo, establecer que los dos primeros bytes indiquen la longitud del paquete recibido)
- Comparar el *buffer* con algún string
- Enviar *strings* alojados en la ROM del PIC (máximo 2048 bytes)
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 2048 bytes)

- Cerrar la conexión de cada cliente

Objetivo

Conocer algunas de las características disponibles para el módulo cuando es establecido como cliente TCP

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_SSL_TCP_UDP_CLIENT_UNICON para la primera actividad
- STATION y E_SSL_TCP_UDP_CLIENT_MULTICON para la segunda actividad

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF
- 1 pantalla LCD 16x2

Descripción

Esta práctica está divida en dos actividades:

- Uni-conexión
- Multi-conexión

I Uni-conexión

En esta primera actividad, se establecerá al módulo como un cliente TCP, el cual usa el canal de comunicación por defecto del módulo (uni-conexión) para comunicarse con un servidor. El módulo se deberá conectar a un servidor creado en el programa *packet sender*, especificando la dirección IP de la computadora y el puerto TCP del que dispone el programa.

El servidor al que se conectará no ofrece un servicio en específico, por lo que sólo se envían mensajes desde el servidor al módulo. Al conectarse al servidor, el módulo saludará con el mensaje de “Hola Servidor”, a partir de allí en la LCD se mostrarán todos los mensajes que se envían al módulo, además se podrá desconectar al módulo enviando el mensaje “Cerrar”.

II Multi-conexión

Para esta segunda actividad se realizará las mismas acciones que en la práctica anterior, pero con la diferencia de que se usarán dos canales de comunicación para crear dos clientes. Para esta actividad los dos clientes pueden ser conectados al mismo servidor creado en *packet sender,* o si se prefiere se puede utilizar dos distintos usando otro *software*.

Código

El código que deberá ejecutar el microcontrolador PIC para la primera actividad es el mostrado en la figura P7.4, en que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP.
- Dirección IP o nombre de dominio del servidor al que se conecta el módulo y su puerto.

Para la segunda actividad se ejecuta el código mostrado en la figura P7.5, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP
- Dirección IP o nombre de dominio del primer servidor al que se conecta el módulo del servidor y su puerto.
- Dirección IP o nombre de dominio del segundo servidor al que se conecta el módulo del servidor y su puerto.

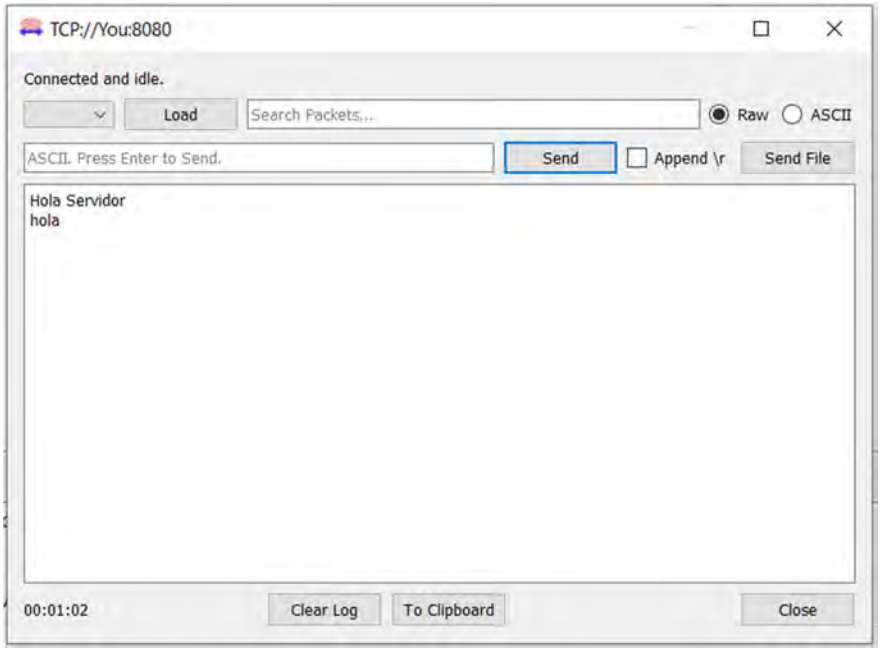


Figura P7.1 *Conexión del cliente creado en el módulo y envío de mensaje.*

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P7.6.

Resultados

Para la primera actividad, lo primero que sucederá es que se abrirá una ventana en *packet sender*, esto significa que el módulo se ha conectado como cliente, y se mostrará el mensaje de saludo (ver figura P7.1), posteriormente cada vez que se envíe un mensaje al módulo éste será mostrado en la LCD. Al enviar el mensaje "Cerrar", el módulo indica que se desconectará y eventualmente se indica en la ventana que se ha finalizado la conexión (ver figura P7.2).

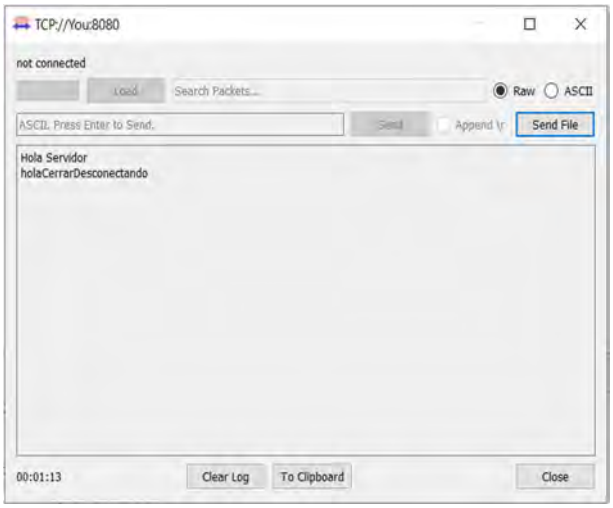


Figura P7.2 *Desconexión del cliente creado*.

Para la segunda actividad aparecerán dos ventanas, a cada una de éstas le corresponde un cliente creado en el módulo (ver figura P7.3), cada que se envíe un mensaje de alguna de estas ventanas, se mostrará en la LCD. Si el mensaje fue enviado al primer cliente en la LCD se mostrará que el mensaje proviene del canal 0, mientras que, para el segundo cliente se mostrará que proviene del canal 1.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y los códigos de la práctica:

https://bit.ly/2zPOSBS

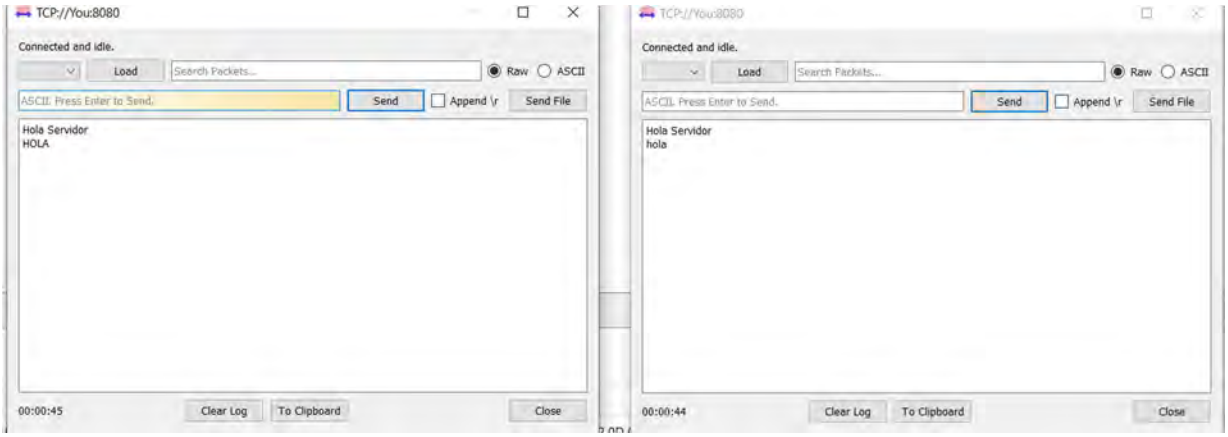


Figura P7.3 *Conexión de dos clientes creados en el módulo y envío de mensaje.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define server "dirección IP del servidor"
#define port 8080

int buffer[21]={0};
const char saludo[]="Hola Servidor\r\n";

void main()
{
    //Inicialización de la LCD
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Establece buffer para recibir mensajes provenientes
    del servidor TCP*/
    set_buffer_client(buffer,sizeof(buffer));
    //Crea un cliente TCP para el servidor y puerto indicado
    create_tcp_pclient(server, port);
    //Envía saludo al conectarse
    send_client_const(saludo);
    while(1)
    {
        //Limpia la LCD
        printf(lcd_putc,"\f");
        //Posiciona escritura en la LCD
        lcd_gotoxy(5,1);
        //Imprime en la LCD
        printf(lcd_putc,"Mensaje:");
        //Posiciona escritura en la LCD
        lcd_gotoxy(1,2);
        //Imprime en la LCD
        printf(lcd_putc,"%s",buffer);
        //Cierra la conexión del cliente
        if(compare_Cbuffer("Cerrar",1))
        {
            send_client("Desconectando");
            delete_tcp_client();
        }
        delay_ms(500);
    }
}
```

Figura P7.4 *Código de la primera actividad de la práctica 7.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define server_1 "dirección IP del servidor"
#define port_1 8080
#define server_2 "dirección IP del servidor"
#define port_2 8080

int buffer[21]={0};
const char saludo[]="Hola Servidor\r\n";

void main()
{
    //Inicialización de la LCD
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Establece buffer para recibir mensajes provenientes
    de los servidores TCP*/
    set_buffer_client(buffer,sizeof(buffer));
    /*Crea primer cliente TCP para el servidor y
    puerto indicado para el canal 0*/
    create_tcp_pclient(server_1,port_1,0);
    //Envia saludo al conectarse
    send_mclient_const(0,saludo);
    /*Crea segundo cliente TCP para el servidor y
    puerto indicado indicado para el canal 1*/
    create_tcp_pclient(server_2,port_2,1);
    //Envia saludo al conectarse
    send_mclient_const(1,saludo);
    while (1)
    {
        //Limpia la LCD
        printf(lcd_putc,"\f");
        //Posiciona escritura en la LCD
        lcd_gotoxy(5,1);
        //Imprime en la LCD
        printf(lcd_putc,"Canal %u:",buffer[20]);
        //Posiciona escritura en la LCD
        lcd_gotoxy(1,2);
        //Imprime en la LCD
        printf(lcd_putc,"%s",buffer);
        //Cierra la conexión del cliente 1 (canal 0)
        if(buffer[20]==0 && compare_Cbuffer("Cerrar",1))
        {
            send_client(0,"Desconectando");
            delete_tcp_client(0);
        }
        //Cierra la conexión del cliente 2 (canal 1)
        else if(buffer[20]==1 && compare_Cbuffer("Cerrar",1))
        {
            send_client(1,"Desconectando");
            delete_tcp_client(1);
        }
        delay_ms(100);
    }
}
```

Figura P7.5 *Código de la segunda actividad de la práctica 7.*

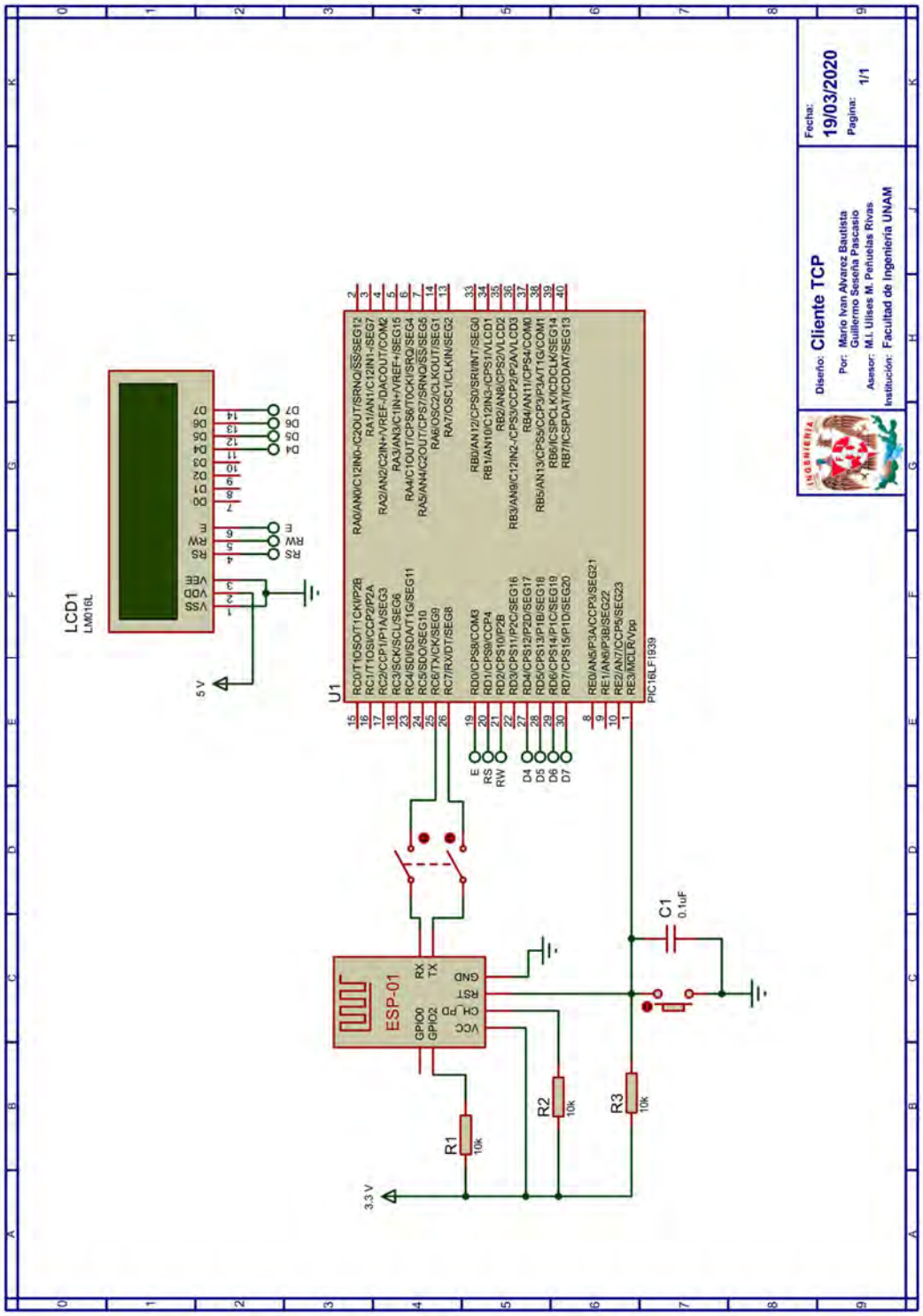


Figura P7.6 *Circuito de la práctica 7.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 8 Cliente UDP

Introducción

El Protocolo de datagramas de usuario (*User Datagram Protocol,* UDP) es un protocolo de transporte de internet que envía paquetes llamados datagramas. A diferencia de TCP, UDP es un protocolo ligero y simple, que no cuenta con mecanismos para garantizar la entrega correcta de la información, por lo que ofrece un servicio poco confiable. Además, UDP ofrece un servicio sin conexión, esto quiere decir que al enviar información no existe interacción previa con el destinatario para asegurar que este último esté disponible para recibir información. UDP solo se encarga de enviar datagramas al puerto destino sin que sea necesario obtener una respuesta de él [1, 2].

Ya que UDP no se basa en la conexión, la transmisión de información se efectúa entre iguales, por lo que el modelo cliente-servidor es construido por completo en la capa de aplicación.

Los módulos ESP pueden enviar y recibir información directamente de la capa de transporte usando el protocolo UDP. Aunque los roles se determinan en la capa de aplicación, se puede decir que los módulos se establecen como clientes UDP, ya que, a un canal de comunicación se le asigna un puerto UDP de origen, una dirección IP destino y puerto destino, por lo que puede recibir información de distintos dispositivos, pero solo puede enviar información a un solo dispositivo en específico, por lo que se asemeja más a un cliente que a un servidor.

La biblioteca ESP permite utilizar el canal de comunicación por defecto del módulo para establecer un cliente UDP o emplear varios canales para más clientes, para la biblioteca, a la primera opción se le conoce como uni-conexión, mientras que a la segunda se le llama multi-conexión. Es importante señalar que los nombres de estos modos fueron designados para dar homogeneidad a todos los tipos de clientes que se pueden establecer en el módulo, ya que como se ha indicado en UDP no existe conexión, más propiamente se podría referir como uni-canal y multi-canal respectivamente. La biblioteca ESP permite realizar las siguientes acciones:

- Habilitar uno o más clientes UDP especificando para cada uno de ellos su puerto origen, la dirección IP o nombre de dominio y puerto remoto del servidor a quien envía información.
- Establecer un *buffer* compartido para los clientes (tomar en cuenta que los paquetes recibidos son guardados como *strings* por ello al mensaje recibido se le inserta el terminador \0, si se requiere recibir información binaria el usuario

tiene que establecer una forma para reconocer en donde termina cada mensaje recibido, por ejemplo, establecer que los dos primeros bytes indiquen la longitud del paquete recibido).
- Comparar el *buffer* con algún string.
- Enviar *strings* alojados en la ROM del PIC (máximo 2048 bytes).
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 2048 bytes).
- Cerrar la conexión de cada cliente.

Objetivo

Conocer algunas de las características disponibles para el módulo cuando es establecido como cliente UDP

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_SSL_TCP_UDP_CLIENT_UNICON para la primera actividad
- STATION y E_SSL_TCP_UDP_CLIENT_MULTICON para la segunda actividad

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 pantalla LCD 16x2

Descripción

Esta práctica está divida en dos actividades:

- Uni-conexión
- Multi-conexión

I Uni-conexión (uni-canal)

En esta primera actividad, se establecerá al módulo como un cliente UDP, el cual usa el canal por defecto de comunicación del módulo para comunicarse con un servidor UDP. El módulo enviará mensajes a un servidor UDP creado por el programa *packet sender*, especificando la dirección IP de la computadora y el puerto UDP del que dispone el programa. El servidor con el que se comunicará no ofrece algún

servicio específico. En la LCD se mostrarán todos los mensajes que se envían desde *packet sender*, para los que se tendrán que especificar la IP asignada al módulo junto con el puerto local del cliente, además se podrá eliminar el cliente del módulo enviando el mensaje “Eliminar”.

II Multi-conexión (multi-canal)

Para esta segunda actividad se realizará las mismas acciones que en la actividad anterior, pero con la diferencia en que se hará uso de dos canales de comunicación para crear dos clientes. Los dos clientes podrán intercambiar información con el mismo puerto creado en *packet sender,* o si se prefiere se puede utilizar otro *software.*

Código

El código que deberá ejecutar el microcontrolador PIC para la primera actividad es el mostrado en la figura P8.4, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP
- Puerto local
- Dirección IP o nombre del dominio del servidor al que se comunica
- Puerto remoto

Para la segunda actividad se ejecuta el código mostrado en la figura P8.5 y figura P8.6, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP
- Puerto local del primer cliente
- Dirección IP o nombre del dominio del primer servidor al que se comunica
- Puerto remoto del primer servidor
- Puerto local del segundo cliente
- Dirección IP o nombre del dominio del segundo servidor al que se comunica
- Puerto remoto del segundo servidor

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P8.7.

Resultados

Para la primera actividad, lo primero que se observará en la LCD será la dirección IP asignada al módulo, posteriormente en el programa *packet sender* llegará el mensaje “Hola\r\n”. Se podrá enviar mensajes al módulo desde el programa escribiendo la dirección mostrada en la LCD y especificando el puerto local de cliente UDP (ver figura P8.1), cada

From IP	From Port	To IP	To Port	Method	Error	ASCII
You	56458	192....	800	UDP		hola
192.168.8.100	800	You	56458	UDP		Hola\r\n

Figura P8.1 *Intercambio de mensajes entre cliente y servidor UDP para uni-conexión (uni-canal).*

mensaje enviado será mostrado en la LCD.

Si se envía el mensaje “Eliminar” se eliminará el cliente, por lo que si mandan más mensajes no llegarán al módulo y en consecuencia no se mostrarán en la LCD (ver figura P8.2).

Para la segunda actividad sucederá lo mismo que en la anterior, pero ahora llegarán dos mensajes “Hola\r\n”, cada uno de éstos proviene de un puerto distinto, es decir de un cliente diferente.

Para enviar mensajes se indicará la dirección IP del módulo junto con el puerto de origen del respectivo cliente (ver figura P8.3). Cada mensaje que se envíe será mostrado en la LCD, si está destinado para el primer cliente, se indicará que proviene del canal 0, mientras que para el segundo cliente se indicará el canal 1. Cuando se elimine algún cliente, el módulo ya no recibirá los mensajes destinados para dicho cliente y en consecuencia no se mostrarán en la LCD.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y los códigos de la práctica:

https://bit.ly/3fKx9uZ

From IP	From Port	To IP	To Port	Method	Error	ASCII
You	56458	192....	800	UDP		HOLA
192.168.8.100	800	You	56458	UDP		Eliminando
You	56458	192....	800	UDP		Eliminar

Figura P8.2 *Mensajes envidos en la eliminación del cliente UDP para uni-conexión (uni-canal).*

From IP	From Port	To IP	To Port	Method	Error	ASCII
You	56458	192....	801	UDP		HOLA
You	56458	192....	800	UDP		hola
192.168.8.100	801	You	56458	UDP		Hola\r\n
192.168.8.100	800	You	56458	UDP		Hola\r\n

Figura P8.3 *Intercambio de mensajes entre clientes y el servidor UDP para multi-conexión (multi-canal).*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define server "dirección IP del servidor"
#define remote_port 56458
#define local_port 800

int buffer[21]={0};
const char saludo[]="Hola Servidor\r\n";

void main()
{
    //Inicialización de la LCD
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtención de la dirección IP
    get_ip_station(buffer,sizeof(buffer));
    //Posiciona escritura en la LCD
    lcd_gotoxy(2,1);
    //Imprime en la LCD
    printf(lcd_putc,"Direccion IP:");
    //Posiciona escritura en la LCD
    lcd_gotoxy(2,2);
    //Imprime en la LCD
    printf(lcd_putc,"%s",buffer);
    delay_ms(4000);
    /*Establece buffer para recibir mensajes provenientes
    del servidor UDP*/
    set_buffer_client(buffer,sizeof(buffer));
    //Crea un cliente UDP para el servidor y puerto indicado
    create_udp_client(server,remote_port,local_port,0);
    //Envía saludo al conectarse
    send_client_const(saludo);
    while(1)
    {
        //Limpia la LCD
        printf(lcd_putc,"\f");
        //Posiciona escritura en la LCD
        lcd_gotoxy(5,1);
        //Imprime en la LCD
        printf(lcd_putc,"Mensaje:");
        //Posiciona escritura en la LCD
        lcd_gotoxy(1,2);
        //Imprime en la LCD
        printf(lcd_putc,"%s",buffer);
        if(compare_Cbuffer("Eliminar",1))
        {
            send_client("Eliminando");
            delete_udp_client();
        }
        delay_ms(500);
    }
}
```

Figura P8.4 *Código de la primera actividad de la práctica 8.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define server_1 "dirección IP del servidor"
#define remote_port_1 56458
#define local_port_1 800
#define server_2 "dirección IP del servidor"
#define remote_port_2 56458
#define local_port_2 800

int buffer[21]={0};
const char saludo[]="Hola\r\n";

void main()
{
    //Inicialización de la LCD
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtención de la dirección IP
    get_ip_station(buffer,sizeof(buffer));
    //Posiciona escritura en la LCD
    lcd_gotoxy(2,1);
    //Imprime en la LCD
    printf(lcd_putc,"Direccion IP:");
    //Posiciona escritura en la LCD
    lcd_gotoxy(2,2);
    //Imprime en la LCD
    printf(lcd_putc,"%s",buffer);
    delay_ms(4000);
    /*Establece buffer para recibir mensajes provenientes
    de los servidores TCP*/
    set_buffer_client(buffer,sizeof(buffer));
    /*Crea primer cliente TCP para el servidor y
    puerto indicado para el canal 0*/
    create_udp_client(server_1,0,remote_port_1,local_port_1,0);
    //Envía saludo al conectarse
    send_mclient_const(0,saludo);
    /*Crea segundo cliente TCP para el servidor y
    puerto indicado indicado para el canal 1*/
    create_udp_client(server_2,1,remote_port_2,local_port_2,0);
    //Envía saludo al conectarse
    send_mclient_const(1,saludo);
```

Figura P8.5 *Primera parte del código de la segunda actividad de la práctica 8.*

```
    while (1)
    {
        //Limpia la LCD
        printf(lcd_putc,"\f");
        //Posiciona escritura en la LCD
        lcd_gotoxy(5,1);
        //Imprime en la LCD
        printf(lcd_putc,"Canal %u:",buffer[20]);
        //Posiciona escritura en la LCD
        lcd_gotoxy(1,2);
        //Imprime en la LCD
        printf(lcd_putc,"%s",buffer);
        //Cierra la conexión para el cliente 1 (canal 0)
        if(buffer[20]==0 && compare_Cbuffer("Eliminar",1))
        {
            send_client(0,"Eliminando");
            delete_tcp_client(0);
        }
        //Cierra la conexión para el cliente 2 (canal 1)
        else if(buffer[20]==1 && compare_Cbuffer("Eliminar",1))
        {
            send_client(1,"Eliminando");
            delete_tcp_client(1);
        }
        delay_ms(100);
    }
}
```

Figura P8.6 *Segunda parte del código de la segunda actividad de la práctica 8.*

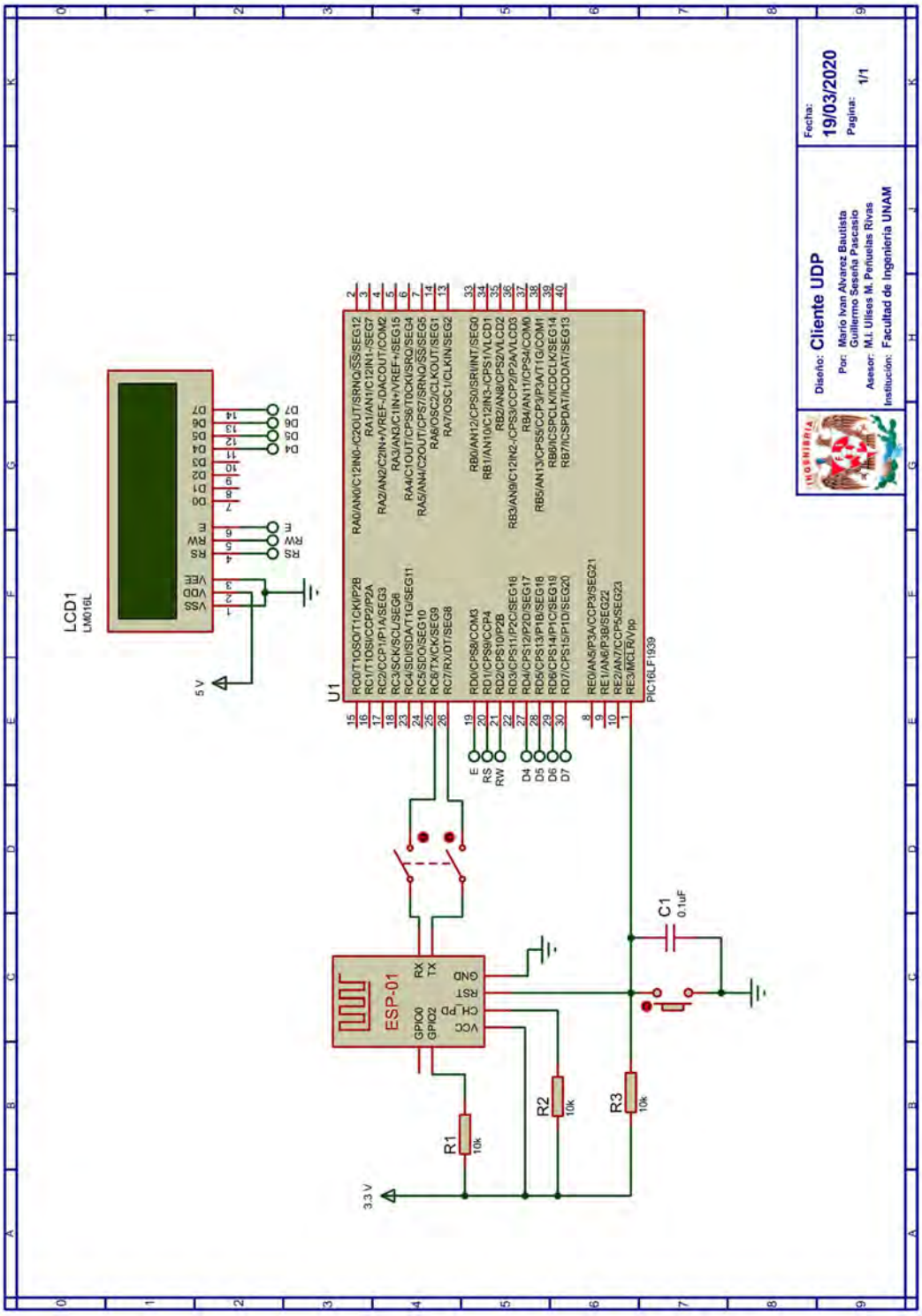


Figura P8.7 *Circuito de la práctica 8.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.
2. Kurose, J.F. and K.W. Ross, *Computer networking : A top-down approach*. 6th ed. 2013.

Práctica 9 Cliente transparente

Introducción

Como ya se ha mostrado en las prácticas 7 y 8, el módulo puede crear uno o varios clientes TCP y/o UDP. En esas prácticas el módulo transmite información en paquetes con un tamaño máximo de 2048 bytes, cada uno de estos paquetes es enviado a través de la ejecución del respectivo comando AT. No obstante, existe otra forma de transmitir información cuando el módulo es establecido como un cliente TCP o UDP, la cual es conocida como transparente.

Para este modo de transmisión solo se permite crear un cliente. Al activarse el cliente, los paquetes a enviar son llevados al módulo sin la necesidad de ejecutar algún comando, del mismo modo, la información que el módulo retransmite al PIC proveniente de otro dispositivo, no contiene ningún identificador y que es tratado por la biblioteca ESP. Otra característica de este modo de transmisión es que cuando se establece una conexión TCP y ésta sea cerrada al perder comunicación o cuando el servidor haya decidido finalizarla, el módulo automáticamente se volverá a conectar al servidor. Por ello, la conexión realmente termina hasta que se le indique al módulo que debe finalizarla, o hasta que le sea imposible al módulo la reconexión.

Ya que durante la transmisión no hay forma de reconocer en donde inicia y donde termina la información proveniente de otro dispositivo, la biblioteca ESP almacena cada byte que reciba del módulo en un *buffer*, si el *buffer* ha sido llenado se empezará a sobrescribir desde su posición inicial. Para el correcto almacenamiento los mensajes enviados al módulo deben contener un retorno de carro (\r) y un salto de línea (\n), entonces la biblioteca reconoce éstos como el final del mensaje y con ello puede insertar en el *buffer* el terminador del string (\0).

La biblioteca ESP permite realizar las siguientes acciones:

- Establecer conexión con servidor TCP o UDP.
- Establecer un *buffer* para la información proveniente del servidor (tomar en cuenta que los paquetes recibidos son guardados como *strings* por ello al mensaje recibido se le inserta el terminador \0, no es posible la recepción binaria, ya que ‘\r’ y ‘\n’ no pueden ser guardados puesto que son utilizados para indicar a la biblioteca el final del mensaje).
- Comparar el *buffer* con algún *string.*
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 2048 bytes)

- Concluir la transmisión transparente y finalización de conexión.
- Concluir la transmisión transparente y finalización de conexión para cualquier momento de la ejecución del programa en caso de que se presente un fallo.

Objetivo

Conocer las características disponibles para el módulo cuando es establecido como cliente transparente.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_TRANSPARENT_CLIENT

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 LED de 2.2V y 20 mA
- 1 resistor de 56 Ω

Descripción

Para esta práctica se establecerá al módulo como un cliente TCP usando la transmisión transparente. El módulo se conectará al servidor TCP creado en el programa *packet sender*. Este servidor no ofrecerá algún servicio en específico, solo servirá para enviar mensajes y controlar el estado de un LED en el microcontrolador PIC.

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica es el mostrado en la figura P9.5, en él se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP
- Dirección IP o nombre del dominio del servidor al que se comunica y su puerto.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P9.6.

Resultados

En el programa *packet sender* se abrirá una ventana en el momento que el módulo establezca conexión, posteriormente comenzará la recepción del mensaje “Activo” proveniente del módulo (ver figura P9.1).

Para controlar el LED se enviará los mensajes “on” y “off”, ambos deben ser enviado con un retorno de carro y un salto de línea (\r\n) para su correcto almacenamiento en el *buffer*.

Para este ejercicio se muestra un escenario en el que no se usan dichas secuencias de escape, partiendo de que el *buffer* está vacío “\0\0\0\0\0\0…\0”. Si se envía el mensaje “Hola”, en el *buffer* se tendría “Hola\0\0…\0”, ya que el mensaje no contiene el retorno de carro y salto de línea, el uC no sabrá hasta dónde finaliza el mensaje por ello, los bytes que se reciban posteriormente se guardaran en las siguientes posiciones.

Si después se envía “on\r\n” entonces en el *buffer* se tiene “Holaon\0…\0” aunque se haya enviado correctamente el mensaje de encender no realizará la acción indicada (ver figura P9.2), ya que no coincide el *string* “on\0”. Sin embargo, en el momento de recibir “\r\n” la biblioteca a insertado \0 al *buffer* y ha reiniciado la posición del *buffer*, así que la siguiente vez que se envíe “on\r\n” se guardará correctamente y con ello realizará la acción especificada (ver figura P9.3).

Si se cierra la ventana de la conexión se volverá a conectar el cliente en una nueva ventana, esto hasta que se envíe el mensaje “Eliminar”, en ese momento se cerrará la conexión (ver figura P9.4).

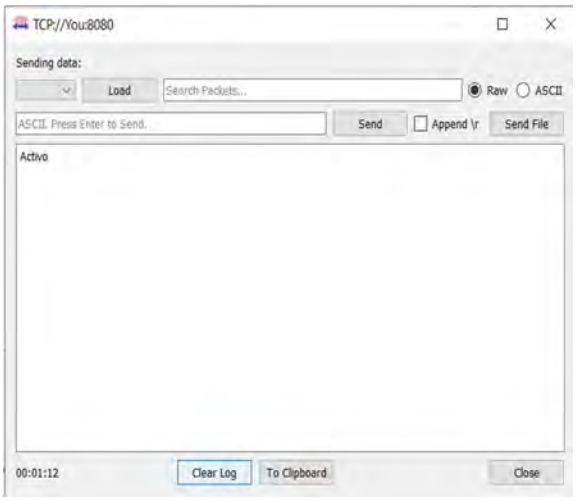


Figura P9.1 *Conexión del cliente TCP*

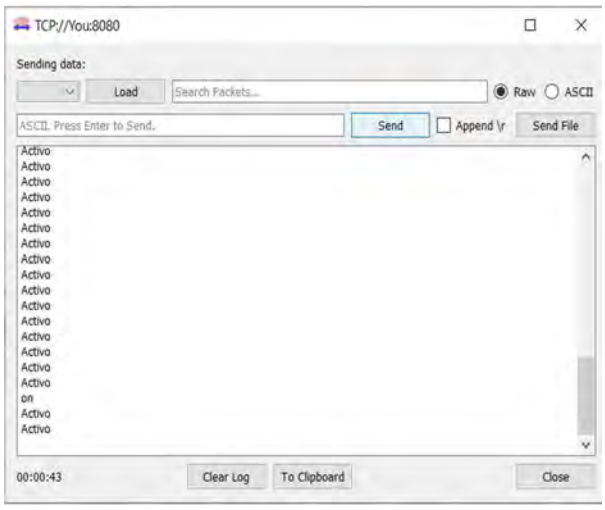


Figura P9.2 *Envío de mensaje “on\r\n” cuando el buffer no ha reiniciado su posición.*

Es importante señalar que cuando se utilice este modo de transmisión siempre debe ser finalizado, aunque el módulo sea reiniciado, la conexión se restablecerá y no se podrá controlar el módulo, inahbilitando por completo el funcionamiento de la biblioteca. Si se produce algún fallo o no se finalizó este modo de transmisión, existe una función con nombre `fail_delete_tclient` para finalizar dicha transmisión, la cual puede ser ejecutada en cualquier parte del código incluso antes de la inicialización del módulo.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3fLVPTX

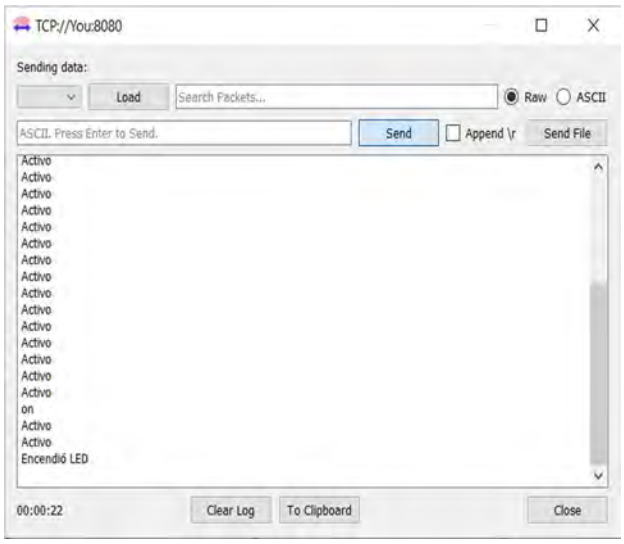


Figura P9.3 *Envío de mensaje "on\r\n" cuando el buffer ha reiniciado su posición.*

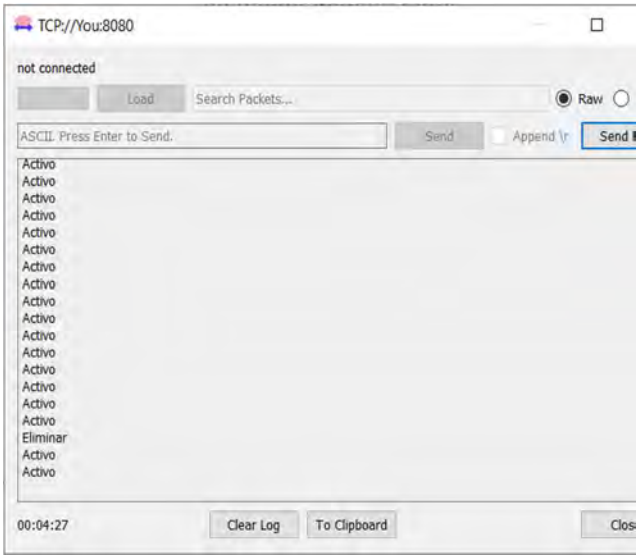


Figura P9.4 *Cierre de conexión de cliente TCP transparente.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define server "dirección IP del servidor"
#define remote_port 8080

char buffer[21]={0};

void main()
{
    /*Establece al PIN B7 como salida y al resto del puerto B
    como entradas*/
    set_tris_b(0b01111111);
    //Establece en cero todo el puerto B
    output_b(0);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Establece buffer para recibir mensajes provenientes
    del servidor TCP*/
    set_buffer_tclient(buffer,sizeof(buffer));
    //Crea un cliente TCP para el servidor y puerto indicado
    create_tclient(0,server,remote_port);
    while(1)
    {
        send_tclient("Activo\r\n");
        //Enciende LED si buffer es igual on\0
        if(compare_string("on",buffer))
        {
            send_tclient("Encendió LED\r\n");
            output_bit(PIN_B7,1);
            buffer[0]='\0';
        }
        //Enciende LED si buffer es igual off\0
        else if(compare_string("off",buffer))
        {
            send_tclient("Apagó LED\r\n");
            output_bit(PIN_B7,0);
            buffer[0]='\0';
        }
        if(compare_string("Eliminar",buffer))
        {
            delete_tclient();
        }
        delay_ms(500);
    }
}
```

Figura P9.5 *Código de la práctica 9.*

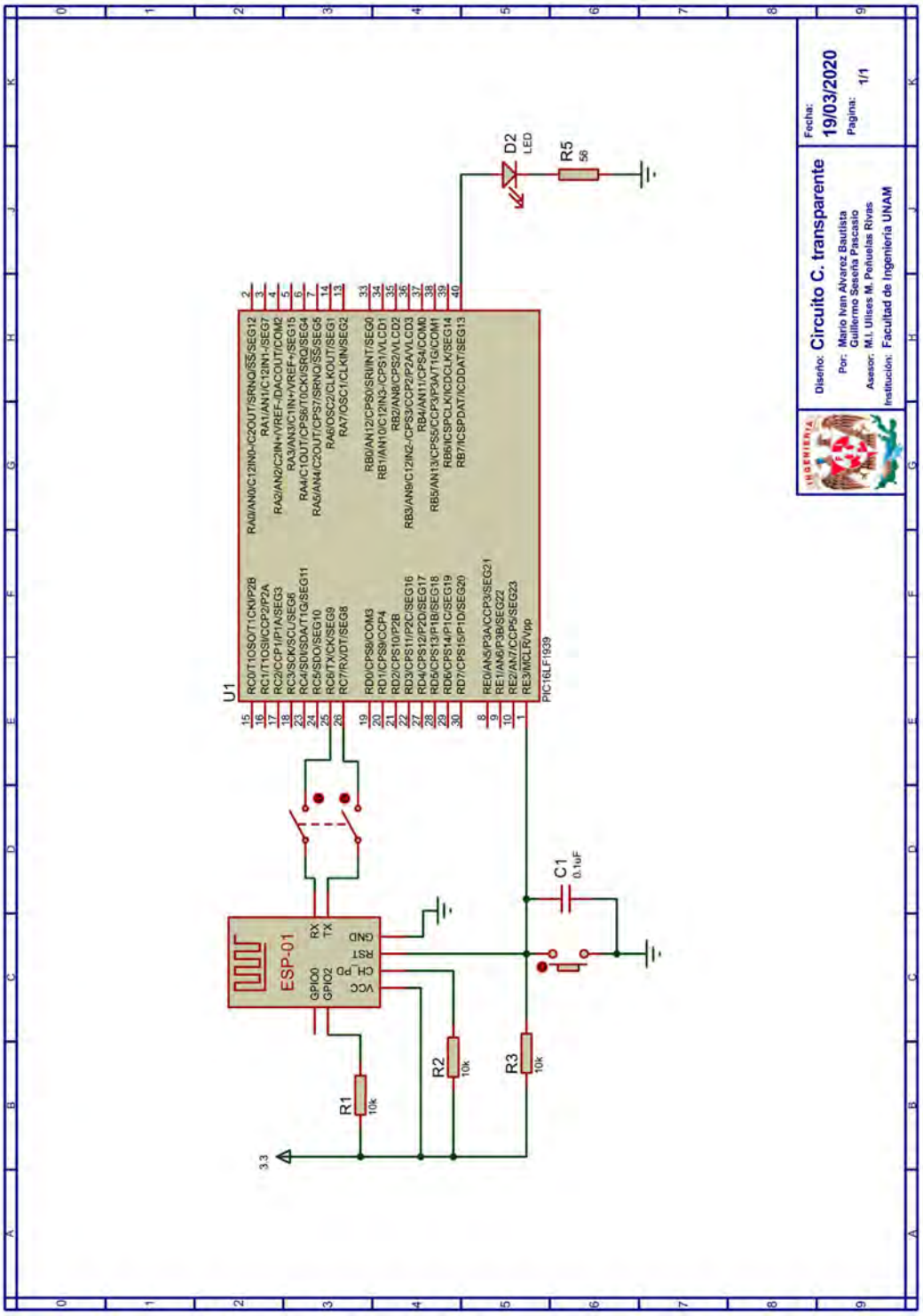


Figura P9.6 *Circuito de la práctica 9.*

Referencias

Esta aplicación toma como base los comandos AT creados para este fin, los cuales pueden ser consultados en:

- Espressif Systems, ESP32 AT Instruction Set and Examples. 2019.
- Espressif Systems, ESP8266 AT Instruction Set Version 3.0.2. 2019.

Práctica 10-A Cliente SSL

Introducción

Capa de sockets seguros (*Secure Sockets Layer*, SSL) y su sucesor, Seguridad de capa de transporte (*Transport Layer Security,* TLS) son dos protocolos que proveen seguridad a los datos generados desde la capa de aplicación, los cuales son encapsulados en paquetes SSL o TLS y transmitidos por TCP (un protocolo de transporte confiable) para proveer una conexión segura y cifrada a capas superiores [1]. Uno de sus principales usos se da en HTTPS (Protocolo seguro de transferencia de hipertexto, *Secure Hypertext Transfer Protocol*), en el cual HTTP opera con SSL o TLS [2].

Estos protocolos están diseñados para proporcionar tres servicios:

- Confidencialidad, todos los datos enviados son cifrados.
- Autenticación del servidor o ambas partes, mediante un certificado digital firmado por alguna autoridad de certificación. La autenticación puede ser opcional.
- Integridad del mensaje, todos los mensajes cifrados incluyen un resumen del mensaje para comprobar si han sido manipulados o falsificados [1-3].

Cada protocolo se subdivide en diferentes capas. Como TLS y SSL v3, que es la última versión del protocolo SSL, son muy similares [2] y como tanto el SoC ESP8266 como los ESP32 utilizan principalmente TLS para establecer las conexiones seguras, se describirán las capas que componen a TLS, las cuales se pueden observar en la figura P10-A.1

TLS está compuesto de dos capas, la capa inferior que se ubica sobre la capa de transporte y está constituida por el

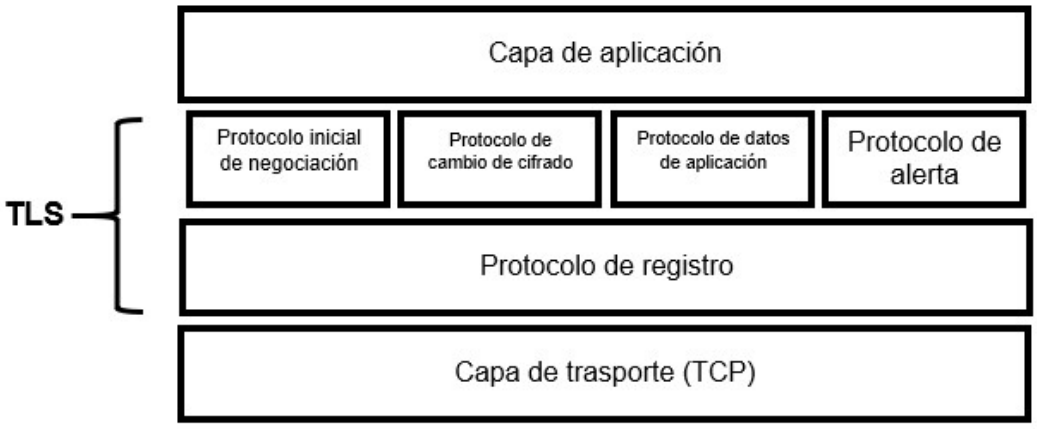


Figura P10-A.1 *Composición de TLS. Modificado de [2].*

protocolo de registro TLS (*Record Protocol*), el cual "toma los mensajes a ser transmitidos, fragmenta los datos en bloques manejables, opcionalmente comprime los datos, aplica una MAC (*Message Authentication Code*), cifra y transmite el resultado"[4]. Cuando en esta misma capa se recibe un mensaje, se invierte totalmente el proceso para recuperar la información, es decir," los datos recibidos se descifran, verifican, descomprimen, se vuelven a ensamblar"[4] y son transmitidos a los protocolos de nivel superior.

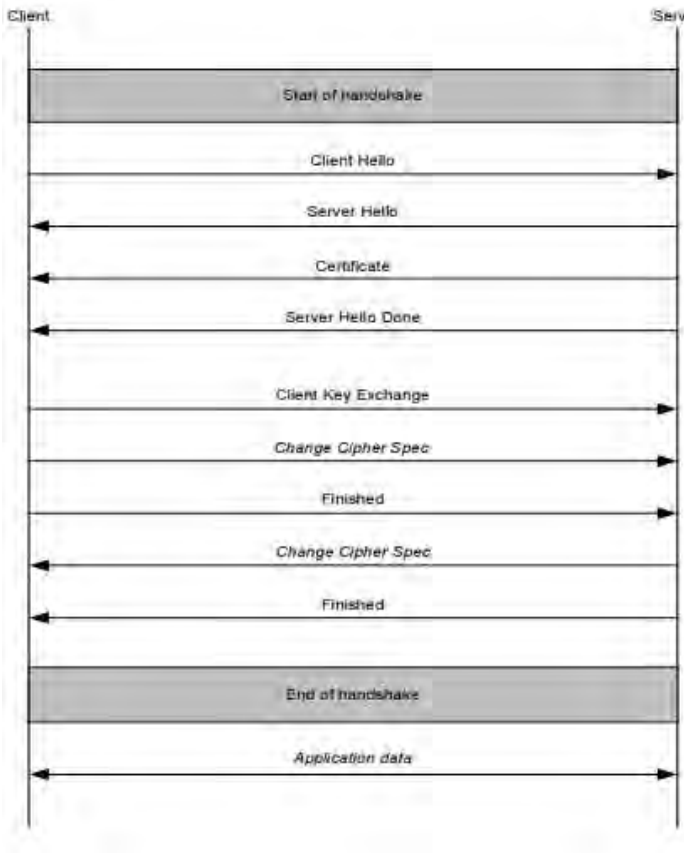


Figura P10-A.2 *Secuencia de mensajes durante handshake[2].*

La capa superior de TLS está constituida por una serie de protocolos:

- Protocolo de negociación inicial TLS (*Handshake Protocol*), utilizado para establecer el algoritmo de cifrado, realizar la autenticación de una o ambas partes, e intercambiar claves [2, 4].
- Protocolo de alerta, utilizado para señalar que un error o problema ha ocurrido [2].
- Protocolo de cambio de especificaciones de cifrado, utilizado para señalar las transiciones en las estrategias de cifrado [2].
- Protocolo de aplicación, todos los datos enviados utilizan este protocolo una vez que la etapa del *handshake* ha terminado. Estos datos son transparentes para la aplicación [2, 4].

Durante el protocolo de negociación inicial (*handshake*), se intercambian varios mensajes entre los *peers* que desean establecer una conexión segura (ver figura P10-A.2) y es en estos mensajes donde se verifica la identidad de dichos *peer* mediante certificados [2]. En las computadoras este proceso se realiza comparando el certificado que se recibe del *peer* con una lista de certificados raíz almacenados localmente[5], sin embargo, en la mayoría de los módulos WiFi que

incorporan el SoC ESP8266 o los ESP32 esto no es posible debió a la cantidad de memoria que se requiere para almacenarlos en la memoria flash de éstos. Es por esto, que el cliente SSL está configurado para deshabilitar la autenticación del certificado del servidor por defecto, sin embargo, con las modificaciones realizadas para los SoC ESP8266 ahora es posible utilizar el *thumbprint (huella)* de un certificado, también llamado *fingerprint,* para verificar la identidad del servidor y decidir si continuar o abortar la conexión con éste [5, 6]. Todo esto en caso de que no sea posible grabar un certificado en el módulo WiFi.

Los módulos que incorporan el SoC ESP8266 o los ESP32 soportan la versión de TLS 1.0, 1.1 y 1.2 tanto en las versiones ESP-IDF y SDK NONOS del *firmware*, aunque éste último en su versión v3.0.3 también soporta SSL3.0

A pesar de todos los protocolos que se manejan en SSL, la biblioteca ESP extrae y envía mensajes directamente de como el módulo lo obtiene de la capa de transporte sin ningún formato dejando al cliente la libertad para definir las reglas del intercambio de información. Es por tanto que la biblioteca ESP permite realizar las siguientes acciones:

- Establecer conexión con uno o más servidores SSL, especificando el puerto remoto de cada uno de los servidores.
- Establecer un *buffer* compartido para los clientes (tomar en cuenta que los paquetes recibidos son guardados como *strings* por ello al mensaje recibido se le inserta el terminador \0, si se requiere recibir información binaria el usuario tiene que establecer una forma para reconocer en donde termina cada mensaje recibido, por ejemplo, establecer que los dos primeros bytes indiquen la longitud del paquete recibido).
- Comparar el *buffer* con algún *string*.
- Enviar *strings* alojados en la ROM del PIC (máximo 2048 bytes).
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 2048 bytes).
- Cerrar la conexión de cada cliente.
- Habilitar el uso de *fingerprint* como auxiliar de autentificación en SDK NONOS.

Objetivo

Mostrar cómo crear un cliente SSL para establecer una conexión segura con un servidor.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT,

por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

STATION y E_SSL_TCP_UDP_CLIENT_UNICON.

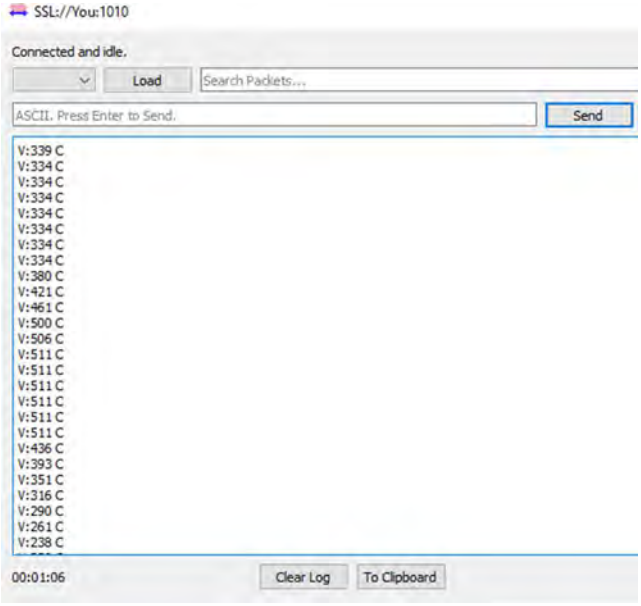


Figura P10-A.3 *Recepción de valores del potenciómetro.*

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 potenciómetro de 10 kΩ
- 1 resistor de 220 Ω
- 1 resistor de 510 Ω

Descripción

Se deberá conectar una computadora y el módulo WiFi a un mismo punto de acceso. En la computadora se deberá ejecutar el programa *packet sender* y crear un servidor SSL en el puerto 1010. El módulo deberá conectarse a dicho servidor y enviar cada 5 segundos un mensaje que contenga el valor obtenido de un potenciómetro que estará conectado al uC PIC.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P10-A.4, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP.
- Dirección IP y puerto del servidor SSL.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P10-A.5.

Resultados

Se deberá crear un servidor SSL en la computadora con el programa *packet sender* antes de que el microcontrolador ejecute su programa, consultar https://packetsender.com/documentation cuya dirección IP corresponde a la asignada a la computadora en que se ejecuta el programa (aparece en la parte superior de la interfaz del mismo programa *packet sender*).

Posteriormente se deberá esperar a que el módulo establezca conexión el servidor, momento en el cual se abrirá una ventana en el programa *packet sender* (ver figura P10-A.3) y se comenzará a recibir los valores obtenidos del potenciómetro.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/37URTgZ

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

long valor;
char buffer[10]={0};

void main()
{
    setup_adc(adc_clock_div_64);
    delay_us(10);
    setup_adc_ports(sAN10);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","SSID_ESP","123456789");
    /*Creación de un cliente SSL uniconexión que se conectará al servidor SSL
     creado por el programa packet sender (Dirección IP de la computadora)*/
    create_ssl_pclient("192.168.43.224",1010);
    while(1)
    {
        //Obtención del valor
        set_adc_channel(10);
        valor=(read_adc()>>1);
        //Envío del valor al servidor SSL
        sprintf(buffer,"V:%Ld C\r\n",valor);
        send_ssl(buffer);
        delay_ms(1000);
    }
}
```

Figura P10-A.4 *Código de la práctica 10-A.*

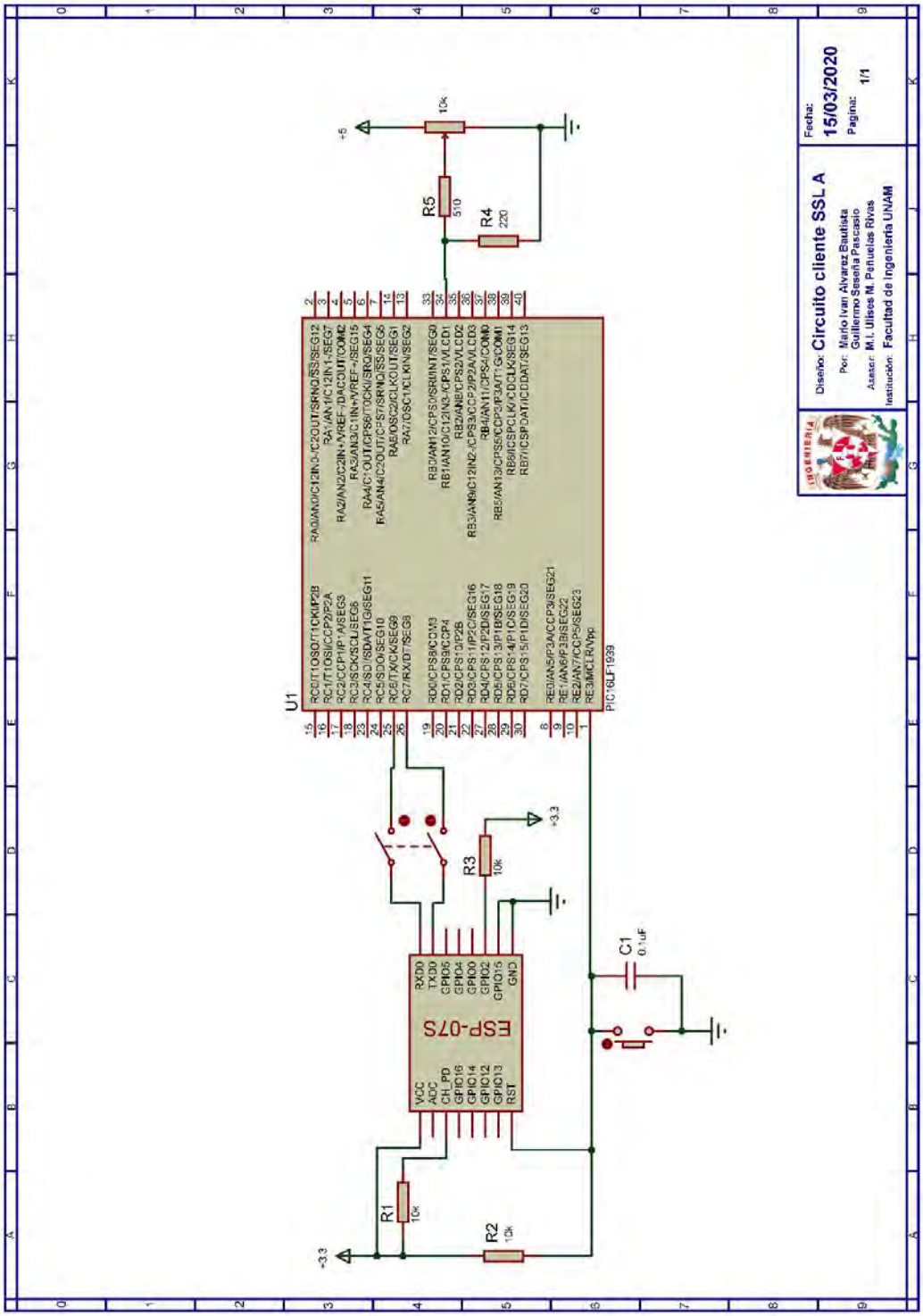


Figura P10-A.5 *Circuito de la práctica 10-A.*

Notas

- Cuando se graba un certificado en el módulo WiFi para compararlo con los enviados por un servidor durante el proceso de conexión, se debe habilitar SNTP para verificar que el certificado aún es válido.
- En el caso del *firmware* basado en el SDK NONOS se recomienda utilizar la versión v3.0.3, ya que es la que tiene habilitada más herramientas (consultar Apéndice C), haciendo que sea compatible con servidores que en la versión v2.2.1 no es posible conectarse.

Referencias

1. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
2. Werstén, B., *Implementing the Transport Layer Security Protocol for Embedded Systems.* 2007.
3. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
4. IETF TLS working group. *The Transport Layer Security (TLS) Protocol Version 1.2*. 2008 [Citado 2020 Marzo]; Disponible en:https://tools.ietf.org/html/rfc5246#page-65.
5. Grokhotkov, I. *Client Secure*. Docs 2017 [Citado 2020 Marzo]; Disponible en:https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/client-secure-examples.html.
6. Zaverucha, G. and D. Shumow, *Are Certificate Thumbprints Unique?* 2019.

Práctica 10-B Cliente SSL: uso de *fingerprint*

Introducción

Capa de sockets seguros (*Secure Sockets Layer*, SSL) y su sucesor, Seguridad de capa de transporte (*Transport Layer Security,* TLS) son dos protocolos que proveen seguridad a los datos generados desde la capa de aplicación, los cuales son encapsulados en paquetes SSL o TLS y transmitidos por TCP (un protocolo de transporte confiable) para proveer una conexión segura y cifrada a capas superiores[1]. Uno de sus principales usos se da en HTTPS (Protocolo seguro de transferencia de hipertexto, *Secure Hypertext Transfer Protocol*), en el cual HTTP opera con SSL o TLS[2].

Estos protocolos están diseñados para proporcionar tres servicios:

- Confidencialidad, todos los datos enviados son cifrados.
- Autenticación del servidor o ambas partes, mediante un certificado digital firmado por alguna autoridad de certificación. La autenticación puede ser opcional.
- Integridad del mensaje, todos los mensajes cifrados incluyen un resumen del mensaje para comprobar si han sido manipulados o falsificados[1-3].

Cada protocolo se subdivide en diferentes capas. Como TLS y SSL v3, que es la última versión del protocolo SSL, son muy similares [2] y como el SoC ESP8266 como los ESP32 utilizan principalmente TLS para establecer la conexión segura que permiten, se describirán las capas que componen a TLS, las cuales se pueden observar en la figura P10-B.1.

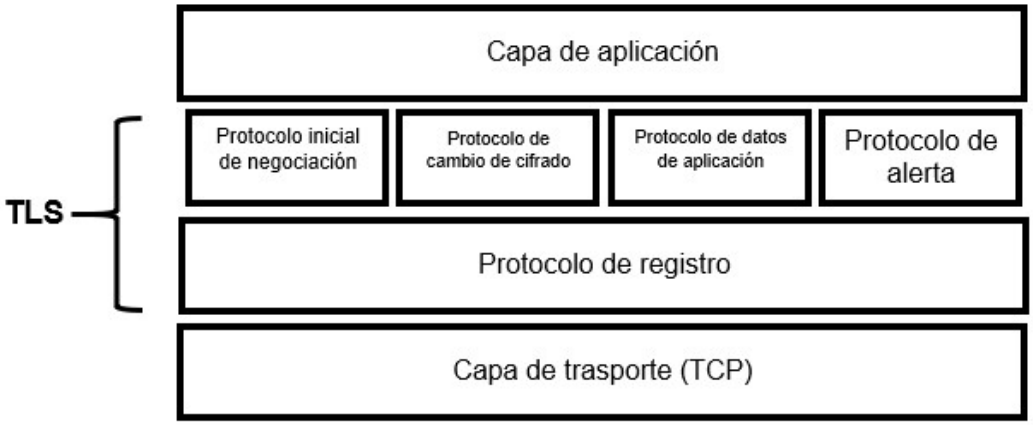


Figura P10-B.1 *Composición de TLS. Modificado de [2].*

TLS está compuesto de dos capas, la capa inferior que se ubica sobre la capa de transporte está constituida por el protocolo de registro TLS (*Record Protocol*), el cual "toma los mensajes a ser transmitidos, fragmenta los datos en bloques manejables, opcionalmente comprime los datos, aplica una MAC (*Message Authentication Code*), encripta y transmite el resultado"[4]. Cuando en esta misma capa se recibe un mensaje, se invierte totalmente el proceso para recuperar la información, es decir," los datos recibidos se descifran, verifican, descomprimen, se vuelven a ensamblar"[4] y son transmitidos a los protocolos de nivel superior.

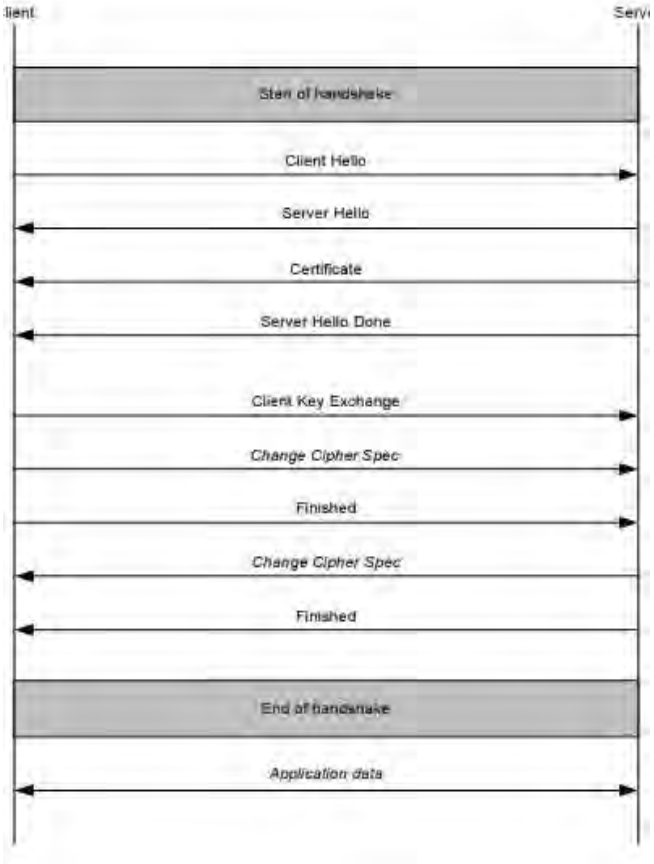


Figura P10-B.2 *Secuencia de mensajes durante handshake[2].*

La capa superior de TLS está constituida por una serie de protocolos:

- Protocolo de negociación inicial TLS (*Handshake Protocol*), utilizado para establecer el algoritmo criptográfico, realizar la autenticación de una o ambas partes, e intercambiar claves [2, 4].
- Protocolo de alerta, utilizado para señalar que un error o problema ha ocurrido [2].
- Protocolo de cambio de especificaciones de cifrado, utilizado para señalar las transiciones en las estrategias de cifrado[2].
- Protocolo de aplicación, todos los datos enviados utilizan este protocolo una vez que la etapa del *handshake* ha terminado. Estos datos son transparentes para la aplicación [2, 4].

Durante el protocolo de negociación inicial (*handshake*), se intercambian varios mensajes entre los *peers* que desean establecer una conexión segura (ver figura P10-B.2) y es en estos mensajes donde se verifica la identidad de dichos *peer* mediante certificados[2].

En las computadoras este proceso se realiza comparando el certificado que se

recibe del *peer* con una lista de certificados raíz almacenados localmente[5], sin embargo, en la mayoría de los módulos WiFi que incorporan el SoC ESP8266 o los ESP32 esto no es posible debió a la cantidad de memoria que se requiere para almacenarlos en la memoria flash de éstos. Es por esto, que el cliente SSL está configurado para deshabilitar la autenticación del certificado del servidor, sin embargo, con las modificaciones realizadas para los SoC ESP8266 ahora es posible utilizar el *thumbprint* (huella) de un certificado, también llamado *fingerprint,* para verificar la identidad del servidor y decidir si continuar o abortar la conexión con éste[5, 6]. Todo esto en caso de que no sea posible grabar un certificado en el módulo WiFi.

El *thumbprint* es un resumen del certificado obtenido tras aplicar una función hash a todos los datos del certificado y su firma, los cuales deben estar codificados en formato DER, este resumen es utilizado como identificador único del certificado y, por tanto, permite tomar decisiones confiables comparando uno establecido previamente con otro que se calcula durante el *handshake* [5, 6]. En este caso, los módulos que incorporan el SoC ESP8266 pueden obtener el *thumbprint* durante el *handshake* mediante los métodos MD5, SHA-1 y SHA-256. Aunque el uso *thumbprint* no es totalmente confiable, es una alternativa para añadir un grado seguridad que contrasta con simplemente deshabilitar la autenticación del *peer* o servidor con el que se quiere conectar de manera segura. Como el *thumbprint* se obtiene a partir del certificado, éste cambia cuando el certificado es renovado.

Para obtener el *thumbprint* de un servidor web se debe utilizar un navegador web, en este caso, se describe el procedimiento utilizando el navegador Google Chrome.

- Hacer clic sobre el candado que aparece en la barra de direcciones (ver figura P10-B.3)

Figura P10-B.3 *Barra de direcciones.*

- Hacer clic en certificado.

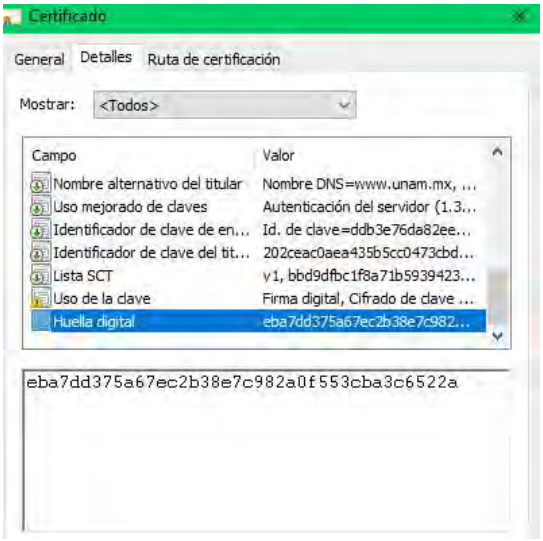


Figura P10-B.4 *Fingerprint* de un certificado.

- Hacer clic en la pestaña detalles.
- Desplazarse hasta el *thumbprint* y hacer clic sobre éste (ver figura P10-B.4).

Así mismo, los módulos que incorporan el SoC ESP8266 o los ESP32 soportan la versión de TLS 1.0, 1.1 y 1.2 tanto en las versiones ESP-IDF y SDK NONOS del *firmware*, aunque éste último en su versión v3.0.3 también soporta SSL3.0

La biblioteca ESP extrae y envía mensajes directamente de como el módulo lo obtiene de la capa de transporte sin ningún formato dejando al cliente la libertad para definir las reglas del intercambio de información. Es por tanto que la biblioteca ESP permite realizar las siguientes acciones:

- Establecer conexión con uno o más servidores SSL, especificando el puerto remoto de cada uno de los servidores
- Establecer un *buffer* compartido para los clientes (tomar en cuenta que los paquetes recibidos son guardados como *strings* por ello al mensaje recibido se le inserta el terminador \0, si se requiere recibir información binaria el usuario tiene que establecer una forma para reconocer en donde termina cada mensaje recibido, por ejemplo, establecer que los dos primeros bytes indiquen la longitud del paquete recibido)
- Comparar el *buffer* con algún string
- Enviar *strings* alojados en la ROM del PIC (máximo 2048 bytes)
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 2048 bytes)
- Cerrar la conexión de cada cliente
- Habilitar el uso de *fingerprint* como auxiliar de autentificación en SDK NONOS.

Objetivo

Mostrar cómo utilizar el *fingerprint* para autenticar el servidor con el cual se quiere establecer una conexión segura.

***Firmware* de comandos AT**

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3 o en el SDK NONOS 2.2.1, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL

- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF

Habilitación de aplicaciones

STATION y E_SSL_TCP_UDP_CLIENT_UNICON deben ser TRUE en esp.h

Descripción

Se deberá obtener el *fingerprint* de api.github.com.

Posteriormente, se conectará el módulo WiFi a un AP con acceso a internet, habilitará el uso de *fingerprint* estableciendo uno inventado por el propio usuario de la misma longitud del original y tratará de establecer una conexión segura a api.github.com en el puerto 443.

Se deberá imprimir en una terminal si la conexión se pudo realizar o no. Después, establecerá el *fingerprint* original y tratará de establecer nuevamente una conexión segura con el mismo servidor y deberá imprimir en la terminal si la conexión se pudo realizar o no.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P10-B.7, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet.
- *Fingerprint* del servidor api.github.com y uno inventado.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P10-B.8.

Resultados

Previo a la ejecución del programa se deberá obtener el *fingerprint* de api.github.com (ver figura P10-B.5).

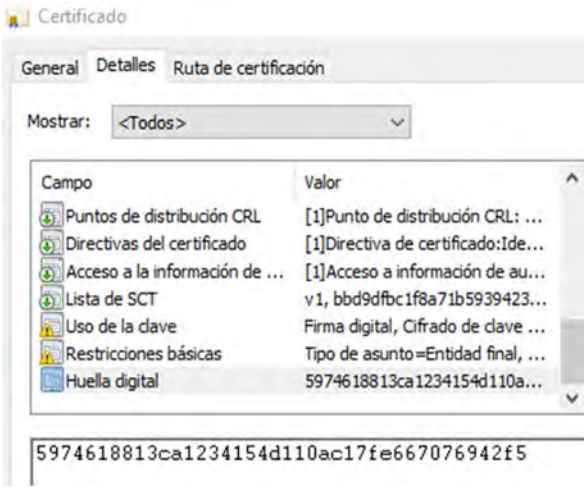


Figura P10-B.5 *Fingerprint de api.github.com*

Al iniciar la ejecución del programa del microcontrolador PIC, el primer mensaje que se mostrará en la terminal corresponderá al que notifica del fracaso de la conexión con el servidor (ver figura P10-B.6.

```
Modulo esp listo para su uso\r\n
Conexion con fingerprint incorrecto\r\n
Conexion fallo\r\n
```

Figura P10-B.6 *Conexión fallida con el servidor.*

```
#include <16lf1939.h>
#include "esp.h"

int err=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","SSID","password");
    //Se establece el fingerprint inventado por el usuario
    enable_fingerprint_ssl("0174618813ca1234154d110ac17fe667076942f5");
    fprintf(PIC,"Conexión con fingerprint incorrecto\r\n");
    /*Creación de un cliente SSL uniconexión que se conectará al servidor SSL*/
    err=create_ssl_client("api.github.com",443);
    if(err==0)
        fprintf(PIC,"Conexión exitosa\r\n");
    else
        fprintf(PIC,"Conexión falló\r\n");
    //Se establece el fingerprint del servidor
    enable_fingerprint_ssl("5974618813ca1234154d110ac17fe667076942f5");
    fprintf(PIC,"Conexión  con fingerprint correcto\r\n");
    /*Creación de un cliente SSL uniconexión que se conectará al servidor SSL*/
    err=create_ssl_client("api.github.com",443);
    if(err==0)
        fprintf(PIC,"Conexión exitosa\r\n");
    else
        fprintf(PIC,"Conexión falló\r\n");
    while(1)
    {
    }
}
```

Figura P10-B.7 *Código de la práctica 10-B.*

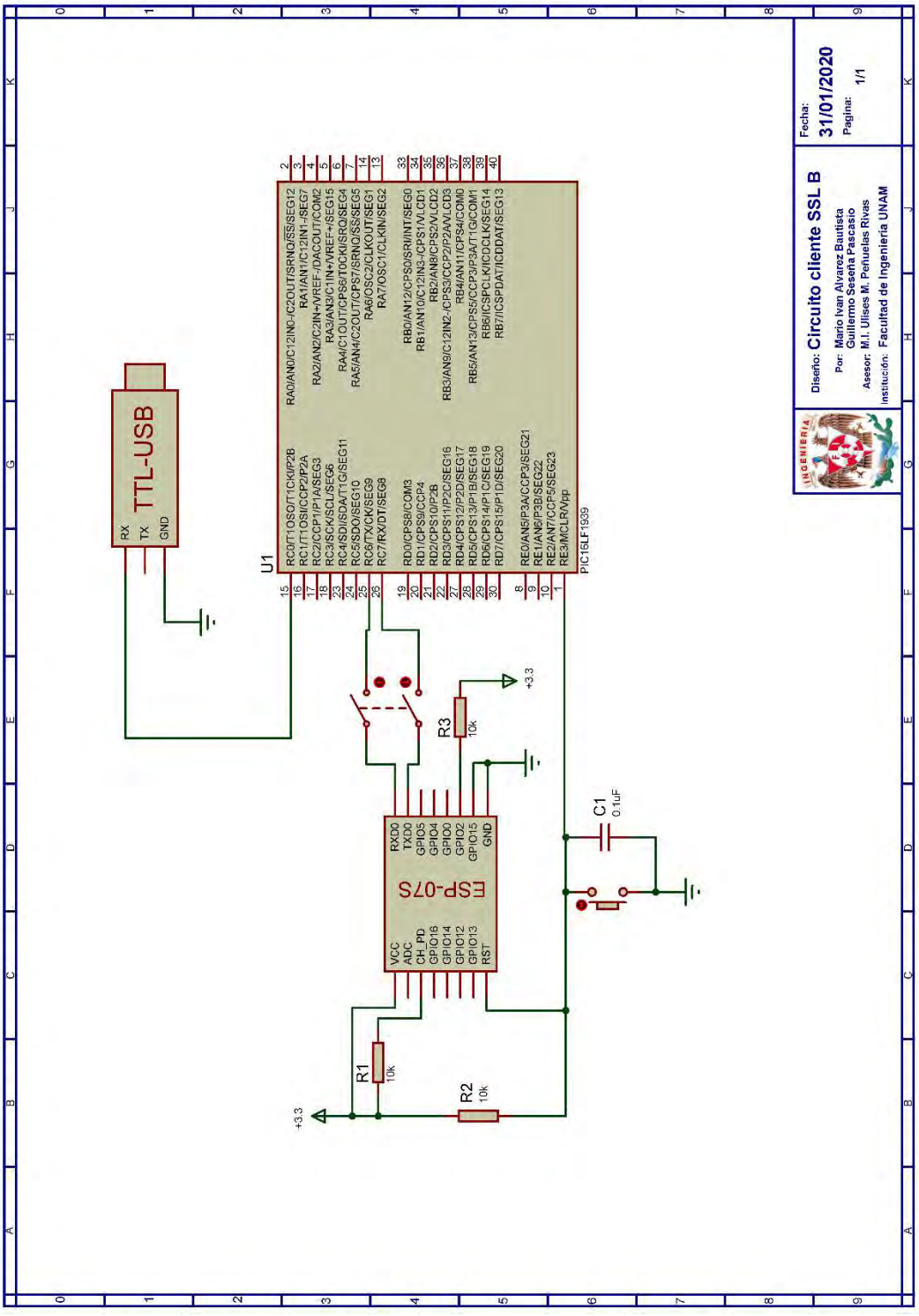


Figura P10-B.8 *Circuito de práctica 10-B.*

Posteriormente, internamente se establecerá el *fingerprint* correcto y en la terminal se deberá mostrar el mensaje de que se ha conectado exitosamente al servidor (ver figura P10-B.9).

```
Conexion con fingerprint correcto
Conexion exitosa
```

Figura P10-B.9 *Conexión exitosa con el servidor.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3hLJ6SH

Notas

- Algunos antivirus emiten sus propios certificados para páginas web por lo que el *fingerprint* podría no coincidir con el certificado emitido por una CA. Para evitar este problema se recomienda consultar el *fingerprint* de una página a través de un teléfono inteligente y verificar que este coincide con el obtenido al usar una computadora.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- En el caso del *firmware* basado en el SDK NONOS se recomienda utilizar la versión v3.0.3, ya que es la que tiene habilitada más herramientas (consultar Apéndice C), haciendo que sea compatible con servidores que en la versión v2.2.1 no es posible conectarse.

Referencias

1. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.
2. Werstén, B., *Implementing the Transport Layer Security Protocol for Embedded Systems.* 2007.
3. Kurose, J.F. and K.W. Ross, *Redes de computadoras. Un enfoque decendente.* Quinta ed. 2010, España: Pearson eduaction.
4. IETF TLS working group. *The Transport Layer Security (TLS) Protocol Version 1.2*. 2008 [Citado 2020 Marzo]; Disponible en:https://tools.ietf.org/html/rfc5246#page-65.
5. Grokhotkov, I. *Client Secure*. Docs 2017 [Citado 2020 Marzo]; Disponible en:https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/client-secure-examples.html.
6. Zaverucha, G. and D. Shumow, *Are Certificate Thumbprints Unique?* 2019.

Práctica 11-A RSSI: estimación de distancia

Introducción

El Indicador de intensidad de señal recibida (*Received Signal Strength Indicator,* RSSI) es una medida del nivel de potencia que recibe la antena de un dispositivo, es decir la intensidad de la señal (amplitud). Esta medida está expresada como un número negativo en *dBm* y puede ser obtenida al procesar los cambios en el nivel de voltaje del circuito indicador de intensidad de señal recibida del receptor. Entre más cercano esté el valor RSSI de cero, la señal será más fuerte y mientras más alejado esté la señal será más débil [1-3].

RSSI es una medida arbitraria, ya que "la asignación de los valores de RSSI a la potencia recibida real depende de la implementación" de cada fabricante de las tarjetas inalámbricas. Esto da lugar a la aparición de diferentes rangos de valores y diferentes esquemas de numeración [3]. Es por esto, que cuando se utilizan los valores RSSI de un producto, éste no puede ser remplazado indistintamente por otro, a menos que incorpore una tarjeta inalámbrica del mismo fabricante.

Generalmente el RSSI es utilizado en los módulos WiFi para adaptar dinámicamente su modulación y esquemas de codificación para lograr la velocidad de datos óptima [4]. Aunque también han surgido investigaciones para a partir de este mismo parámetro desarrollar diversas implementaciones, entre las que destacan:

- Estimación de distancia, la cual se puede determinar estableciendo una relación entre la distancia y el valor RSSI obtenido, ya sea mediante un modelo de propagación de la señal o mediante una técnica de ajuste de curva para obtener una ecuación que estime la distancia [5, 6].
- Determinación de posición en interiores mediante métodos como trilateración, en el cual se determina la posición de un objeto a partir de la intersección de círculos, cuyos radios son la estimación de la distancia del objeto a al menos tres puntos de acceso con ubicaciones ya conocidas [5].
- Seguimiento en interiores [7].
- Distinguir entre diferentes espesores de un material en [7].
- Detector de presencia de personas, tal y como se muestra en [7] y [8].

Si bien, existen un gran número de tecnologías inalámbricas de las cuales se pueden obtener valores RSSI, se suele ocupar principalmente la tecnología WiFi ya que su infraestructura ya está presente en un gran número de dispositivos y lugares. Esto hace que no se requiera adquirir, o en su caso muy

poco, equipo adicional para el desarrollo de las aplicaciones antes mencionadas [5].

La señal WiFi, y por tanto los valores RSSI obtenidos, se ven afectados principalmente por efectos de desvanecimiento y multitrayectoria que son causados por obstrucciones físicas (la atenuación depende del material, geometría y espesor del objeto), interferencias de otras redes WiFi o dispositivos electromagnéticos, la distancia, el tipo de antenas y su orientación entre otros [2, 4, 5]. Estos efectos provocan que para la utilización de los valores RSSI se tengan que recabar una gran cantidad de lecturas para obtener una mayor precisión y que para minimizar la variabilidad debida a estos efectos se utilicen filtros digitales como promedios móviles simple, media alfa recortada o Kalman. La elección del filtro depende de las características de la aplicación y puede ser uno o una combinación de éstos [2].

Considerando las aplicaciones antes mencionadas, en la biblioteca ESP se ha desarrollado funciones para cualquiera de los SoC para:

- Obtención de un valor RSSI.
- Obtención del promedio de hasta 600 lecturas RSSI, en intervalos de tiempo entre 5 y hasta 65535 *ms.*
- Obtención del valor máximo, mínimo y promedio tras 99 lecturas RSSI, ya sea con desviación estándar o Cuartiles principalmente.
- Obtención de valores RSSI de hasta cinco diferentes puntos de acceso. Ya sea un valor RSSI o el promedio de hasta 10 lecturas para cada AP.
- Establecer la obtención continua de valores RSSI, aplicando o no promedios móviles y verificando que se encuentren dentro de un rango establecido.

Objetivo

Ilustrar que el conjunto uC PIC y módulo WiFi puede ser utilizado para el desarrollo de trabajos que involucren estimación de distancia mediante valores RSSI.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

- STATION y RSSI en el módulo que funcione cono estación.
- ACCESS_POINT y RSSI en el módulo que funcione como punto de acceso.

Materiales

- 1 computadora
- 2 microcontrolador PIC
- 2 módulo WiFi
- 2 interruptor DPST
- 1 adaptador USB-TTL
- 9 resistores de 10 kΩ
- 3 botones pulsadores n.o.
- 2 condensador de 0.1 µF

Descripción

Esta práctica se realizará en dos partes, la primera consistirá en la obtención de los valores RSSI que permitan desarrollar un modelo para estimar la distancia, mientras que la segunda se hará uso de dicho modelo para obtener una estimación de distancia.

I Obtención de valores RSSI

Para la primera parte, se conectará el módulo WiFi al punto de acceso creado por el primer módulo y se obtendrán valores promedios RSSI (40 lecturas cada 5 segundos para cada valor promedio) en al menos 5 diferentes distancias respecto al AP y serán impresos en una terminal. Estos valores se registrarán en una hoja de cálculo del programa Excel, formando una tabla distancia vs RSSI, se graficarán los datos obtenidos y se aplicará una regresión polinómica que se ajuste a la curva obtenida para obtener un modelo matemático.

II Estimación de distancia

Para la segunda parte, el módulo WiFi estará conectado al mismo punto de acceso y realizará estimaciones de la distancia a la que se encuentra del punto de acceso utilizando el modelo matemático previamente determinado y serán hechas cada vez que se presione un botón. Estas estimaciones serán mostradas en una terminal.

Código

El código del módulo que funcionará como punto de acceso se muestra en la figura P11-A.4, en el que se debe ingresar nombre y contraseña del AP que se creará.

El código que deberá ejecutar el microcontrolador PIC para obtener los valores RSSI a ciertas distancias se muestra en la figura P11-A.5, mientras que el código para obtener las estimaciones de distancia se muestra en la figura P11-A.6. En ambos se deben ingresar nombre y contraseña del AP del que se estimará la distancia (creado con el otro módulo).

Circuito

El circuito del módulo que funcionará como estación se muestra en la figura P11-A.7, mientras que el circuito del módulo que funcionará como punto de acceso es el mismo que el de la figura P11-A.7 sin la inclusión del adaptador serial ni el botón conectado a A1.

Resultados

La primera parte de la práctica consiste en obtener el valor RSSI promedio a cierta distancia del AP. Este promedio es obtenido cada vez que se presione un botón. Los valores obtenidos son registrados en una hoja de cálculo.

En este caso, se han obtenido los RSSI para las distancias que se muestran en la figura P11-A.1.

Distancia [cm]	RSSI
20	-21.1
25	-21
30	-22.9
35	-24.52
40	-27.57
45	-29.02
50	-31.87
55	-30.8

Figura P11-A.1 *Mediciones RSSI obtenidas.*

De estos valores obtenidos se han seleccionado aquellos puntos que muestra un comportamiento ideal (de 25 a 45 cm), se han graficado los puntos y se ha aplicado una regresión polinómica de tercer grado para obtener un modelo matemático que relaciones la distancia con el valor RSSI (ver figura P11-A.2)

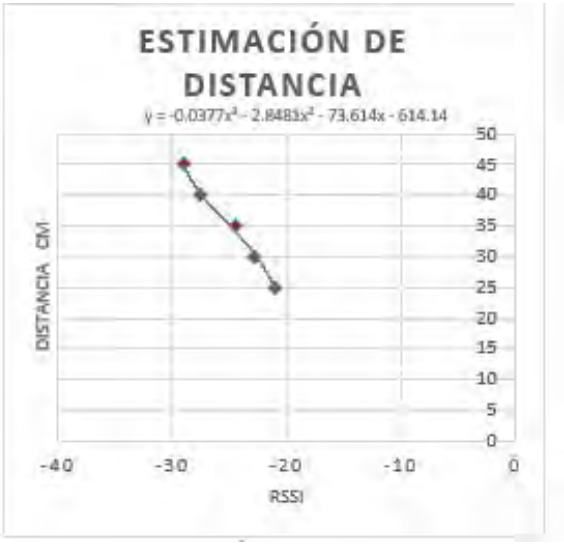


Figura P11-A.2 Gráfica de valores RSSI.

Posteriormente, el programa del microcontrolador PIC será cambiado por uno que contenga el modelo matemático para realizar las estimaciones de distancia cada vez que se presiona un botón asociado al microcontrolador y las imprima en una terminal, tal y como se muestra en la figura P11-A.3.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2Nf3qy6

```
Modulo esp listo para su uso\r\n
RSSI:-19.10\r\n
Distancia:15.56 cm\r\n
RSSI:-22.12\r\n
Distancia:28.69 cm\r\n
RSSI:-26.30\r\n
Distancia:37.72 cm\r\n
RSSI:-25.67\r\n
Distancia:36.49 cm\r\n
RSSI:-33.15\r\n
Distancia:69.70 cm\r\n
RSSI:-31.25\r\n
Distancia:55.46 cm\r\n
RSSI:-32.60\r\n
Distancia:64.98 cm\r\n
```

Figura P11-A.3 Estimaciones de distancia.

Notas

- Para mejorar los resultados se recomienda utilizar una antena dipolar con conexión IPEX para el módulo ESP-07S.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- Este tipo de estimación presenta mejores resultados en el orden de metros.
- El tiempo entre cada lectura RSSI también puede afectar la precisión de la estimación.
- Se recomienda usar el programa Hterm para simular la terminal.

Práctica propuesta

En un área de un metro por un metro, se colocarán tres módulos que funcionarán como puntos de acceso. Un cuarto módulo será introducido en esta misma área, el cual funcionará como estación y obtendrá valores RSSI de los tres puntos de acceso cada 30 segundos. Así mismo, este módulo estará conectado a uno de estos puntos de acceso, al cual también estará conectado una computadora que estará ejecutando el programa *Packet Sender.* Este programa habrá creado un servidor TCP para recibir los valores RSSI. En esta misma computadora se ejecutará un programa matemático como Maple o Matlab para determinar por trilateración la ubicación del módulo que funciona como estación con los valores RSSI obtenidos. Se deben utilizar las funciones *setup_ssid_rsssi, start_ssid_rssi* y *get_ssid_rssi*

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Creación de un punto de acceso en el canal 2
    begin_ap(2,"RSSI_AP","123456789",2);
    //Selección de 802.11g
    wifi_mode(2);
    //Se establece el máximo valor de RF TX
    wifi_tx_power(40);
    while(1)
    {
    }
}
```

Figura P11-A.4 *Código de la práctica 11-A, módulo como punto de acceso.*

```
#include <16lf1939.h>
#include "esp.h"

short button_state=false; // variable de control
float rssi=0;

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","RSSI_AP","123456789");
    /*Función que cambia el nivel para poder utilizar las funciones RSSI*/
    up_level2_to_level3();
    while(1)
    {
        if(1==input_state(PIN_A1) && button_state==false)
        {
            while(1==input_state(PIN_A1))
            {
                //Se mantiene hasta soltar el botón
            }
            /*Inicia el cálculo del valor promedio RSSI de 100 lecturas
            cada 400 ms*/
            start_mean_rssi(100,400);
            button_state=true;
        }
        //¿Ya terminó el cálculo del valor promedio RSSI?
        if(1==get_mean_rssi(&rssi))
        {
            fprintf(PIC,"Valor RSSI:%f\r\n",rssi);
            button_state=false;
            rssi=0;
        }
    }
}
```

Figura P11-A.5 *Código de la práctica 11-A para la parte 1, módulo como estación.*

```
#include <16lf1939.h>
#include "esp.h"

short button_state=false; // variable de control para la petición de lecturas rssi
float rssi=0, distancia=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","RSSI_AP","123456789");
    //Sube de nivel para usar las funciones RSSI
    up_level2_to_level3();
    while(1)
    {
        if(1==input_state(PIN_A1) && button_state==false)
        {
            while(1==input_state(PIN_A1))
            {
                //Se mantiene hasta soltar el botón
            }
            /*Inicia el cálculo del valor promedio RSSI de 100 lecturas cada 20 ms*/
            start_mean_rssi(40,500);
            button_state=true;
        }
        //¿Ya terminó el cálculo del valor promedio RSSI?
        if(1==get_mean_rssi(&rssi))
        {
            fprintf(PIC,"RSSI:%.2f\r\n",rssi);
            //Estimacion de la distancia
            distancia=-0.0377*rssi*rssi*rssi-2.8481*rssi*rssi-73.614*rssi- 614.14;
            fprintf(PIC,"Distancia:%.2f cm\r\n",distancia);
            rssi=0;
            button_state=false;
        }
    }
}
```

Figura P11-A.6 *Código de la práctica 11-A para la parte 2, módulo como estación.*

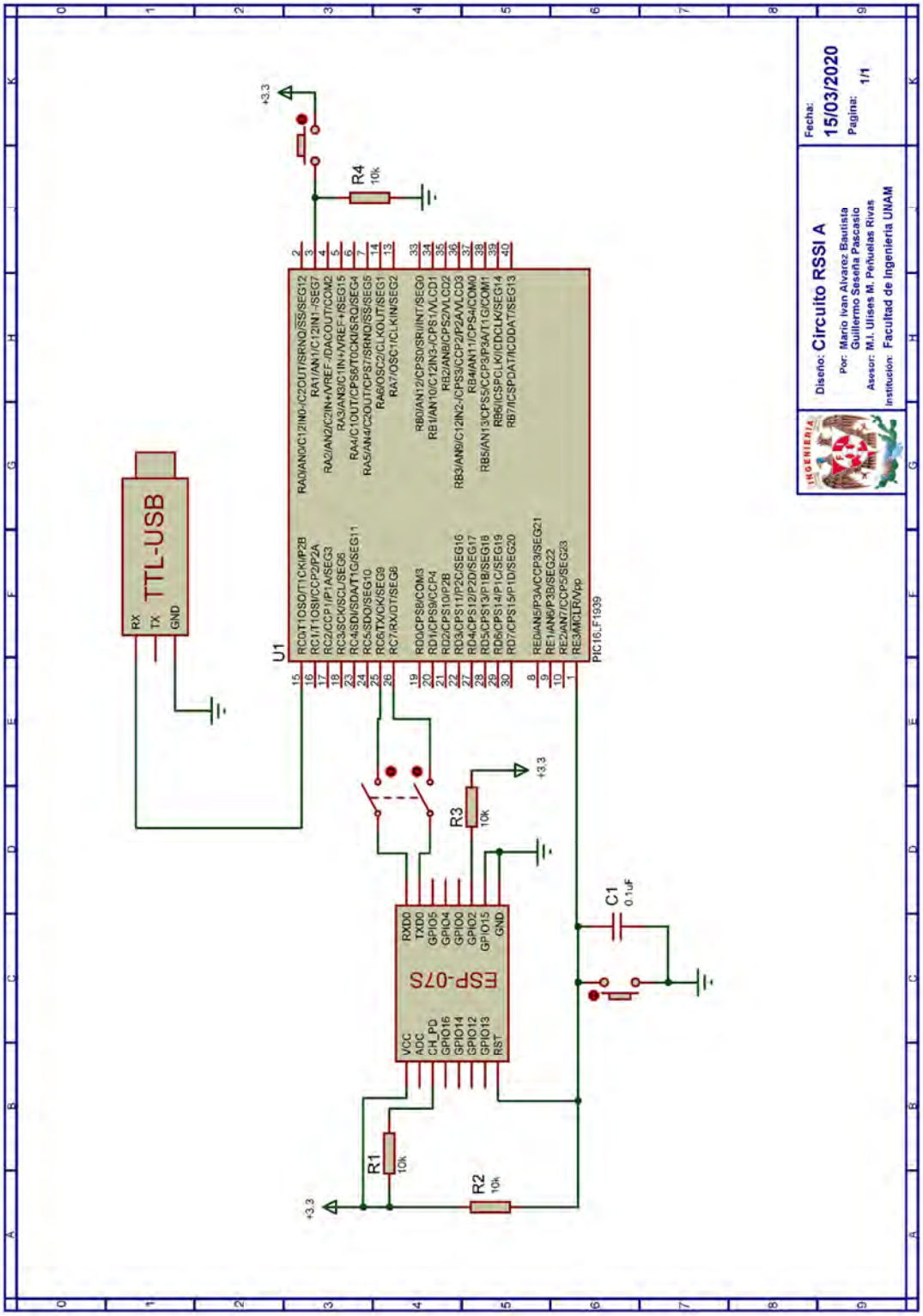


Figura P11-A.7 *Circuito de la práctica 11-A.*

Referencias

1. Sonia, A.J. and N.J. Joby, *Analyzing RFID Tags in a Distributed Environment*. 2015, International Conference on Emerging Trends in Engineering, Science and Technology.
2. Rosli, R.S., M.H. Habaebi, and M.R. Islam, *Analysis of different digital filters for received signal strength indicator*, in *Bulletin of Electrical Engineering and Informatics*. 2019.
3. Coleman, D.D. and D.D. Westcott, *CWNA® Certified Wireless Network Administrator Study Guide Exam CWNA-107*. Quinta ed. 2018, Indianapolis, Indiana: John Wiley & Sons.
4. Chia, Y., et al., *RSSI Comparison of ESP8266 Modules* 2018.
5. Ilci, V., et al., *Trilateration Technique for WiFi-Based Indoor Localization*. 2015, The Eleventh International Conference on Wireless and Mobile Communications.
6. Barai, S., D. Biswas, and B. Sau, *Estimate distance measurement using NodeMCU ESP8266 based on RSSI technique*. 2017.
7. Rosl, R.S., M.H. Habaebi, and M.R. Islam, *Characteristic Analysis of Received Signal Strength Indicator from ESP8266 WiFi Transceiver Module* 2018, 7th International Conference on Computer and Communication Engineering (ICCCE)
8. Habaebi, M.H. and N.I.N.B. Azizan, *Harvesting WiFi Received Signal Strength Indicator (RSSI) For Control/Automation System In SOHO Indoor Environment with ESP8266*. 2016, International Conference on Computer & Communication Engineering.

Práctica 11-B RSSI: simulación de sensor de barrera

Introducción

El Indicador de intensidad de señal recibida (*Received Signal Strength Indicator,* RSSI) es una medida del nivel de potencia que recibe la antena de un dispositivo, es decir la intensidad de la señal (amplitud). Esta medida está expresada como un número negativo en *dBm* y puede ser obtenida al procesar los cambios en el nivel de voltaje del circuito indicador de intensidad de señal recibida del receptor. Entre más cercano esté el valor RSSI de cero, la señal será más fuerte y mientras más alejado esté la señal será más débil [1-3].

RSSI resulta es medida arbitraria, ya que "la asignación de los valores de RSSI a la potencia recibida real depende de la implementación" de cada fabricante de las tarjetas inalámbricas. Esto da lugar a la aparición de diferentes rangos de valores y diferentes esquemas de numeración [3]. Es por esto que cuando se utilizan los valores RSSI de un producto, éste no puede ser remplazado indistintamente por otro, a menos que incorpore una tarjeta inalámbrica del mismo fabricante.

Generalmente el RSSI es utilizado en los módulos WiFi para adaptar dinámicamente su modulación y esquemas de codificación para lograr la velocidad de datos óptima [4]. Aunque también han surgido investigaciones para a partir de este mismo parámetro desarrollar diversas implementaciones, entre las que destacan:

- Estimación de distancia, la cual se puede determinar estableciendo una relación entre la distancia y el valor RSSI obtenido, ya sea mediante un modelo de propagación de la señal o mediante una técnica de ajuste de curva para obtener una ecuación que estime la distancia [5, 6].
- Determinación de posición en interiores mediante métodos como trilateración, en el cual se determina la posición de un objeto a partir de la intersección de círculos, cuyos radios son la estimación de la distancia del objeto a al menos tres puntos de acceso con ubicaciones ya conocidas [5].
- Seguimiento en interiores [7].
- Distinguir entre diferentes espesores de un material [7].
- Detector de presencia de personas, tal y como se muestra en [7] y [8].

Si bien, existen un gran número de tecnologías inalámbricas de las cuales se pueden obtener valores RSSI, se suele ocupar principalmente la tecnología WiFi ya que su infraestructura ya está presente en un gran número de dispositivos y lugares. Esto hace que no

se requiera adquirir, o en su caso muy poco, equipo adicional para el desarrollo de las aplicaciones antes mencionadas [5].

La señal WiFi, y por tanto los valores RSSI obtenidos, se ven afectados principalmente por efectos de desvanecimiento y multitrayectoria que son causados por obstrucciones físicas (la atenuación depende del material, geometría y espesor del objeto), interferencias de otras redes WiFi o dispositivos electromagnéticos, la distancia, el tipo de antenas y su orientación entre otros [2, 4, 5]. Estos efectos provocan que para la utilización de los valores RSSI se tengan que recabar una gran cantidad de lecturas para obtener una mayor precisión y que para minimizar la variabilidad debida a estos efectos se utilicen filtros digitales como promedios móviles simple, media alfa recortada o Kalman. La elección del filtro depende de las características de la aplicación y puede ser uno o una combinación de éstos[2].

Considerando las aplicaciones antes mencionadas, en la biblioteca ESP se ha desarrollado funciones para cualquiera de los SoC para:

- Obtención de un valor RSSI.
- Obtención del promedio de hasta 600 lecturas RSSI, en intervalos de tiempo entre 5 y hasta 65535 *ms.*
- Obtención del valor máximo, mínimo y promedio tras 99 lecturas RSSI, ya sea con desviación estándar o cuartiles principalmente.
- Obtención de valores RSSI de hasta cinco diferentes puntos de acceso. Ya sea un valor RSSI o el promedio de hasta 10 lecturas para cada AP.
- Establecer la obtención continua de valores RSSI, aplicando o no promedios móviles y verificando que se encuentren dentro de un rango establecido.

Objetivo

Ilustrar que el conjunto uC PIC y módulo WiFi puede ser utilizado para el desarrollo de trabajos que involucren simular el comportamiento un sensor de barrera.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

- STATION y RSSI en el módulo que funcione cono estación
- ACCESS_POINT y RSSI en el módulo que funcione como punto de acceso.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 2 interruptores DPST
- 6 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 2 condensadores de 0.1 µF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 *mA*

Descripción

Se utilizarán dos módulos WiFi, uno como estación conectado a otro que funcionará como punto de acceso. Ambos se colocarán al mismo nivel separados a una distancia de 5 centímetros. El módulo que funciona como estación determinará los valores máximo y mínimo RSSI, para posteriormente establecer la obtención continua de valores RSSI con el rango previamente obtenido. Cuando se obtenga un valor RSSI fuera del rango se tendrá que encender un LED por 4 segundos.

Código

El código que deberá ejecutar el microcontrolador PIC asociado al módulo que funcionará como punto de acceso se muestra en la figura *P11-B.1*, en el que se debe ingresar nombre y contraseña de un AP que se creará.

Mientras que el código del microcontrolador PIC asociado al módulo que funcionará como estación se muestra en la figura P11-B.2, en el que se debe ingresar nombre y contraseña del AP creado con el otro módulo.

Circuito

El circuito para el módulo que funciona como punto de acceso se muestra en la figura P11-B.3, mientras que el circuito del módulo que funciona como estación se muestra en la figura P11-B.4.

Resultados

Una vez los módulos WiFi estén alineados, al mismo nivel, separados por una distancia de 5 centímetros y sin algún objeto entre ellos, se deberá ejecutar primero el código del microcontrolador PIC asociado al módulo que funcionará como punto de acceso y verificar que dicho punto de acceso se ha creado.

Después, se ejecutará el código del microcontrolador PIC asociado al módulo que funcionará como estación que se conectará al AP creado por el otro

módulo, el LED asociado a éste permanecerá encendido hasta que se obtengan el valor máximo y mínimo. Posteriormente el LED se encenderá únicamente cuando se obtengan valores RSSI fuera del rango, para esto se pasará un objeto entre los módulos WiFi.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/37LQxol

Notas

- Para mejorar los resultados se recomienda utilizar una antena dipolar con conexión IPEX para el módulo ESP-07S.
- Se recomienda utilizar objetos que presenten diferencias significativas respecto a los RSSI que se obtienen.
- Se recomienda agregar un margen de error a los valores RSSI máximo y mínimo que se obtienen.
- La sensibilidad depende del tiempo entre lecturas RSSI.

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Creación de un punto de acceso en el canal 2
    begin_ap(2,"RSSI_AP","123456789",2);
    //Selección de 802.11g
    wifi_mode(2);
    //Se establece el máximo valor de RF TX
    wifi_tx_power(40);
    while(1)
    {
    }
}
```

Figura P11-B.1 Código de la práctica 11-B, módulo como punto de acceso.

```
#include <16lf1939.h>
#include "esp.h"

float rssi_max=0, rssi_min=0, rssi_mean=0;

void main()
{
    output_bit(PIN_A2, TRUE); // LED encendido
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso especificado
    begin_station("ESP","RSSI_AP","123456789");
    /*Función que cambia el nivel para poder utilizar las funciones RSSI*/
    up_level2_to_level3();
    output_bit(PIN_A2, TRUE); // LED encendido
    //Inicia la obtención de los valores RSSI máximo y mínimo
    start_mm_rssi(1,92);
    while (0==get_mm_rssi(&rssi_max,&rssi_min,&rssi_mean))
    {
        //Espera hasta la obtención de los valores
    }
    /*Comienza la obtención continua de valores RSSI, aplicando promedio
     moviles de 4 lecturas. Cada  lectura se realizará cada 30 ms*/
    output_bit(PIN_A2, FALSE); // Se apaga LED
    //Se agrega un majer de error a los valores obtenidos
    rssi_min=rssi_min-5;
    rssi_max=rssi_max+5;
    start_check_interval_rssi(4,rssi_min,rssi_max,30);
    while(1)
    {
        //¿Se detecto un valor RSSI fuera del intervalo?
        if(1==interval_alarm_rssi())
        {
            output_bit(PIN_A2, TRUE); // Enciende el LED
            delay_ms(4000);
             /*Comienza la obtención continua de valores RSSI*/
            start_check_interval_rssi(4,rssi_min,rssi_max,30);
        }
        else
        {
            output_bit(PIN_A2, FALSE); // Apaga el LED
        }
    }
}
```

Figura P11-B.2 *Código de la práctica 11-B, módulo como estación.*

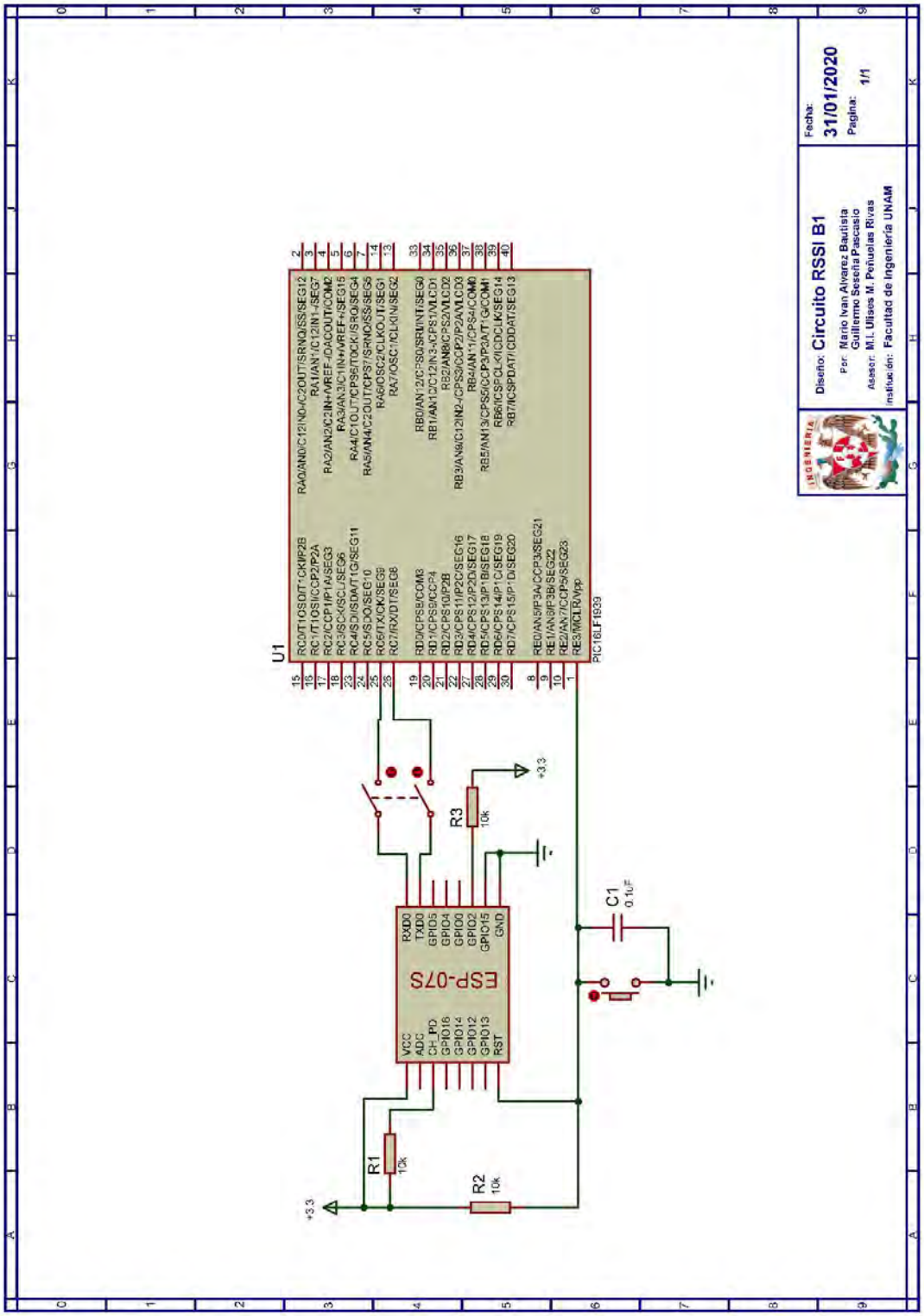


Figura P11-B.3 *Circuito de la práctica 11-B, módulo como punto de acceso.*

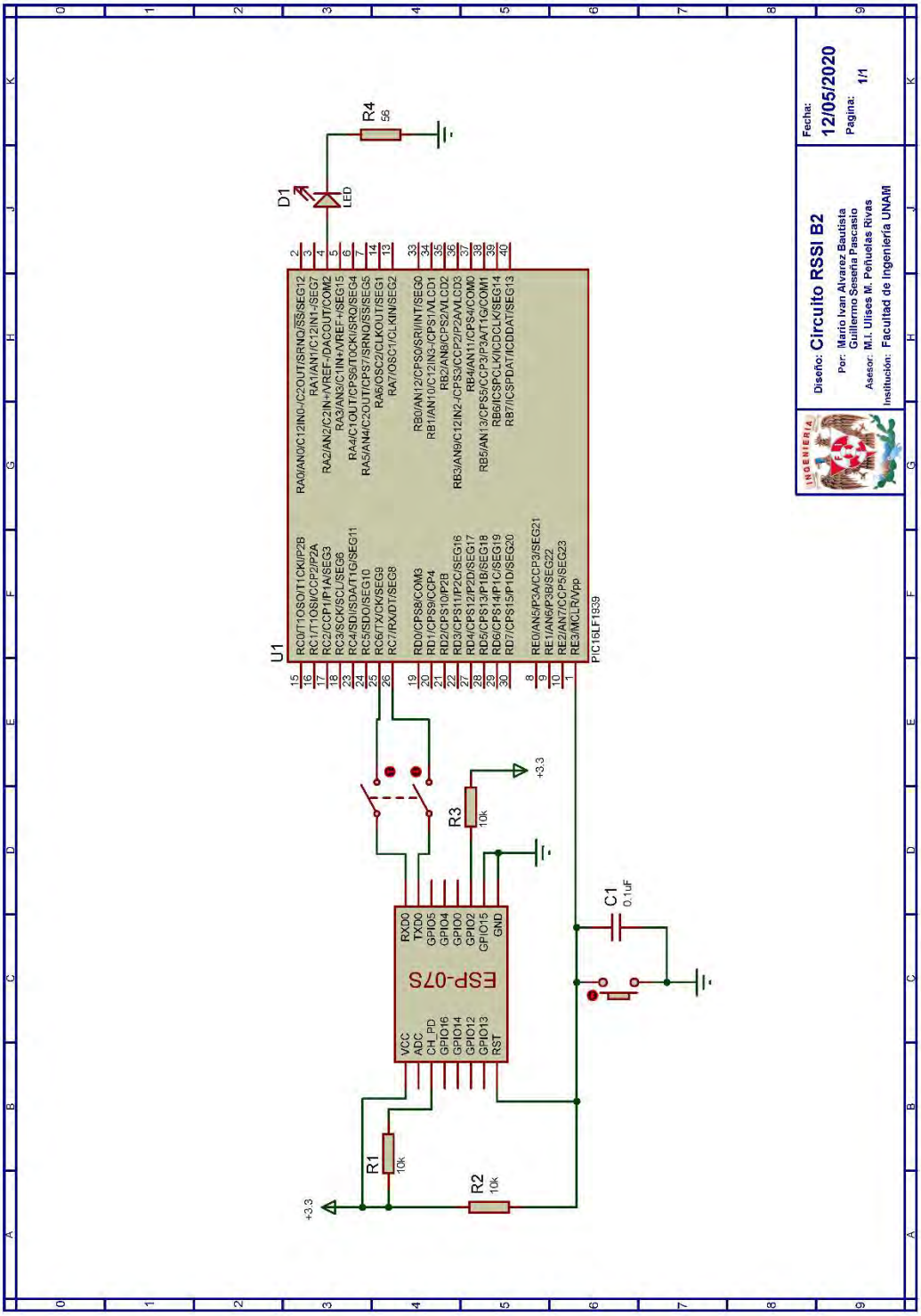


Figura P11-B.4 *Circuito de la práctica 11-B, módulo como estación.*

Referencias

1. Sonia, A.J. and N.J. Joby, *Analyzing RFID Tags in a Distributed Environment*. 2015, International Conference on Emerging Trends in Engineering, Science and Technology.
2. Rosli, R.S., M.H. Habaebi, and M.R. Islam, *Analysis of different digital filters for received signal strength indicator*, in *Bulletin of Electrical Engineering and Informatics*. 2019.
3. Coleman, D.D. and D.D. Westcott, *CWNA® Certified Wireless Network Administrator Study Guide Exam CWNA-107*. Quinta ed. 2018, Indianapolis, Indiana: John Wiley & Sons.
4. Chia, Y., et al., *RSSI Comparison of ESP8266 Modules* 2018.
5. Ilci, V., et al., *Trilateration Technique for WiFi-Based Indoor Localization*. 2015, The Eleventh International Conference on Wireless and Mobile Communications.
6. Barai, S., D. Biswas, and B. Sau, *Estimate distance measurement using NodeMCU ESP8266 based on RSSI technique*. 2017.
7. Rosl, R.S., M.H. Habaebi, and M.R. Islam, *Characteristic Analysis of Received Signal Strength Indicator from ESP8266 WiFi Transceiver Module* 2018, 7th International Conference on Computer and Communication Engineering (ICCCE)
8. Habaebi, M.H. and N.I.N.B. Azizan, *Harvesting WiFi Received Signal Strength Indicator (RSSI) For Control/Automation System In SOHO Indoor Environment with ESP8266*. 2016, International Conference on Computer & Communication Engineering.

Práctica 11-C RSSI: distinción de objetos

Introducción

El Indicador de intensidad de señal recibida (*Received Signal Strength Indicator,* RSSI) es una medida del nivel de potencia que recibe la antena de un dispositivo, es decir la intensidad de la señal (amplitud). Esta medida está expresada como un número negativo en *dBm* y puede ser obtenida al procesar los cambios en el nivel de voltaje del circuito indicador de intensidad de señal recibida del receptor. Entre más cercano esté el valor RSSI de cero, la señal será más fuerte y mientras más alejado esté la señal será más débil [1-3].

RSSI resulta es una medida arbitraria, ya que "la asignación de los valores de RSSI a la potencia recibida real depende de la implementación" de cada fabricante de las tarjetas inalámbricas. Esto da lugar a la aparición de diferentes rangos de valores y diferentes esquemas de numeración [3]. Es por esto por lo que cuando se utilizan los valores RSSI de un producto, éste no puede ser remplazado indistintamente por otro, a menos que incorpore una tarjeta inalámbrica del mismo fabricante.

Generalmente el RSSI es utilizado en los módulos WiFi para adaptar dinámicamente su modulación y esquemas de codificación para lograr la velocidad de datos óptima [4]. Aunque también han surgido investigaciones para a partir de este mismo parámetro desarrollar diversas implementaciones, entre las que destacan:

- Estimación de distancia, la cual se puede determinar estableciendo una relación entre la distancia y el valor RSSI obtenido, ya sea mediante un modelo de propagación de la señal o mediante una técnica de ajuste de curva para obtener una ecuación que estime la distancia [5, 6].
- Determinación de posición en interiores mediante métodos como trilateración, en el cual se determina la posición de un objeto a partir de la intersección de círculos, cuyos radios son la estimación de la distancia del objeto a al menos tres puntos de acceso con ubicaciones ya conocidas [5].
- Seguimiento en interiores [7].
- Distinguir entre diferentes espesores de un material [7].
- Detector de presencia de personas, tal y como se muestra en [7] y [8].

Si bien, existen un gran número de tecnologías inalámbricas de las cuales se pueden obtener valores RSSI, se suele ocupar principalmente la tecnología WiFi ya que su infraestructura ya está presente en un gran número de dispositivos y lugares [5]. Esto hace que

no se requiera adquirir, o en su caso muy poco, equipo adicional para el desarrollo de las aplicaciones antes mencionadas [5].

La señal WiFi, y por tanto los valores RSSI obtenidos, se ven afectados principalmente por efectos de desvanecimiento y multitrayectoria que son causados por obstrucciones físicas (la atenuación depende del material, geometría y espesor del objeto), interferencias de otras redes WiFi o dispositivos electromagnéticos, la distancia, el tipo de antenas y su orientación entre otros[2, 4, 5]. Estos efectos provocan que para la utilización de los valores RSSI se tengan que recabar una gran cantidad de lecturas para obtener una mayor precisión y que para minimizar la variabilidad debida a estos efectos se utilicen filtros digitales como promedios móviles simple, media alfa recortada o Kalman. La elección del filtro depende de las características de la aplicación y puede ser uno o una combinación de éstos [2].

Considerando las aplicaciones antes mencionadas, en la biblioteca ESP se ha desarrollado funciones para cualquiera de los SoC para:

- Obtención de un valor RSSI.
- Obtención del promedio de hasta 600 lecturas RSSI, en intervalos de tiempo entre 5 y hasta 65535 *ms.*
- Obtención del valor máximo, mínimo y promedio tras 99 lecturas RSSI, ya sea con desviación estándar o cuartiles principalmente.
- Obtención de valores RSSI de hasta cinco diferentes puntos de acceso. Ya sea un valor RSSI o el promedio de hasta 10 lecturas para cada AP.
- Establecer la obtención continua de valores RSSI, aplicando o no promedios móviles y verificando que se encuentren dentro de un rango establecido.

Objetivo

Ilustrar que el conjunto uC PIC y módulo WiFi puede ser utilizado para el desarrollo de trabajos que involucren la distinción de objetos mediante valores RSSI.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

- STATION y RSSI en el módulo que funcione cono estación
- ACCESS_POINT y RSSI en el módulo que funcione como punto de acceso.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 2 interruptores DPST
- 1 adaptador USB-TTL
- 6 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 2 condensadores de 0.1 µF
- 3 resistores de 56 Ω
- 3 LED de 2.2V y 20 *mA*

Descripción

Se utilizarán dos módulos WiFi, uno como estación conectado a otro que funcionará como punto de acceso.

I Obtención de rango de valores

Ambos módulos se colocarán al mismo nivel separados a una distancia de 20 centímetros, entre éstos se colocará un objeto. En la primera parte, el módulo que funciona como estación determinará los valores máximo y mínimo RSSI para dicho objeto cada vez que se presione un botón. Este proceso se repetirá para tres objetos diferentes.

II Distinción de objetos

Una vez obtenidos los rangos de valores RSSI asociados a cada objeto, en la segunda parte de esta práctica se cambiará el programa que ejecuta el microcontrolador PIC asociado al módulo que funciona como estación y ahora al presionar un botón obtendrá el valor promedio RSSI de 100 lecturas (debe haber un objeto entre los módulos). Conforme al resultado encenderá uno de los tres LED asociados a cada rango que se obtuvo en la primera parte.

Código

Para microcontrolador PIC que esté asociado al módulo que funcionará como punto de acceso se muestra en la figura P11-C.1, en el que se debe ingresar nombre y contraseña del AP que se creará.

El código que deberá ejecutar el microcontrolador PIC que esté asociado al módulo que funcionará como estación en la parte uno de la práctica se muestra en la figura P11-C.3, mientras que el código de la parte dos para este mismo módulo se muestra en la figura P11-C.4. En ambos se deben ingresar nombre y contraseña de del AP creado con el otro módulo.

Circuito

El circuito para el módulo que funciona como punto de acceso se muestra en la figura P11-C.5, mientras que el circuito

del módulo que funciona como estación se muestra en la figura P11-C.6.

Resultados

Una vez los módulos WiFi estén alineados, al mismo nivel, separados una distancia de medio metro y sin un objeto entre ellos. Se deberá ejecutar primero el código del microcontrolador PIC asociado al módulo que funcionará como punto de acceso y verificar que dicho punto de acceso se ha creado.

Después, se ejecutará el código de la primera parte del microcontrolador PIC asociado al módulo que funcionará como estación, el cual se conectará al AP creado por el otro módulo. Se colocará un objeto entre ambos módulos y se presionará un botón para obtener los valores RSSI máximo y mínimo que serán impresos en la terminal. Este proceso se debe repetir para tres objetos cuyos rangos de valores RSSI sean independientes y se debe registrar la parte entera de los valores obtenidos (ver figura P11-C.2).

Objeto	RSSI min	RSSI max
Botella de plastico	-22	-26
Desodorantes	-34	-37
Carpeta de tela	-46	-48

Figura P11-C.2 *Valores RSSI obtenidos para cada objeto.*

Para la segunda parte, se cambiará el programa del microcontrolador asociado al módulo que trabaja como estación, en el cual se establecerán los rangos previamente obtenidos (se agrega un factor error de +-2 a cada valor obtenido). Se colocará un objeto entre los módulos y al presionar un botón obtendrá el valor RSSI promedio y si éste pertenece a uno de los rangos previamente definidos, encenderá un LED correspondiente a uno de éstos.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3ejo7EH

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Creación de un punto de acceso en el canal 2
    begin_ap(2,"RSSI_AP","123456789",2);
    //Selección de 802.11g
    wifi_mode(2);
    //Se establece el máximo valor de RF TX
    wifi_tx_power(10);
    while(1)
    {
    }
}
```

Figura P11-C.1 *Código de la práctica 11-C, módulo como punto de acceso.*

```
#include <16lf1939.h>
#include "esp.h"

short button_state=false; // variable de control para la  obtención de RSSI
float rssi_max=0, rssi_min=0, rssi_mean=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso especificado
    begin_station("ESP","RSSI_AP","123456789");
    /*Función que cambia el nivel para poder utilizar las funciones RSSI*/
    up_level2_to_level3();
    while(1)
    {
        //¿Se ha presionado el botón?
        if(1==input_state(PIN_A1) && button_state==false)
        {
            while(1==input_state(PIN_A1))
            {
                //Se mantiene hasta soltar el botón
            }
            //Inicia la obtención de los valores RSSI máximo y mínimo
            start_mm_rssi(1,50);
            button_state=true;
        }
        //¿Se han obtenido los valores RSSI?
        if(1==get_mm_rssi(&rssi_max,&rssi_min,&rssi_mean))
        {
            fprintf(PIC,"MAX:%.3f,",rssi_max);
            fprintf(PIC,"MIN:%.3f\r\n",rssi_min);
            button_state=false;
        }
    }
}
```

Figura P11-C.3 *Código de la práctica 11-C, módulo como estación, para determinación de valores de cada rango.*

```
#include <16lf1939.h>
#include "esp.h"
// Se agrega un rango de error a los valores originales de +-2
#define V1_max -20.000
#define V1_min -28.000
#define V2_max -32.000
#define V2_min -39.000
#define V3_max -44.000
#define V3_min -50.000
short button_state=false; // variable de control pra la obtención de RSSI
float rssi=0;
void main()
{
    begin_esp(); //Inicialización del módulo Wi-Fi
    begin_station("ESP","RSSI_AP","123456789"); //Conexión al punto de acceso especificado
    up_level2_to_level3(); /*Función que cambia el nivel para poder utilizar las funciones RSSI*/
    while(1)
    {
        if(1==input_state(PIN_A1) && button_state==false)
        {
            while(1==input_state(PIN_A1))//Se mantiene hasta soltar el botón
            {
            }
            start_mean_rssi(100,100); // Comienza con la obtención del valor RSSI promedio
            button_state=true;
        }
        if(1==get_mean_rssi(&rssi)) //¿Se ha obtenido el valor RSSI promedio?
        {
            if(rssi>V1_min && rssi<V1_max) // Primer rango de valores
            {
                output_bit(PIN_A5,TRUE);
                output_bit(PIN_A2,FALSE);
                output_bit(PIN_A3,FALSE);
            }
            else if(rssi>V2_min && rssi<V2_max) // segundo rango de valores
            {
                output_bit(PIN_A5,FALSE);
                output_bit(PIN_A2,TRUE);
                output_bit(PIN_A3,FALSE);
            }
            else if(rssi>V3_min && rssi<V3_max) // tercer rango de valores
            {
                output_bit(PIN_A5,FALSE);
                output_bit(PIN_A2,FALSE);
                output_bit(PIN_A3,TRUE);
            }
            button_state=false;
        }
    }
}
```

Figura P11-C.4 *Código de la práctica 11-C, módulo como estación para distinción de objetos.*

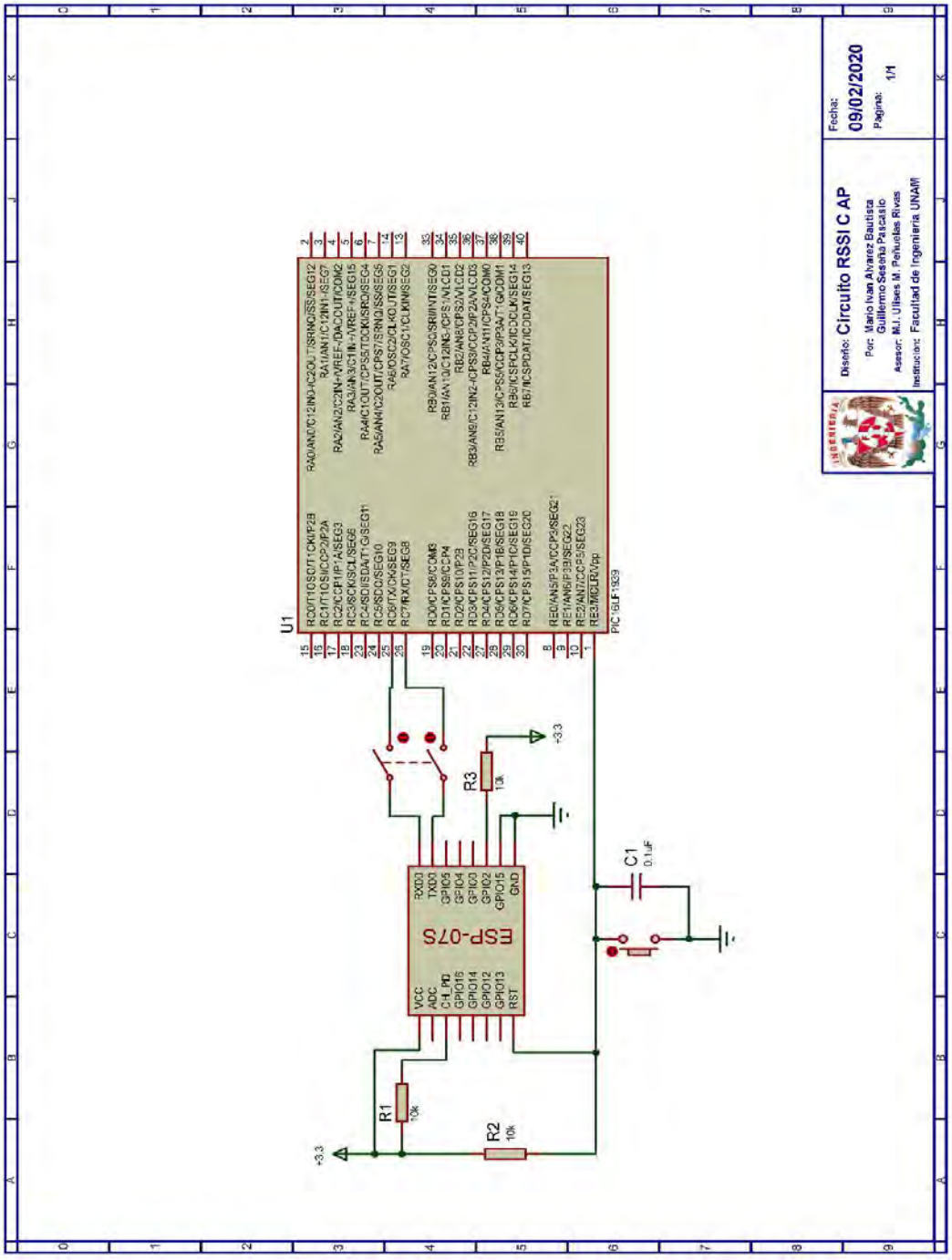


Figura P11-C.5 *Circuito de la práctica 11-C, módulo como AP.*

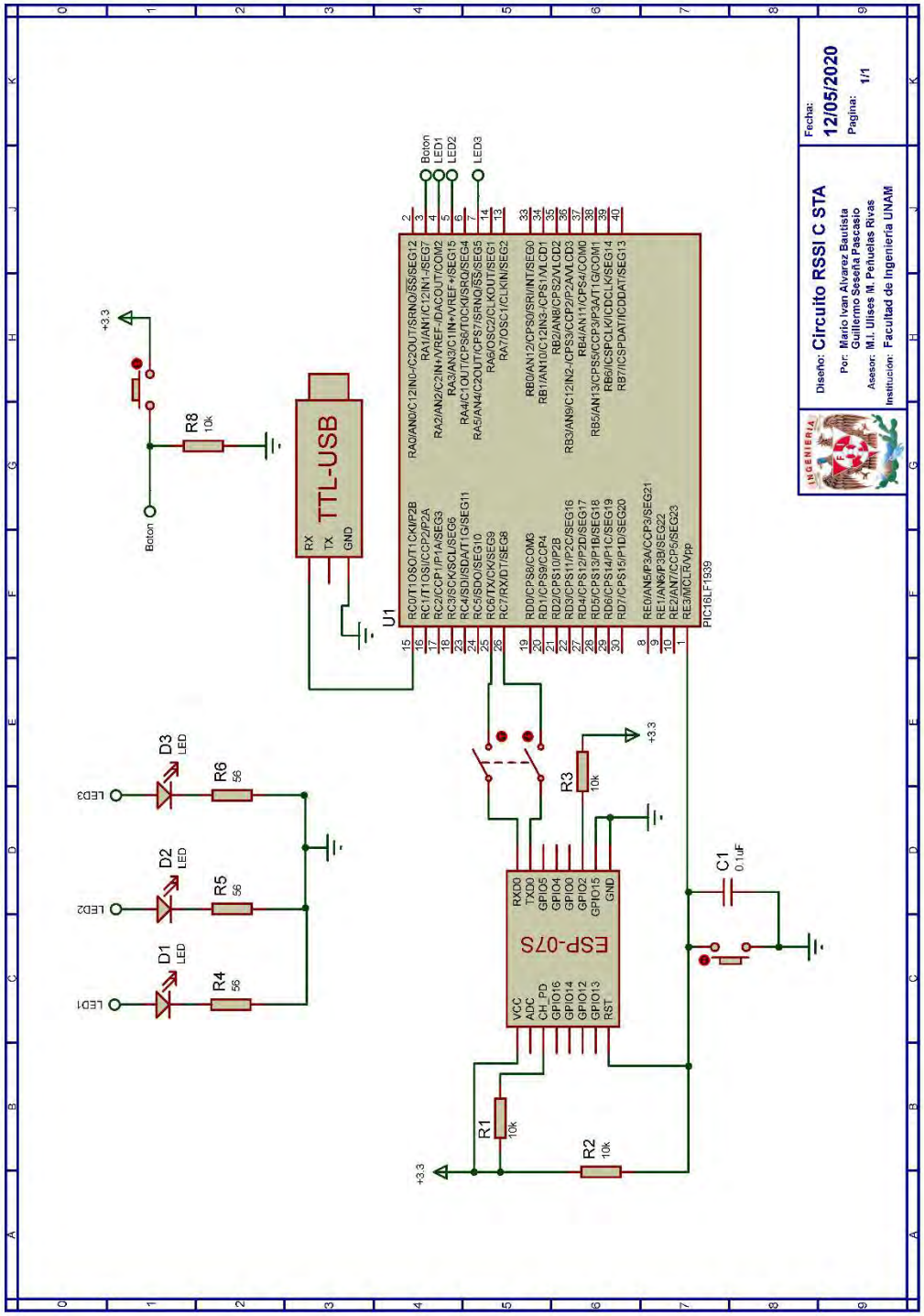


Figura P11-C.6 *Circuito de la práctica 11-C, módulo como estación.*

Notas

- Para mejorar los resultados se recomienda utilizar una antena dipolar con conexión IPEX para el módulo ESP-07S.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.

Referencias

1. Sonia, A.J. and N.J. Joby, *Analyzing RFID Tags in a Distributed Environment*. 2015, International Conference on Emerging Trends in Engineering, Science and Technology.
2. Rosli, R.S., M.H. Habaebi, and M.R. Islam, *Analysis of different digital filters for received signal strength indicator*, in *Bulletin of Electrical Engineering and Informatics*. 2019.
3. Coleman, D.D. and D.D. Westcott, *CWNA® Certified Wireless Network Administrator Study Guide Exam CWNA-107*. Quinta ed. 2018, Indianapolis, Indiana: John Wiley & Sons.
4. Chia, Y., et al., *RSSI Comparison of ESP8266 Modules* 2018.
5. Ilci, V., et al., *Trilateration Technique for WiFi-Based Indoor Localization*. 2015, The Eleventh International Conference on Wireless and Mobile Communications.
6. Barai, S., D. Biswas, and B. Sau, *Estimate distance measurement using NodeMCU ESP8266 based on RSSI technique*. 2017.
7. Rosl, R.S., M.H. Habaebi, and M.R. Islam, *Characteristic Analysis of Received Signal Strength Indicator from ESP8266 WiFi Transceiver Module* 2018, 7th International Conference on Computer and Communication Engineering (ICCCE)
8. Habaebi, M.H. and N.I.N.B. Azizan, *Harvesting WiFi Received Signal Strength Indicator (RSSI) For Control/Automation System In SOHO Indoor Environment with ESP8266*. 2016, International Conference on Computer & Communication Engineering.

Práctica 12 Simple pair

Introducción

Simple pair es un protocolo de emparejamiento basado en ESP IE utilizado para:

- Realizar un rápido intercambio de claves o información entre dos dispositivos en pocos pasos.
- Realizar el emparejamiento entre dos dispositivos que no se habían emparejado antes.

Actualmente, este protocolo sólo puede ser utilizado para enviar información limitada hasta 16 bytes, desde un módulo WiFi en modo AP a otro módulo WiFi en modo estación, siendo este último el que la solicita. *Simple pair* utiliza una clave denominada *Temkey* que debe ser del conocimiento de ambos módulos (AP y estación) antes de iniciar *simple pair,* la cual es utilizada para asegurar el último paso del intercambio. Si bien, *simple pair* es completado tras el envío de la información, en un proceso de emparejamiento para la capa de aplicación este proceso finaliza sólo cuando la clave se verifica como válida [1]. En *simple pair* resulta fundamental conocer la dirección MAC de los módulos que participan, en especial de quien se le solicitará la información.

La biblioteca ESP permite realizar las siguientes acciones relacionadas con *simple pair*:

- Verificar si un módulo ha habilitado *simple* pair (a partir de su dirección MAC).
- Establecer y anunciar un mensaje o clave mediante *simple pair.*
- Obtener un mensaje o clave de mediante *simple pair,* ya sea de manera continua o alterna
- Establecer un filtro que especifica a que módulos se entregará la información (Se requiere establecer sus direcciones MAC).

Objetivo

Mostrar como utilizar *simple pair* para compartir información.

Firmware de comandos AT

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3 o en el SDK NONOS 2.2.1, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_SIMPLE_PAIR módulo que funcione cono estación

- ACCESS_POINT y E_SIMPLE_PAIR en el módulo que funcione como punto de acceso.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 2 interruptores DPST
- 1 adaptador USB-TTL
- 8 resistores de 10 kΩ
- 4 botones pulsadores n.o.
- 2 condensadores de 0.1 µF

Descripción

Se utilizarán dos módulos WiFi, uno funcionará como estación mientras que el otro como punto de acceso. El módulo que funcione como punto de acceso deberá mandar el mensaje “hola mundo” al módulo que funciona como estación, para esto, éste último deberá comprobar que el módulo que funciona como AP ha habilitado *simple pair* y así solicitar la información que éste ofrece al presionar un botón. El mensaje recibido será impreso en una terminal.

Una vez realizado la anterior, se deberá presionar el botón asociado al microcontrolador PIC del módulo WiFi que funciona como punto de acceso, el cual activará el filtro *simple* pair especificando una dirección MAC ficticia. Posteriormente el módulo que funciona como estación deberá solicitar nuevamente la información e imprimir un mensaje en la terminal que indique si fue posible obtener la información o no.

Antes de comenzar la práctica se deben conocer las direcciones MAC de cada módulo, conforme a la interfaz que ejecutarán (estación o punto de acceso).

Código

El código del microcontrolador asociado al módulo que funcionará como estación se muestra en la figura P12.4, en el que se debe ingresar la dirección MAC del módulo (que funciona como AP) al que se le solicitará la información.

Mientras que el código del microcontrolador asociado al módulo que funcionará como punto de acceso se muestra en la figura P12.5, en el que se debe establecer para el filtro una dirección MAC diferente a la del módulo (que funciona como estación) que solicitará la información con la intención de que no le sea entregado la información a éste cuando se active.

Circuito

Tanto el circuito del módulo que funciona como punto de acceso como el que funciona como estación, se muestran en la figura P12.6, nótese que, el circuito del módulo que funciona como AP no requiere del adaptador USB-TTL.

Resultados

Primero debe iniciar la ejecución del programa del microcontrolador PIC asociado al módulo WiFi que funciona como punto de acceso. Posteriormente se ejecutará el código del microcontrolador asociado al módulo que funciona como estación y una vez que en la terminal aparezca el mensaje “ready”, se presionará el botón asociado a este microcontrolador para solicitar la información. En la terminal se deberá observar el mensaje solicitado tal y como se muestra en la figura P12.1.

```
Modulo esp listo para su uso\r\n
ready\r\n
Mensaje:Hola mundo\r\n
```

Figura P12.1 *Obtención correcta del mensaje.*

Una vez recibido el mensaje, se deberá presionar el botón del microcontrolador PIC asociado al módulo WiFi que funciona como punto de acceso para activar el *filtro simple pair*. Después, se deberá solicitar nuevamente la información y se deberá observar en la terminal el mensaje de la figura P12.2.

```
Modulo esp listo para su uso\r\n
ready\r\n
Mensaje:Hola mundo\r\n
No se pudo obtener\r\n
```

Figura P12.2 *Fallo al obtener el mensaje.*

Nuevamente, se presionará el botón del microcontrolador PIC asociado al módulo WiFi que funciona como punto de acceso para desactivar el filtro *simple pair.* Es entonces, cuando al solicitar la información nuevamente, se deberá apreciar el mensaje otra vez (ver figura P12.3).

```
Modulo esp listo para su uso\r\n
ready\r\n
Mensaje:Hola mundo\r\n
No se pudo obtener\r\n
Mensaje:Hola mundo\r\n
```

Figura P12.3 *Obtención correcta del mensaje tras un intento fallido.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2YeVUta

```
#include <16lf1939.h>
#include "esp.h"

char buffer[17]={0};
int key[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int err=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Sube nivel para poder usar las funciones simple pair*/
    up_to_level2();
    //Habilita el uso de las funciones simple pair*/
    begin_simple_pair();
    fprintf(PIC,"ready\r\n");
    while (1)
    {
        if(1==input_state(PIN_A2)) //¿Solicitar información?
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta soltar el botón
            }
            //¿El dispositivo objetivo ha habilitado simple pair
            if(1==simple_pair_check("5E:CF:7F:D0:DA:E6"))
            {
                // Obtiene el mensaje que se esté anunciando
                err=get_message_simple_pair(0,1,"5E:CF:7F:D0:DA:E6",buffer, sizeof(buffer),key, sizeof(key));
                if(err==0)
                    fprintf(PIC,"Mensaje:%s\r\n",buffer);
                else
                    fprintf(PIC,"No se pudo obtener\r\n");
            }
            else
            {
                fprintf(PIC,"No ha habilitado simple pair el dispositivo objetivo\r\n");
            }

        }
    }
}
```

Figura P12.4 *Código de la práctica 12, módulo como estación.*

```
#include <16lf1939.h>
#include "esp.h"

int key[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short ctr=false; // varible de control
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Sube nivel para poder usar las funciones simple pair*/
    up_to_level2();
    //Habilita el uso de las funciones simple pair*/
    begin_simple_pair();
    // Establece el mensaje a compartir
    set_message_simple_pair(0,"Hola mundo", sizeof("Hola mundo"));
    //Anuncia que está disponible un mensaje para que lo soliciten
    announcing_message_simple_pair(1,key, sizeof(key));
    while (1)
    {

        if(1==input_state(PIN_A2))
        {
            /*Habilita o deshabilita el filtro simple pair
             para  determinar  a quién se le entregará el mensaje*/
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta soltar el botón
            }
            ctr=!ctr;
            if(ctr)
                begin_filter_simple_pair("00:00:00:00:00:00",key, sizeof(key));
            else
                finish_filter_simple_pair();
        }
    }
}
```

Figura P12.5 *Código de la práctica 12, módulo como AP.*

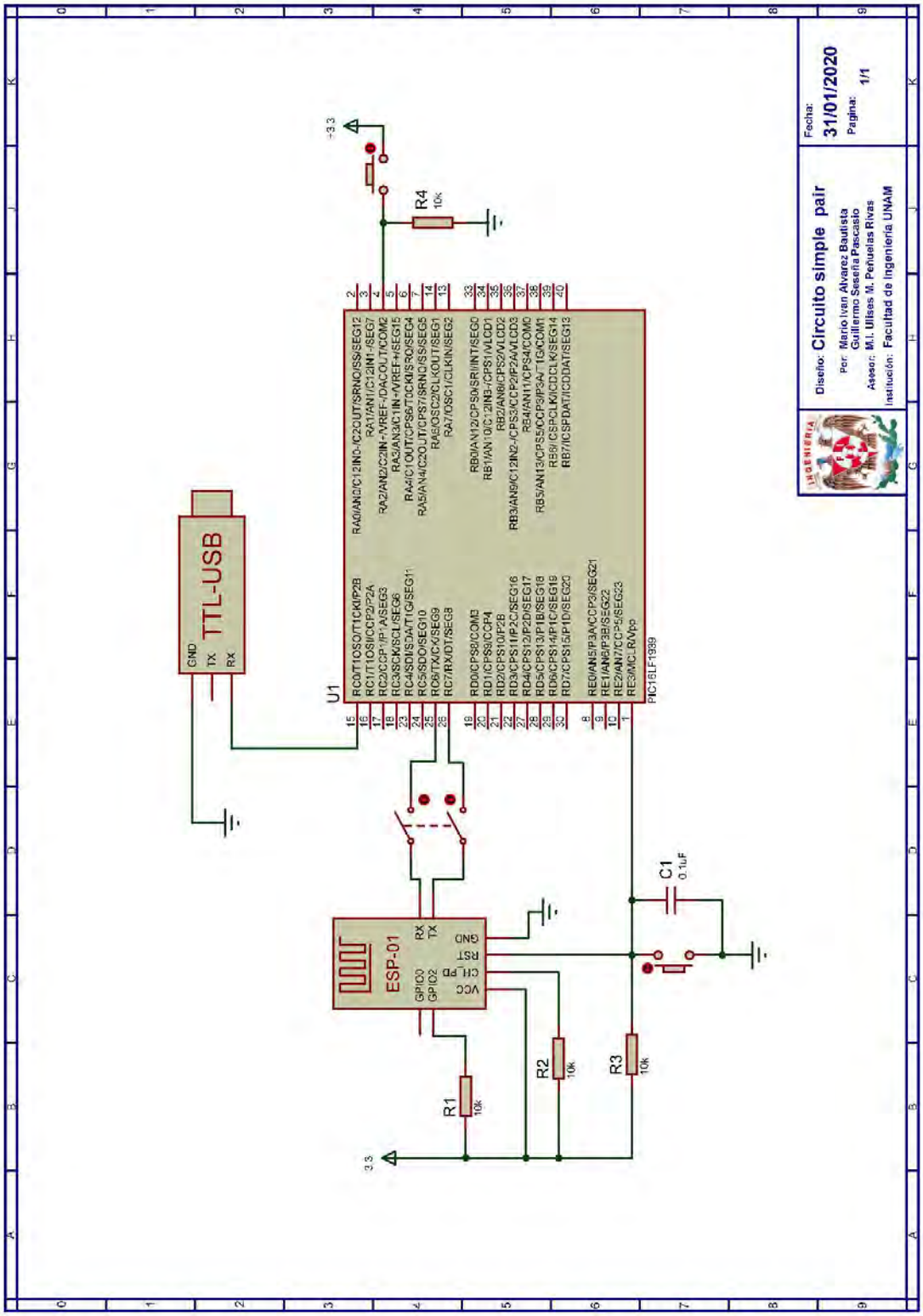


Figura P12.6 *Circuito de la práctica 12.*

Notas

- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- El *timeout* de *simple pair* es de casi 30 segundos, razón por la que se debe comprobar al menos una vez si el *peer* ha habilitado *simple pair*. Una vez realizado la comprobación, ésta se puede deshabilitar para hacer los intercambios de información más rápidos.

Referencias

1. Espressif Systems, *Simple-Pair User Guide*. 2016.

Práctica 13 ESPNOW

Introducción

ESPNOW es un protocolo desarrollado por Espressif Systems para la comunicación entre dispositivos, el cual se caracteriza por ser rápido, no orientado a la conexión, con transmisión en pequeños paquetes y por requerir un emparejamiento entre los dispositivos previo a la comunicación. Una vez realizado el emparejamiento, la conexión es segura, *peer-to-peer* y no se requiere realizar un *handshake*.

Este protocolo es similar a la conectividad inalámbrica de baja potencia de 2.4 *GHz*, utiliza el *action frame* de IEEE802.11 para encapsular y transmitir datos, junto con la función IE desarrollada por Espressif y la tecnología de cifrado CCMP para el *action frame* [1-3].

Entre las características de ESPNOW que soportan los SoC ESP8266 y ESP32 se tienen:

Tabla P13.1 *Descripción de cada parámetro de ESPNOW [2].*

Dispositivo	Información	Valor/longitud	Descripción	Nota
Local	PMK	16 bytes	*Primary Master Key* usada para cifrar la clave del *peer* (KOK en API)	El sistema utilizará una PMK por defecto. Si es establecida, deberá ser la misma que la del dispositivo local.
	Rol	Maestro Esclavo	El rol del dispositivo	El rol del dispositivo define la interfaz de transmisión (Dirección MAC). Maestro: Interfaz estación. Esclavo: Interfaz *softAP.*
Peer	Key	16 bytes	Clave utilizada para cifrar el *payload* durante la comunicación.	
	Dirección MAC	*String*	Dirección MAC del *peer*	La dirección MAC debe ser la de la interfaz de transmisión.
	Rol	Maestro (0) Esclavo (1)	El rol del *peer*	El rol del *peer* no afecta ninguna función, sólo es almacenado en la capa de aplicación.
	Canal	0-14	Canal a través del cual el dispositivo local y el *peer* se comunicarán	El canal del *peer* no afecta ninguna función, sólo es almacenado en la capa de aplicación. Cero significa que el canal no está definido.

- Comunicación de difusión única (*unicast*) cifrada y decifrada.
- Mezclar *peers* de dispositivos cifrados y decifrados.
- El *payload,* es limitado a 250 bytes.
- Los *peer* cifrados están limitados. 10 *peer* cifrados en modo estación, 6 *peer* cifrados en modo *softAP*.
- Múltiples *peer* decifrados son soportados.
- En total el número de *peers* debe ser menor a 20, incluyendo los *peer* cifrados [2, 3].

La información que requiere ESPNOW se detalla en la tabla P13.1. Esta información es usada para enviar y recibir información, y es mantenida en una capa de bajo nivel de ESPNOW. Este protocolo brinda una alternativa de baja potencia y eficiencia energética para el control directo y comunicación sin la necesidad de un *router.*

La biblioteca ESP permite realizar las siguientes acciones:

- Establecer opcionalmente un *buffer* para almacenar los mensajes que se reciban, de lo contrario sólo podrá enviar información.
- Enviar *strings* alojados en la ROM del PIC (máximo 250 bytes).
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 250 bytes).
- Registrar *peers* en una tabla de comunicación ESPNOW.
- Enviar un mensaje a un *peer* en particular.
- Enviar el mismo mensaje a todos los *peer* registrados en la tabla de comunicación ESPNOW.
- Establecer un filtro que especifica de que módulos se recibe información (se requiere establecer sus direcciones MAC).

Objetivo

Mostrar como utilizar ESPNOW como otra alternativa de protocolo de comunicación inalámbrica WiFi entre módulos ESP.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP32-DevKitC.

Habilitación de aplicaciones

- STATION y E_ESPNOW en el módulo que funcione como estación.
- ACCESS_POINT y E_ESPNOW en el módulo que funcione como punto de acceso.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 2 interruptores DPST
- 6 resistores de 10 kΩ
- 4 botones pulsadores n.o.
- 2 condensadores de 0.1 μF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 *mA*

Descripción

Se utilizarán dos módulos WiFi, uno como estación y otro como punto de acceso. Previamente, se debe conocer la dirección MAC de ambos módulos.

El uC PIC asociado al módulo WiFi que funcione como punto acceso tendrá un botón asociado, el cual cada vez sea presionado alternará entre los mensajes ON y OFF, los cuales serán enviados al módulo que funciona como estación. El uC PIC asociado al módulo que funciona como estación controlará el estado de un LED conforme a los mensajes recibidos desde el módulo.

Así mismo, este uC PIC contará con un botón, el cual al ser presionado habilitará el filtro ESPNOW especificando una dirección MAC ficticia. Al presionar nuevamente este botón, el filtro debe desactivarse.

Cuando se utiliza el protocolo de comunicación ESPNOW se deben conocer previamente las direcciones MAC de los módulos WiFi conforme a la interfaz que ejecutarán (estación o punto de acceso). En este caso, únicamente se debe conocer la dirección MAC del módulo que funcionará como estación, puesto que la comunicación en esta práctica será unidireccional desde el módulo como AP hacia el módulo como estación.

Código

El código del microcontrolador asociado al módulo WiFi que funciona como estación se muestra en la figura P13.1, en el que se debe ingresar una dirección MAC diferente a la del módulo que funciona como AP para el filtro

Mientras que el código del microcontrolador asociado al módulo que funcionará como punto de acceso se muestra en la figura P13.2, en el que se debe ingresar la dirección MAC del módulo que funciona como estación para enviarle los mensajes.

Circuito

El circuito para el módulo que funciona como punto de acceso se muestra en la figura P13.3, mientras que el circuito para

el módulo que funciona como estación se muestra en la figura P13.4.

Resultados

Al ejecutarse el código de ambos microcontroladores, se debe presionar el botón del microcontrolador asociado al módulo que funciona como punto de acceso y verificar que el LED asociado al módulo que funciona como estación, alterne entre los estados encendido y apagado tras cada pulsación. Esto corroborará que el intercambio de mensajes entre los módulos se está llevando a cabo correctamente.

Posteriormente, se presionará el botón del microcontrolador asociado al módulo que funciona como estación. Cada vez que se presiona se activa y desactiva el filtro espnow, lo cual evita que se reciban mensajes desde dispositivos no especificados. Esto hará que cada vez que se habilite el filtro, el LED no cambie de estado, esto hasta que se desactive el filtro.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3eiqO9B

Nota

- Cuando se utiliza el protocolo de comunicación ESPNOW se deben conocer previamente las direcciones MAC de los módulos WiFi conforme a la interfaz que ejecutarán (estación o punto de acceso).

Práctica propuesta

Utilizar la tabla de comunicación ESPNOW para enviar la información obtenida de un sensor de temperatura hacia 3 módulos WiFi, para lo cual se deben utilizar las funciones *add_peer_espnow* y *send_all_espnow*.

```
#include <16lf1939.h>
#include "esp.h"

char buffer[8]={0};
short ctr=false;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    /*Sube de nivel  para utilizar las funciones espnow y
     establece el módulo como estación (habilitar STATION en esp.h)*/
    up_to_level2();
    //Establece el buffer para recibir mensajes en espnow
    set_buffer_espnow(buffer,sizeof(buffer));
    //Habilita el uso de espnow
    begin_espnow();
    while(true)
    {
        if(1==compare_string(buffer,"ON")) //¿El mensaje recibido fue ON?
        {
            output_bit(PIN_A2,TRUE);
            //Envía el mensaje de confirmación al módulo AP
        }
        else if(1==compare_string(buffer,"OFF")) //¿El mensaje recibido fue OFF?
        {
            output_bit(PIN_A2,FALSE);
            //Envía el mensaje de confirmación al módulo AP
        }
        //Habilita o deshabilita el filtro espnow usando una mac ficticia
        if(1==input_state(PIN_A1))
        {
            while(1==input_state(PIN_A1))
            {
                //Se mantiene hasta soltar el botón
            }
            ctr=!ctr;
            if(ctr)
            {
                // Habilita el filtro con una dirección MAC ficticia
                begin_filter_espnow("5C:CF:7F:D1:9A:77");
            }
            else
            {
                //Deshabilita  el filtro espnow
                finish_filter_espnow();
            }
        }
    }
}
```

Figura P13.1 *Código de la práctica 13, módulo como estación.*

```
#include <16lf1939.h>
#include "esp.h"

char buffer[8]={0};
short ctr=false;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
     /*Sube de nivel  para utilizar las funciones espnow y
     establece el módulo como AP (habilitar ACCESS_POINT en esp.h)*/
    up_to_level2();
    //Habilita el uso de espnow
    begin_espnow();
    while(true)
    {
        // ¿Se presionó el botón?
        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta soltar el botón
            }
            // Envia un mensaje con el estado del LED deseado
            ctr=!ctr;
            if(ctr)
                send_espnow("5B:CF:7F:D1:9A:77","ON") // MAC del módulo en STA
            else
                send_espnow("5B:CF:7F:D1:9A:77","OFF") // MAC del módulo en STA
        }
    }
}
```

Figura P13.2 *Código de la práctica 13, módulo como punto de acceso.*

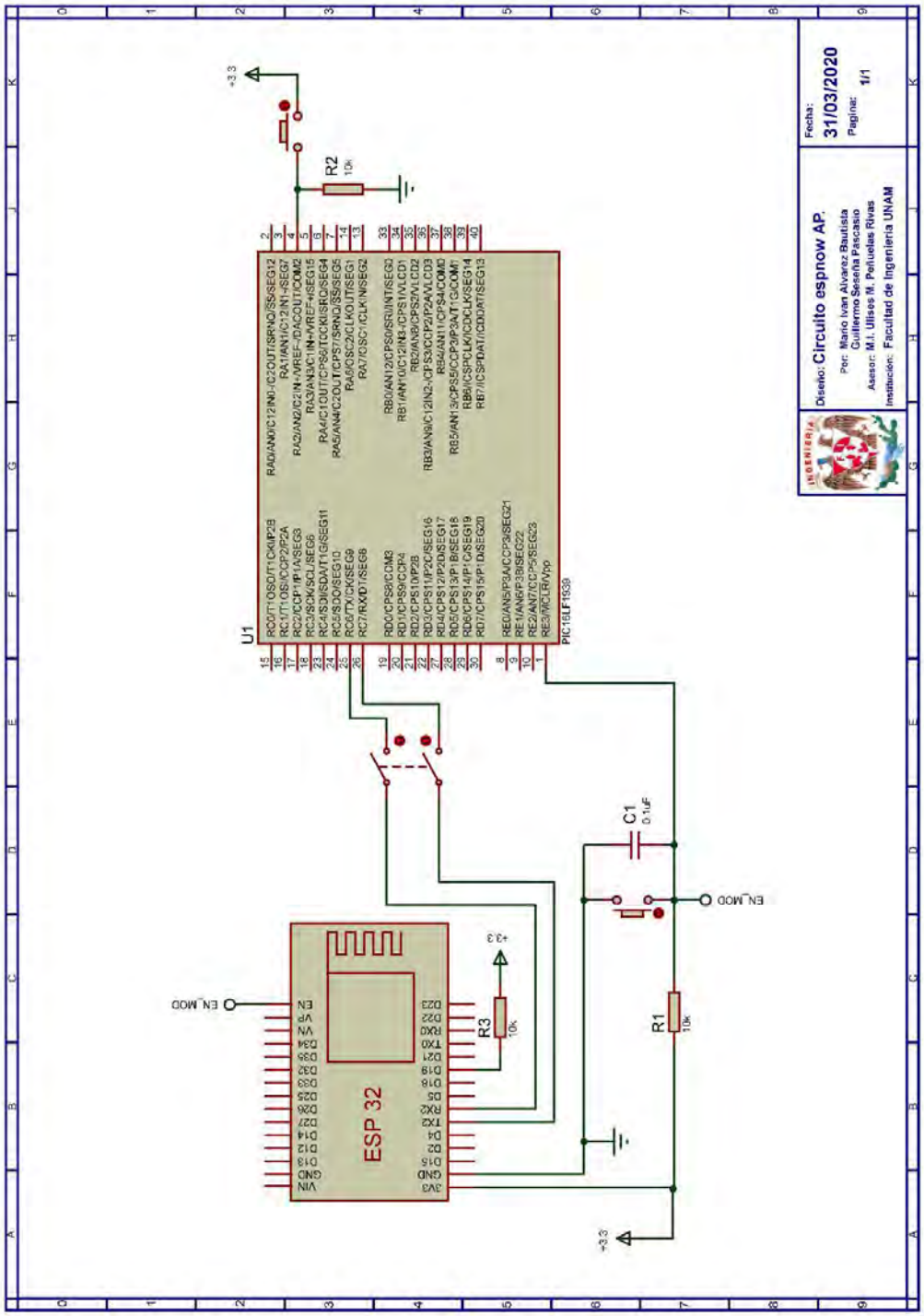


Figura P13.3 *Circuito de la práctica 13, módulo como AP.*

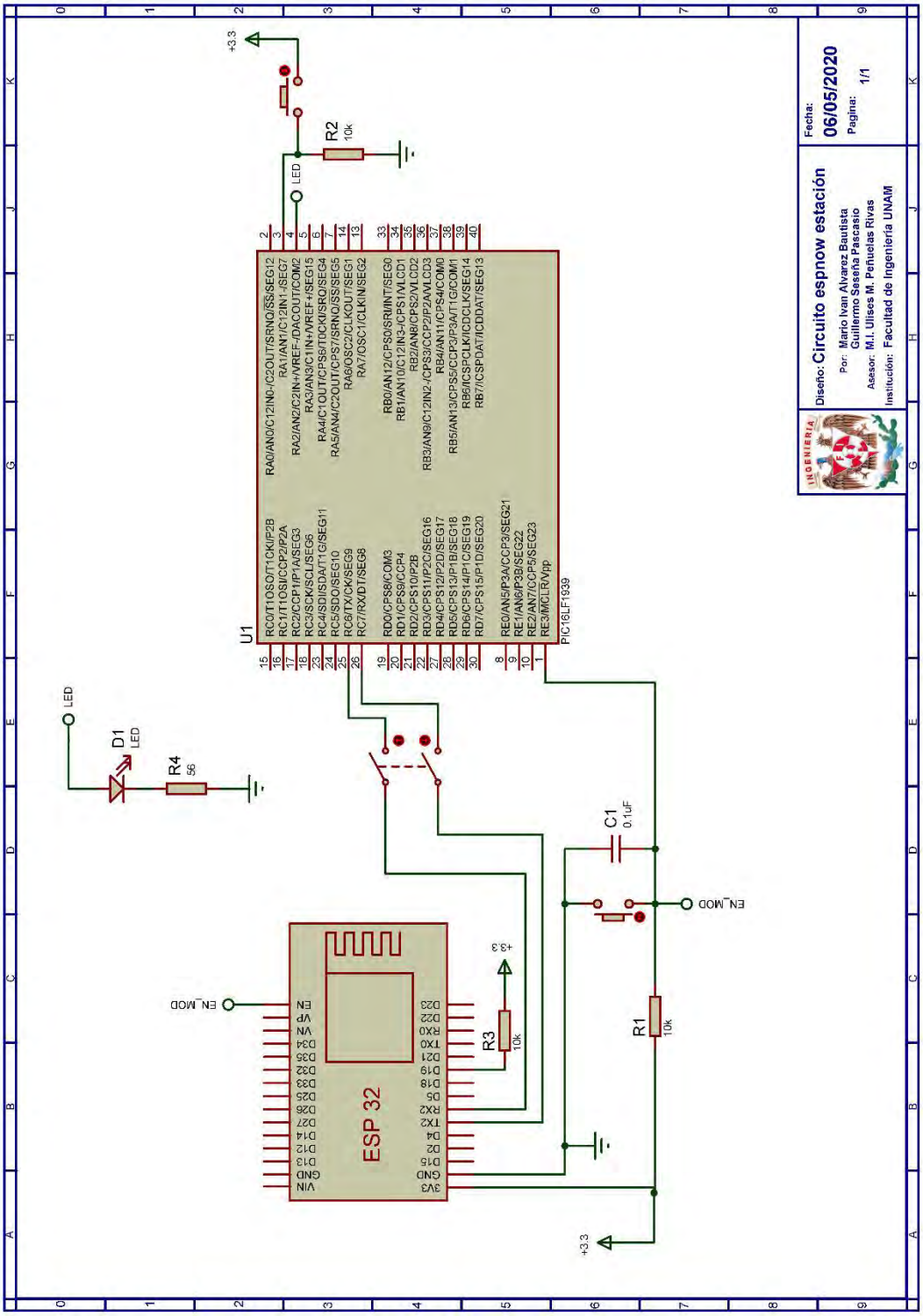


Figura P13.4 *Circuito de la práctica 13, módulo como estación.*

Referencias

1. Espressif Systems. *ESPNOW overview*. [Citado 2020 Marzo]; Disponible en:https://www.espressif.com/en/products/software/esp-now/overview.
2. Espressif Systems, *ESPNOW User Guide v1.0*. 2016.
3. Espressif Systems. *ESP-IDF Programming Guide v3.3*. [Citado 2020 Marzo]; Disponible en:https://docs.espressif.com/projects/esp-idf/en/v3.3/index.html.

Práctica 14-A WiFi Tracking: detección de módulos WiFi

Introducción

WiFi *Tracking* o rastreo WiFi hace referencia a los dispositivos que implementan la tecnología WiFi para detectar paquetes *probe request* que son emitidos por otros dispositivos WiFi en todos los canales, con la finalidad de obtener datos de las personas que usan éstos para:

- Determinar su localización
- Reconstruir o inferir sus trayectorias
- Estimar presencia.

Lo anterior basado en el hecho de que cada dispositivo detectado está asociado a una única persona. Para emplear WiFi *Tracking,* los dispositivos WiFi que se desean rastrear deben estar ejecutando lo que se denomina un escaneo activo para identificar redes disponibles, es decir, los dispositivos deben emitir paquetes *probe request* (sonda de solictud). WiFi *Tracking* es frecuentemente empleado para la detección teléfonos inteligentes, ya que la mayoría de éstos incorporan la tecnología WiFi y generalmente se encuentran ejecutando un escaneo activo. La detección de teléfonos inteligentes puede ser utilizado para ubicar a los propietarios de éstos dispositivos [2-4]. La información obtenida mediante WiFi *Tracking* tiene un alto valor comercial y social, ejemplos de en lo que se puede aprovechar se mencionan en [2] y [3].

El *probe request* pertenece a los *management frames* (marcos de gestion), los cuales son un conjunto de paquetes que están definidos por el estándar IEEE 802.11 y que son utilizados para la comunicación inicial (proceso de conexión) entre una estación y un punto de acceso [1, 5, 6]. El *probe request* es utilizado para detectar puntos de acceso cercanos, el cual contendrá los parámetros y capacidades de la estación (ver figura P14-A.1) y como es transmitido antes del proceso de asociación con un punto de acceso, el paquete no contiene ningún tipo de cifrado. Cuando la estación busca un punto de acceso específico (*directed probe request*), este paquete contendrá el SSID del punto de acceso, mientras que cuando busca cualquier punto de

Order	Informations
1	SSID
2	Supported Rates
3	Request Informations
4	Extended Supported Rates
5	DSSS Parameter Set
6	Supported Operating Classes
7	HT Capabilities
8	20/40 BSS Coexistence
9	Extended Capabilities
10	SSID List
11	Channel Usage
12	Interworking
13	Mesh ID
Last	Vendor Specific

Figura P14-A.1 *Cuerpo de un paquete probe request [1].*

acceso (*null probe request*) no lo contendrá.

En el caso de teléfonos inteligentes, éstos emiten *directed probe request,* cuyos SSID provienen de una lista de redes preferidas correspondiente a los puntos de acceso con los que se ha conectado en el pasado[4, 5].

Así mismo, la cabecera de los paquetes *probe request* contiene la dirección MAC de quien los emite y es precisamente esta dirección la que permite identificar y rastrear a los dispositivos WiFi. Sin embargo, como los dueños de los dispositivos WiFi que emiten los probe *request* no son conscientes de que están siendo rastreados, plantea problemas éticos sobre la obtención y uso de la información que se obtiene. Es por esto que los fabricantes de los celulares han implementado una técnica llamada MAC *randomization* para utilizar direcciones MAC falsas durante la emisión de *probe request* con la intención de que no puedan ser identificados, sin embargo, esta técnica ya ha sido vulnerada [1-4].

Si bien los módulos WiFi que incorporan el SoC ESP8266 y los ESP32 pueden rastrear/identificar teléfonos inteligentes WiFi mediante su modo promiscuo, por los problemas éticos involucrados se optó por desarrollar el *software* para emplear esta técnica únicamente con módulos WiFi. Por lo tanto, un módulo será capaz de emitir *probe request,* ya sea en todos los canales o en uno en específico, y no seguirá el proceso de conexión en caso de recibir un *probe response* (sonda de respuesta) por parte de algún punto de acceso. Así mismo, otro módulo será capaz de detectar estos *probe request* y verificar si la dirección MAC contenida en la cabecera de éste es una de las direcciones MAC que se deben rastrear.

La biblioteca ESP permite realizar las siguientes acciones:

- Emitir paquetes *probe request,* cada cierto tiempo (en milisegundos) para que la presencia del módulo sea detectada.
- Registrar hasta ocho direcciones MAC que serán detectadas.
- Realizar la detección en un canal en específico o haciendo un barrido en todos en intervalos de milisegundos.
- Imprimir en una terminal todas las direcciones MAC que se detecten en cierto número de segundos.
- Obtener el número de módulos que están emitiendo paquetes *probe request* durante cierto número de segundos.

Objetivo

Mostrar cómo utilizar el módulo WiFi para detectar la presencia de otros módulos WiFi.

Firmware de comandos AT

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION en el módulo que funcione cono estación y emita paquetes *probe request*,
- STATION y E_TRACKING en el módulo que funcione en modo promiscuo.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 2 interruptores DPST
- 7 resistores de 10 kΩ
- 3 botones pulsadores n.o.
- 2 condensador de 0.1 μF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 *mA*

Descripción

Se utilizarán dos módulos WiFi, uno funcionará como estación del cual se debe conocer su dirección MAC, mientras que el otro ejecutará su modo promiscuo, es decir, como una estación que será capaz de detectar la presencia de otros módulos. El módulo que funciona como estación comenzará a emitir paquetes *probe request* cuando se presione un botón asociado al uC PIC y dejará de emitirlos cuando el botón sea presionado nuevamente.

El módulo que ha activado su modo promiscuo estará detectando la presencia del módulo que funciona como estación, cuando detecte dicho módulo encenderá un LED por dos segundos.

Código

El código del módulo que funcionará como estación y que será rastreado se muestra en la figura P14-A.2, mientras que el módulo que funcionará en modo promiscuo se muestra en la figura P14-A.3, en el que se debe ingresar la dirección MAC del módulo que funciona como estación.

Circuito

El circuito del módulo que funciona como estación se muestra en la figura P14-A.4, mientras que el circuito del módulo que funciona en modo promiscuo se muestra en la figura P14-A.5.

Resultados

Se debe conocer previamente la dirección MAC del módulo WiFi que funcionará como estación y que será rastreado.

Cuando se ejecute el código de ambos microcontroladores, el LED del microcontrolador asociado al módulo que funciona en modo promiscuo no deberá estar encendido, lo cual significa que no se ha detectado la presencia del módulo que funciona como estación.

Cuando se presione el botón del microcontrolador asociado al módulo que funciona como estación, éste comenzará a enviar paquetes *probe request* y, por tanto, será detectada su presencia haciendo que el LED se encienda.

Cuando se vuelva a presionar el botón, se dejará de detectar su presencia y tras los dos segundos que permanecerá el LED encendido, no encenderá más.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2V0xsK8

Nota

- Cuando el módulo entra en modo promiscuo el único protocolo de comunicación que se puede implementar es ESPNOW.
- El módulo que funcione en modo promiscuo detectará a oros módulos que emitan paquetes *probe request* y que se encuentren dentro del área de cobertura como si funcionará como punto de acceso
- La emisión de paquetes *probe request* puede ser una tarea que se ejecute en segundo plano en el módulo WiFi.

Práctica propuesta

Se deberán utilizar varios módulos esparcidos por un área grande para reconstruir la trayectoria de un módulo que circule por ésta. Los módulos deberán comunicarse mediante ESPNOW y transmitir la información hacia un módulo central que será el que almacene el registro de toda la información.

```
#include <16lf1939.h>
#include "esp.h"

short ctr=false;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    /*Sube de nivel  para utilizar las funciones tracking y
     establece el módulo como estación (habilitar STATION en esp.h)*/
    up_to_level2();
    up_level2_to_level3();
    while(true)
    {
        //¿Se ha presionado el botón?
        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta soltar el botón
            }
            // habilita o deshabilita la emisón de paquetes probe request
            ctr=!ctr;
            if(ctr)
                enable_active_packet(100);
            else
                disable_active_packet();
        }
    }
}
```

Figura P14-A.2 *Código de la práctica 14-A, módulo como estación.*

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    /*Sube de nivel  para utilizar las funciones tracking y
     establece el módulo como estación (habilitar STATION en esp.h)*/
    up_to_level2();
    // Dirección MAC del módulo a detectar
    insert_mac_traking(1,"BC:DD:C2:26:20:F5");
    /* Se configura para analizar  en cada canal la presencia de paquetes
    probe request del dispositivo a rastrear. En cada canal pasará 25 ms*/
    enable_var_channel_traking(30);
    up_level2_to_level3();
    while(true)
    {
        //¿Se detectó el dispositivo?
        if(1==check_mac_traking(1))
        {
            output_bit(PIN_A2,TRUE);
            delay_ms(2000);
        }
        else
        {
            output_bit(PIN_A2,FALSE);
        }
    }
}
```

Figura P14-A.3 *Código de la práctica 14-A, módulo en modo promiscuo.*

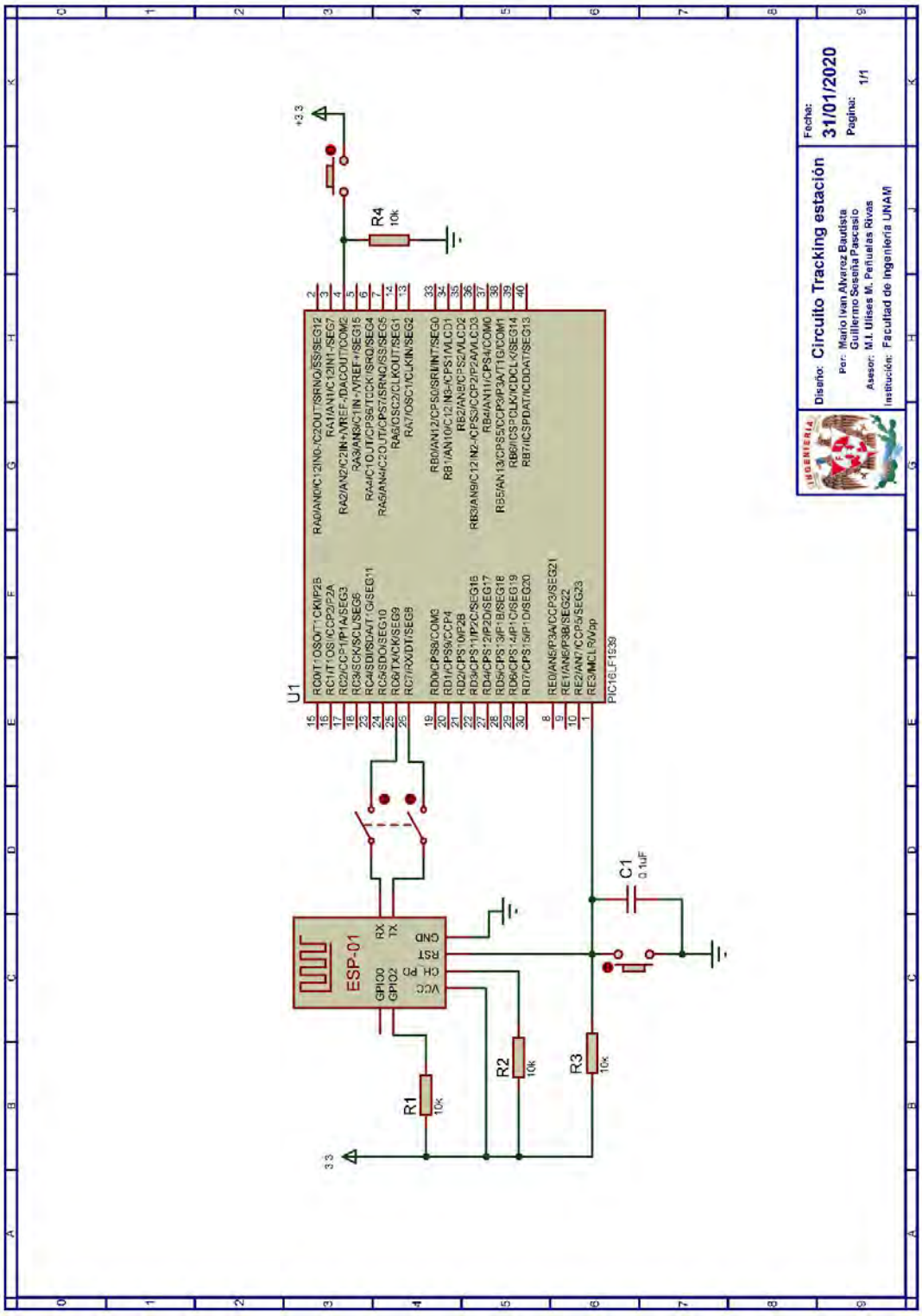


Figura P14-A.4 *Circuito de la práctica 14-A, módulo en modo estación.*

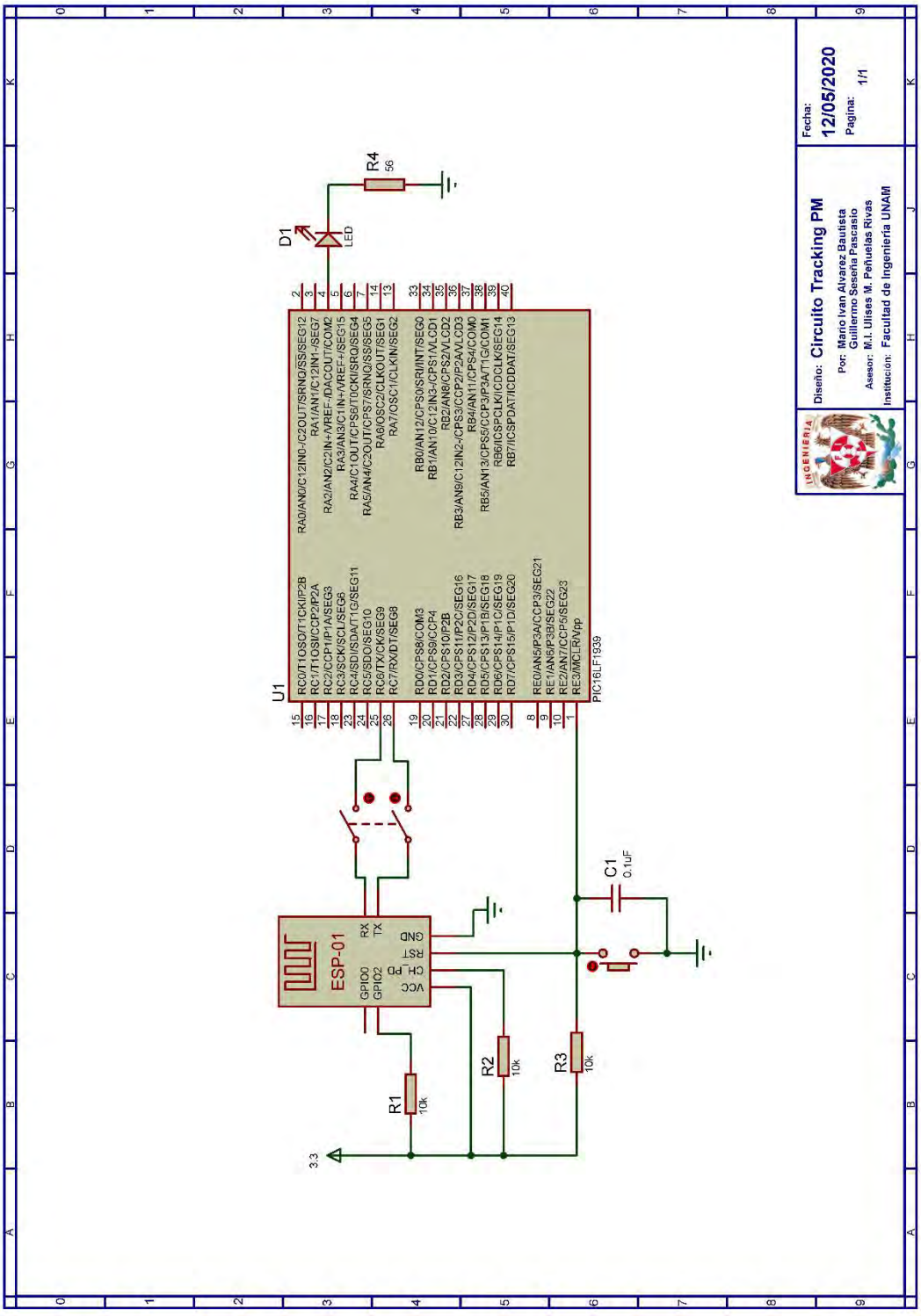


Figura P14-A.5 *Circuito de la práctica 14-A, módulo en modo promiscuo.*

Referencias

1. Luzio, A.D., A. Mei, and J. Stefa, *Mind Your Probes: De-Anonymization of Large Crowds Through Smartphone WiFi Probe Requests*. 2016, The 35th Annual IEEE International Conference on Computer Communications.
2. Pipelidis, G., et al., *HuMAn: Human Movement Analytics via WiFi Probes*. 2019, Demonstration on Pervasive Computing and Communications.
3. Naga, A., *Estimating the number of people in an area Using Bluetooth and WiFi signals*. 2019.
4. Oliveira, L. and J.d. Souza, *Mobile Device Detection Through WiFi Probe Request Analysis*. 2019.
5. Coleman, D.D. and D.D. Westcott, *CWNA® Certified Wireless Network Administrator Study Guide Exam CWNA-107*. Quinta ed. 2018, Indianapolis, Indiana: John Wiley & Sons.
6. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.

Práctica 14-B WiFi Tracking: conteo de módulos WiFi

Introducción

WiFi *Tracking* o rastreo WiFi hace referencia a los dispositivos que implementan la tecnología WiFi para detectar paquetes *probe request* que son emitidos por otros dispositivos WiFi en todos los canales, con la finalidad de obtener datos de las personas que usan éstos para:

- Determinar su localización
- Reconstruir o inferir sus trayectorias
- Estimar presencia.

Lo anterior basado en el hecho de que cada dispositivo detectado está asociado a una única persona. Para emplear WiFi *Tracking,* los dispositivos WiFi que se desean rastrear deben estar ejecutando lo que se denomina un escaneo activo para identificar redes disponibles, es decir, los dispositivos deben emitir paquetes *probe request* (sonda de solicitidud). WiFi *Tracking* es frecuentemente empleado para la detección teléfonos inteligentes, ya que la mayoría de éstos incorporan la tecnología WiFi y generalmente se encuentran ejecutando un escaneo activo. La detección de teléfonos inteligentes puede ser utilizado para ubicar a los propietarios de éstos dispositivos [2-4]. La información obtenida mediante WiFi *Tracking* tiene un alto valor comercial y social, ejemplos de en lo que se puede aprovechar se mencionan en [2] y [3].

El *probe request* pertenece a los *management frames* (marcos de gestion), los cuales son un conjunto de paquetes que están definidos por el estándar IEEE 802.11 y que son utilizados para la comunicación inicial (proceso de conexión) entre una estación y un punto de acceso [1, 5, 6]. El *probe request* es utilizado para detectar puntos de acceso cercanos, el cual contendrá los parámetros y capacidades de la estación (ver figura P14-B.1) y como es transmitido antes del proceso de asociación con un punto de acceso, el paquete no contiene ningún tipo de cifrado. Cuando la estación busca un punto de acceso específico (*directed probe request*), este paquete contendrá el SSID del punto de acceso, mientras que cuando busca cualquier punto de

Order	Informations
1	SSID
2	Supported Rates
3	Request Informations
4	Extended Supported Rates
5	DSSS Parameter Set
6	Supported Operating Classes
7	HT Capabilities
8	20/40 BSS Coexistence
9	Extended Capabilities
10	SSID List
11	Channel Usage
12	Interworking
13	Mesh ID
Last	Vendor Specific

Figura P14-B.1 *Cuerpo de un paquete probe request[1].*

acceso (*null probe request*) no lo contendrá.

En el caso de los celulares, éstos emiten *directed probe request,* cuyos SSID provienen de una lista de redes preferidas correspondiente a los puntos de acceso con los que se ha conectado en el pasado [4, 5].

Así mismo, la cabecera de los paquetes *probe request* contiene la dirección MAC de quien los emite y es precisamente esta dirección la que permite identificar y rastrear a los dispositivos WiFi. Sin embargo, como los dueños de los dispositivos WiFi que emiten los probe *request* no son conscientes de que están siendo rastreados, plantea problemas éticos sobre la obtención y uso de la información que se obtiene. Es por esto que los fabricantes de los celulares han implementado una técnica llamada MAC *randomization* para utilizar direcciones MAC falsas durante la emisión de *probe request* con la intención de que no puedan ser identificados, sin embargo, ésta técnica ya ha sido vulnerada [1-4].

Si bien los módulos WiFi que incorporan el SoC ESP8266 y los ESP32 pueden rastrear/identificar teléfonos inteligentes WiFi mediante su modo promiscuo, por los problemas éticos involucrados se optó por desarrollar el *software* para emplear esta técnica únicamente con módulos WiFi. Por lo tanto, un módulo será capaz de emitir *probe request,* ya sea en todos los canales o en uno en específico, y no seguirá el proceso de conexión en caso de recibir un *probe response* (sonda de respuesta) por parte de algún punto de acceso. Así mismo, otro módulo será capaz de detectar estos *probe request* y verificar si la dirección MAC contenida en la cabecera de éste es una de las direcciones MAC que se deben rastrear.

La biblioteca ESP permite realizar las siguientes acciones:

- Emitir paquetes *probe request,* cada ciertos milisegundos, para que la presencia del módulo sea detectada.
- Registrar hasta ocho direcciones MAC que serán detectadas.
- Realizar la detección en un canal en específico o haciendo un barrido en todos en intervalos de milisegundos.
- Imprimir en una terminal todas las direcciones MAC que se detecten en cierto número de segundos.
- Obtener el número de módulos que están emitiendo paquetes *probe request* durante cierto número de segundos.

Objetivo

Mostrar cómo utilizar un módulo WiFi para determinar el número de módulos WiFi que emiten paquetes *probe request*.

Firmware de comandos AT

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION en el módulo que funcione cono estación y emita paquetes *probe request*.
- STATION y E_TRACKING en el módulo que funcione en modo promiscuo.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 2 interruptores DPST
- 8 resistores de 10 kΩ
- 4 botones pulsadores n.o.
- 2 condensador de 0.1 µF

Descripción

Se utilizarán dos módulos WiFi, uno funcionará como estación mientras que el otro lo hará en su modo promiscuo. El módulo que funciona como estación comenzará a emitir paquetes *probe request* cuando se presione un botón asociado al uC PIC de éste y dejará de emitirlos cuando el botón sea presionado nuevamente.

El módulo que ha activado su modo promiscuo al presionar un botón comenzará a contabilizar a los módulos que emiten los paquetes *probe request* e imprimirá el número resultante en una terminal.

Código

El código del módulo que funcionará como estación y que será contabilizado se muestra en la figura P14-B.2, mientras que el módulo que funcionará en modo promiscuo se muestra en la figura P14-B.3.

Circuito

El circuito del módulo que funciona como estación se muestra en la figura P14-B.4, mientras que el circuito del módulo que funciona en modo promiscuo se muestra en la figura P14-B.5.

Resultados

Cuando se presione el botón del microcontrolador asociado al módulo que funciona como estación, éste comenzará a enviar paquetes *probe request* y, por tanto, será contabilizado por el módulo que funciona en modo promiscuo cuando

se presione el botón asociado a este último.

Cuando se vuelva a presionar el botón asociado al módulo como estación, éste dejará de emitir los paquetes *probe request* y, por tanto, no será contabilizado por el otro módulo.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento de la práctica:

https://bit.ly/2Nfy2PX

Nota

- Cuando el módulo entra en modo promiscuo el único protocolo de comunicación que se puede implementar es ESPNOW.
- El módulo que funcione en modo promiscuo detectará a otros módulos que emitan paquetes *probe request* y que se encuentren dentro del área de cobertura como si éste funcionara como punto de acceso.

```
#include <16lf1939.h>
#include "esp.h"

short ctr=false;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    /*Sube de nivel  para utilizar las funciones tracking y
     establece el módulo como estación (habilitar STATION en esp.h)*/
    up_to_level2();
    up_level2_to_level3();
    while(true)
    {
        //¿Se ha presionado el botón?
        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta soltar el botón
            }
            // habilita o deshabilita la emisón de paquetes probe request
            ctr=!ctr;
            if(ctr)
                enable_active_packet(100);
            else
                disable_active_packet();
        }
    }
}
```

Figura P14-B.2 *Código de la práctica 14-B, módulo como estación.*

```
#include <16lf1939.h>
#include "esp.h"

char num_mod=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    /*Sube de nivel  para utilizar las funciones tracking y
     establece el módulo como estación (habilitar STATION en esp.h)*/
    up_to_level2();
    up_level2_to_level3();
    while(true)
    {
        //¿Se ha presionado el botón?
        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta soltar el botón
            }
            /* Inicia el conteo de módulos que emitan paquetes probe request
             por tres segundos*/
            tracking_counter(3);
        }
        //Ya ha  acabdo con el conteo de módulos
        if(1==get_tracking_counter(&num_mod))
        {
            fprintf(PIC,"Detectados:%d\r\n",num_mod);
        }
    }
}
```

Figura P14-B.3 *Código de la práctica 14-B, módulo en modo promiscuo*

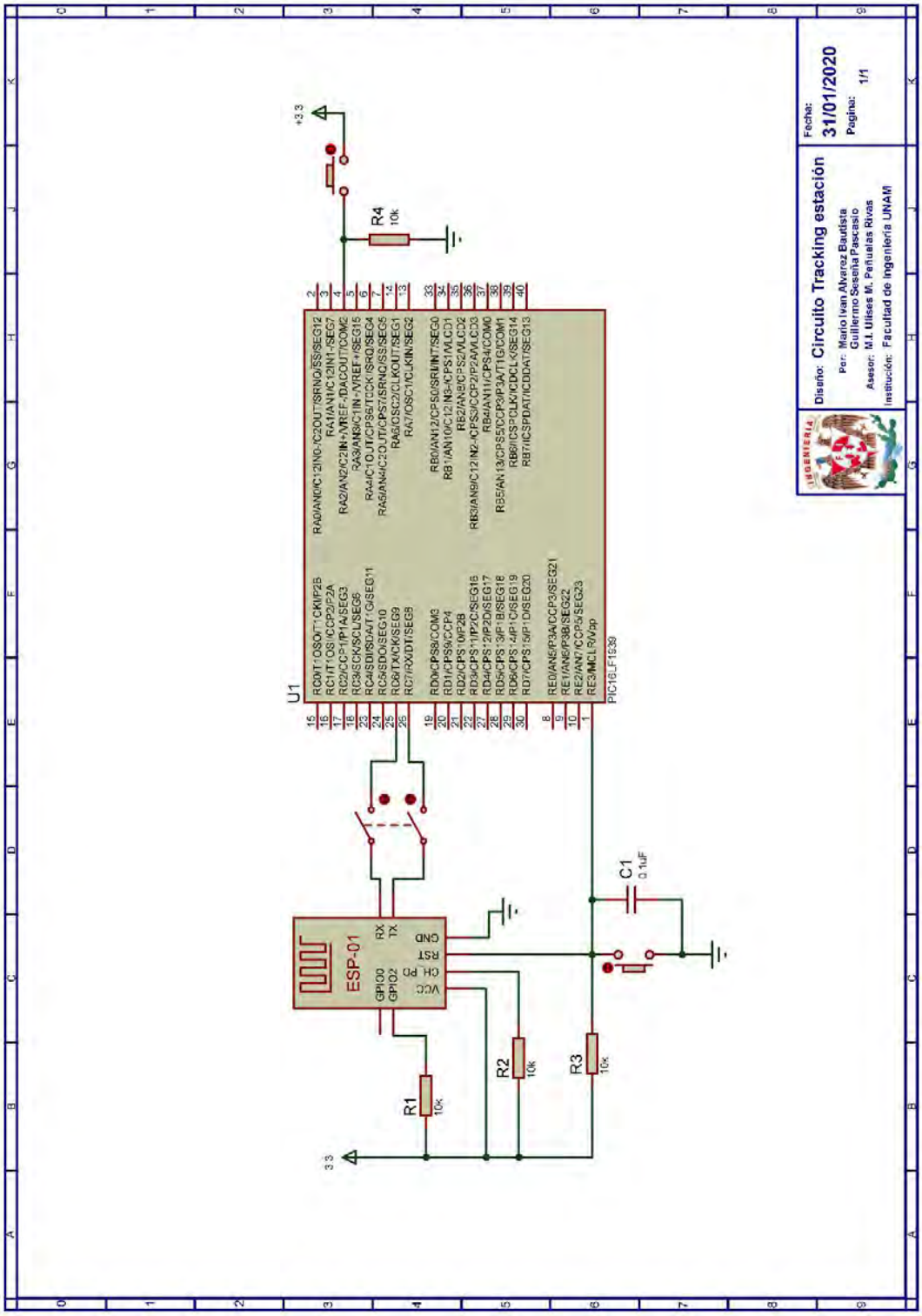


Figura P14-B.4 *Circuito de la práctica 14-B, módulo en modo estación.*

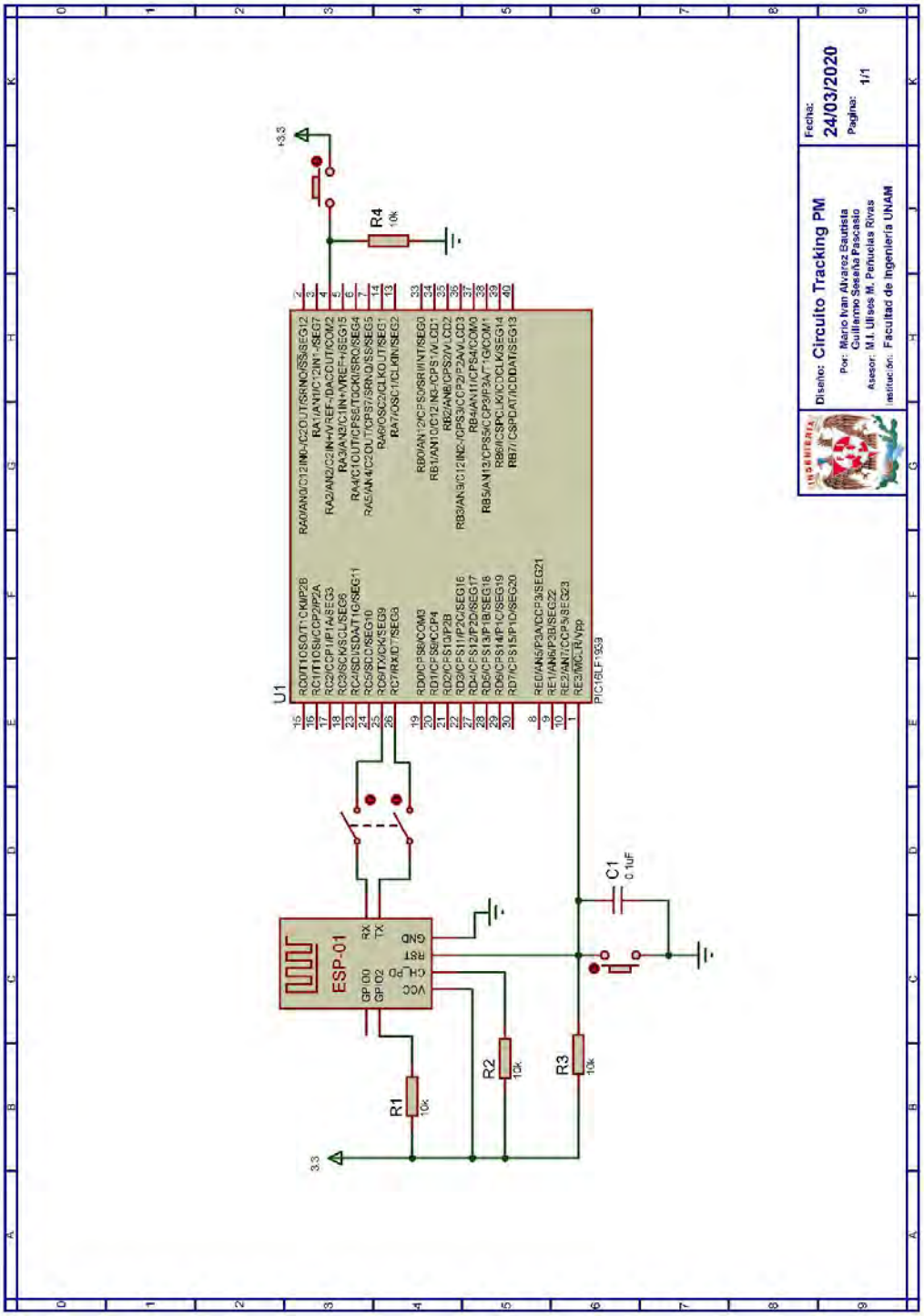


Figura P14-B.5 *Circuito de la práctica 14-B, módulo en modo promiscuo.*

Referencias

1. Luzio, A.D., A. Mei, and J. Stefa, *Mind Your Probes: De-Anonymization of Large Crowds Through Smartphone WiFi Probe Requests*. 2016, The 35th Annual IEEE International Conference on Computer Communications.
2. Pipelidis, G., et al., *HuMAn: Human Movement Analytics via WiFi Probes*. 2019, Demonstration on Pervasive Computing and Communications.
3. Naga, A., *Estimating the number of people in an area Using Bluetooth and WiFi signals*. 2019.
4. Oliveira, L. and J.d. Souza, *Mobile Device Detection Through WiFi Probe Request Analysis*. 2019.
5. Coleman, D.D. and D.D. Westcott, *CWNA® Certified Wireless Network Administrator Study Guide Exam CWNA-107*. Quinta ed. 2018, Indianapolis, Indiana: John Wiley & Sons.
6. Forouzan, B.A., *TCP/IP Protocol Suite*. 2010: McGraw Hill.

Práctica 15-A Funciones criptográficas: funciones hash

Introducción

Con un mundo cada vez más conectado que gira en torno del internet y con un mayor uso de las tecnologías inalámbricas, los riesgos de seguridad de la información han aumentado, ya que la información que se transmite puede ser fácilmente comprometida. La criptografía se ha convertido en una rama de la información utilizada para brindar la seguridad, principalmente en la transferencia de información confidencial, convirtiendo información sin formato en un mensaje codificado secreto, el cual si es interceptado no es legible por el captor. El proceso de conversión es conocido como cifrado y una vez que mensaje llega a su destino, se realiza sobre éste un proceso inverso conocido como descifrado para obtener el contenido original (ver figura P15-A.1) [1, 2].

Tanto para el cifrado como descifrado se utilizan algoritmos que hacen uso de una o más claves (*keys*), las cuales pueden ser palabras, números o frases, y dependiendo del algoritmo como de la clave seleccionada es la fortaleza del cifrado. Estos algoritmos, en función de sus claves, se clasifican en:

- Algoritmos de clave simétrica (Clave secreta)
- Algoritmos de clave asimétrica (Clave pública)
- Funciones hash

En los algoritmos de clave simétrica tanto el emisor como el destinatario utilizan las mismas claves para cifrar y descifrar mensajes. Estas “claves usadas para cifrar y descifrar pueden ser diferentes, pero hay una transformación para ir de una clave hacia la otra”. Estos algoritmos operan en dos modos, continuo (*stream)* en el cual cada bit es cifrado de manera independiente, y en bloque en el cual se cifra por bloques de datos [1, 2]. Entre los modos de operación que utiliza un cifrado por bloques se tienen:

- Libro de códigos electrónico *(Electronic Codebook,* ECB), en el

Figura P15-A.1 *Cifrado y descifrado[1]*.

cual cada bloque es cifrado de manera independiente (ver figura P15-A.2).

- Cadena de cifrado por bloques *(Cipher Block Chaining*, CBC), en el cual cada bloque se le aplica un XOR con el último bloque cifrado. Para esto se requiere de un bloque inicial, conocido como vector de inicialización (IV), para cifrar el primer bloque de datos (ver figura P15-A.2).

- Cifrado de realimentación *(Cipher Feedback,* CFB), el cual inicia cifrando el vector de inicialización y al resultado se le aplica un XOR junto con un bloque de datos. Este resultado es utilizado como el nuevo vector de inicialización para el siguiente bloque de datos (ver figura P15-A.2).

- Salida de realimentación (*Output Feedback* OFB), en el cual se cifra el vector de inicialización y al resultado se le aplica un XOR junto con un bloque de datos. El resultado de cifrar el vector de inicialización es utilizado como el nuevo vector de inicialización (ver figura P15-A.2).

En los algoritmos de clave asimétrica tanto el remitente como el destinatario cuentan con un par de claves que utilizan para cifrar y descifrar los mensajes. Por lo que cuando éstos quieren intercambiar mensajes de manera segura, se deben intercambiar previamente las claves con las que se cifrará la información para

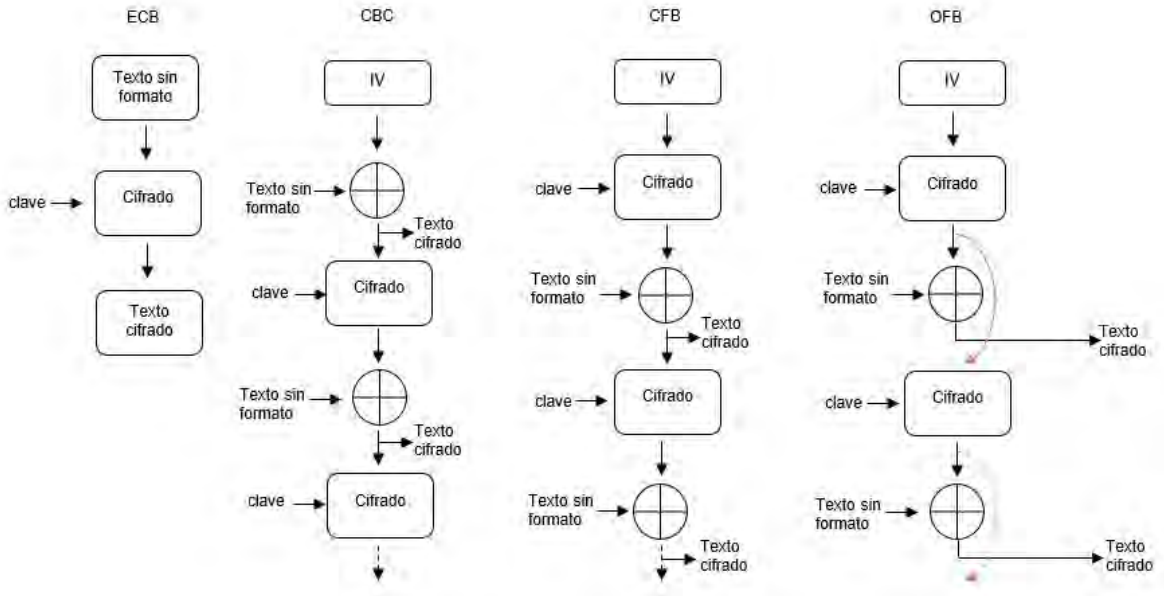


Figura P15-A.2 *Modos de cifrado de bloque. Modificado de [2].*

cada uno, las cuales se conocen como claves públicas. Cuando el remitente desea enviar un mensaje al destinatario, éste cifra la información con la clave pública del destinatario y una vez que el remitente recibe el mensaje utiliza una clave para descifrar la información, la cual es conocida como clave privada y que sólo es del conocimiento del destinatario. El mismo proceso se repite cuando el destinatario quiere enviar un mensaje al remitente [1, 2].

Las funciones hash, también conocidas como resúmenes de mensaje, toman como entrada un bloque de información de longitud variable y producen como salida un texto de longitud fija, conocido somo resumen de mensaje (*digest message*) o suma de verificación (*checksum*). Estas funciones tienen las siguientes propiedades:

- La salida es de tamaño fijo y no depende del tamaño de la entrada.
- El mensaje original no puede ser generado a partir del resumen de éste.
- Es muy poco probable que obtenga el mismo resumen a partir de dos entradas diferentes [1, 2].

Estas funciones suelen utilizarse para la comprobación de integridad de información, seguridad en procesos de identificación y en firmas digitales.

Los algoritmos criptográficos de clave simétrica que son soportados en la biblioteca ESP se muestran en la tabla P15-A.1, mientras que las funciones hash que son soportadas se muestran en la tabla P15-A.2.

Tabla P15-A.1 *Algoritmos de clave simétrica soportados.*

Algoritmo	**IV (bytes)**	**Claves (bytes)**	**Número de Claves**	**Bloque e/s (bytes)**
AES ECB	NA	16,24,32	1 para cifrar 1 para descifrar	16
AES CBC	16	16,24,32	1 para cifrar 1 para descifrar	16 y múltiplos de 16
ARC4	NA	1-58	1 para cifrar y descifrar	1-500
BLOWFISH ECB	NA	4-56	1 para cifrar y descifrar	8
BLOWFICH CBC	8	4-56	1 para cifrar y descifrar	8 y múltiplos de 8
XTEA ECB	NA	16	1 para cifrar y descifrar	8
XTEA CBC	8	16	1 para cifrar y descifrar	8 y múltiplos de 8
CAMELLIA ECB	NA	16,24,32	1 para cifrar 1 para descifrar	16
CAMELLIA CBC	16	16,24,32	1 para cifrar 1 para descifrar	16 y múltiplos de 16
AES CFB128	16	16	1 para cifrar y descifrar	16
AES CFB8	16	16,24,32	1 para cifrar y descifrar	16

Tabla P15-A.2 *Funciones hash soportadas.*

Algoritmo	Entrada (bytes)	Salida (bytes)
SHA1	500	20
SHA254	500	28
SHA256	500	32
SHA384	500	48
SHA512	500	64
MD5	500	16
MD4	500	16

Al establecer el tamaño de la clave para cifrar o descifrar se determinará sí el algoritmo será de 16, 24 o 32 bits.

Cabe señalar que ninguno de los métodos criptográficos de cifrado en bloque cuenta con algún método de relleno (*padding*) por lo que se debe ingresar el tamaño de bloque completo o los múltiplos de éste.

Objetivo

Mostrar cómo utilizar las funciones hash.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

STATION, E_TCP_SERVER y E_CRYPTOGRAPHIC_FUNC

Material

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF
- 1 adaptador USB-TLL

Descripción

Se utilizará un módulo WiFi en el cual se creará un servidor TCP en el puerto 80. Este servidor responderá a cualquier cliente que se conecte con un mensaje secreto. Sin embargo, para acceder a dicho mensaje el cliente deberá autenticarse previamente con el servidor, es decir, debe enviar su usuario y contraseña, siendo esta última a la que se le aplicará un SHA1.

El servidor tendrá un registro de un único usuario permitido y el SHA1 de la contraseña de éste, con la cual se comparará con la que será calculada con la contraseña que se reciba Si el usuario no es autenticado será desconectado. El cliente TCP será creado en el programa

Packet Sender y deberá seguir las instrucciones que indique el servidor hasta recibir el mensaje secreto “el número mágico es 4”.

El almacenar el SHA de una contraseña es una práctica común para evitar que, si un usuario malicioso logra acceder al registro de las contraseñas, no pueda obtener el texto que origina el SHA registrado y por tanto acceder al sistema.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P15-A.6 y figura P15-A.7, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito de esta práctica se muestra en la figura P15-A.8 y.

Resultados

Tanto el módulo WiFi como la computadora donde se ejecutará el programa *packet sender* se deben conectar al mismo punto de acceso.

Al iniciar la ejecución del programa del microcontrolador PIC, en la terminal se mostrará un mensaje con la dirección IP para acceder al servidor TCP que se crea en el módulo WiFi. Se deberá acceder al dicho servidor con un cliente TCP creado en el programa *packet sender*, el cual al conectarse recibirá un mensaje pidiendo que envíe el nombre de usuario (ver figura P15-A.5).

Figura P15-A.5 *Solicitud de ingreso de usuario.*

Desde el cliente se deberá mandar el usuario “ALK” y se recibirá un mensaje pidiendo la contraseña (ver figura P15-A.4), en caso de mandar otro nombre, el cliente TCP será desconectado del servidor.

Figura P15-A.4 *Solicitud de ingreso de contraseña.*

Cuando se solicite la contraseña, se deberá enviar “UNAM” para recibir el mensaje secreto, en caso de que se envíe otra contraseña (ver figura P15-A.3), el cliente TCP será desconectado.

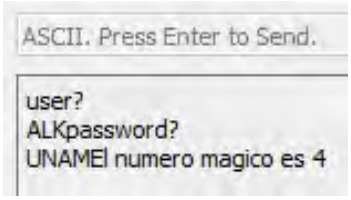


Figura P15-A.3 *Obtención del mensaje secreto.*

```
#include <16lf1939.h>
#include "esp.h"

extern volatile _bool new_message_server1;
/*SHA1 de la palabra UNAM*/
const char password_user[]={0xc6,0xfa,0x18,0x6d,0xbe,0x72,0x17,0x1b,0x0c,0x6d,0xb8
0x5b,0xd9,0x47,0x06,0x34,0xe0,0xba,0x9f,0x38};
#define user "ALK"
char server_buffer[21]={0};
char buffer_sha[21]={0};
int i=0;

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso especificado
    begin_station("ESP","SSID","123456789");
    //Obtención de la IP de la estación
    get_ip_station(server_buffer,sizeof(server_buffer));
    fprintf(PIC,"IP:%s\r\n",server_buffer);
    // Establece el buffer del servidor para recibir mensaje
    set_buffer_server1(server_buffer,sizeof(server_buffer));
    // Crea un servidor TCP  en el puerto 80, con un sólo cliente
    create_server1(80,1,500);
    //Etiqueta para reinicar el proceso tras usuario o contraseña incorrectos
    inicio:
    while (1)
    {
        // Espera a que se conecte un cliente TCP
        if(1==client_connected(0))
        {
            new_message_server1=0;
            break;
        }

    }
    // Envia un mensaje solicitando el usuario al cliente TCP
    send_server1(0,"user?\r\n");
```

Figura P15-A.6 *Primera parte del código de la práctica 15-A.*

```
    while (1)
    {
        if(new_message_server1==1)
        {
            new_message_server1=0;
            //¿Coincide el usuario recibido con el almacenado?
            if(compare_S1buffer(user,1))
                break;
            else
            {
                // no coincidió, se desconecta al cliente
                close_client_server1(0);
                goto inicio;
            }
        }
        if(0==client_connected(0))
            goto inicio;
    }
    //Se reinicia bandera  de recepción de canales
    new_message_server1=0;
    // Envia un mensaje solicitando la contraseña al cliente TCP
    send_server1(0,"password?\r\n");
    while(1)
    {
        //¿Se recibió  un mensaje desde el cliente TCP?
        if(new_message_server1==1)
        {
            // Se aplica el SHA1 al mensaje recibido
            cryptographic(1,get_str_length(server_buffer),server_buffer, buffer_sha);
            for(i=0;i<20;i++)
            {
               if(password_user[i]!=buffer_sha[i])
               {
                   // no coincidió, se desconecta al cliente TCP.
                    close_client_server1(0);
                    goto inicio;
               }
            }
            // Coinciden ambos SHA, se entrega meensaje
            send_server1(0,"El numero magico es 4");
            //Se termina la conexión con el cliente TCP
            close_client_server1(0);
            goto inicio;
        }
        if(0==client_connected(0))
            goto inicio;
    }
    while(1)
    {
    }
}
```

Figura P15-A.7 *Segunda parte del código de la práctica 15-A.*

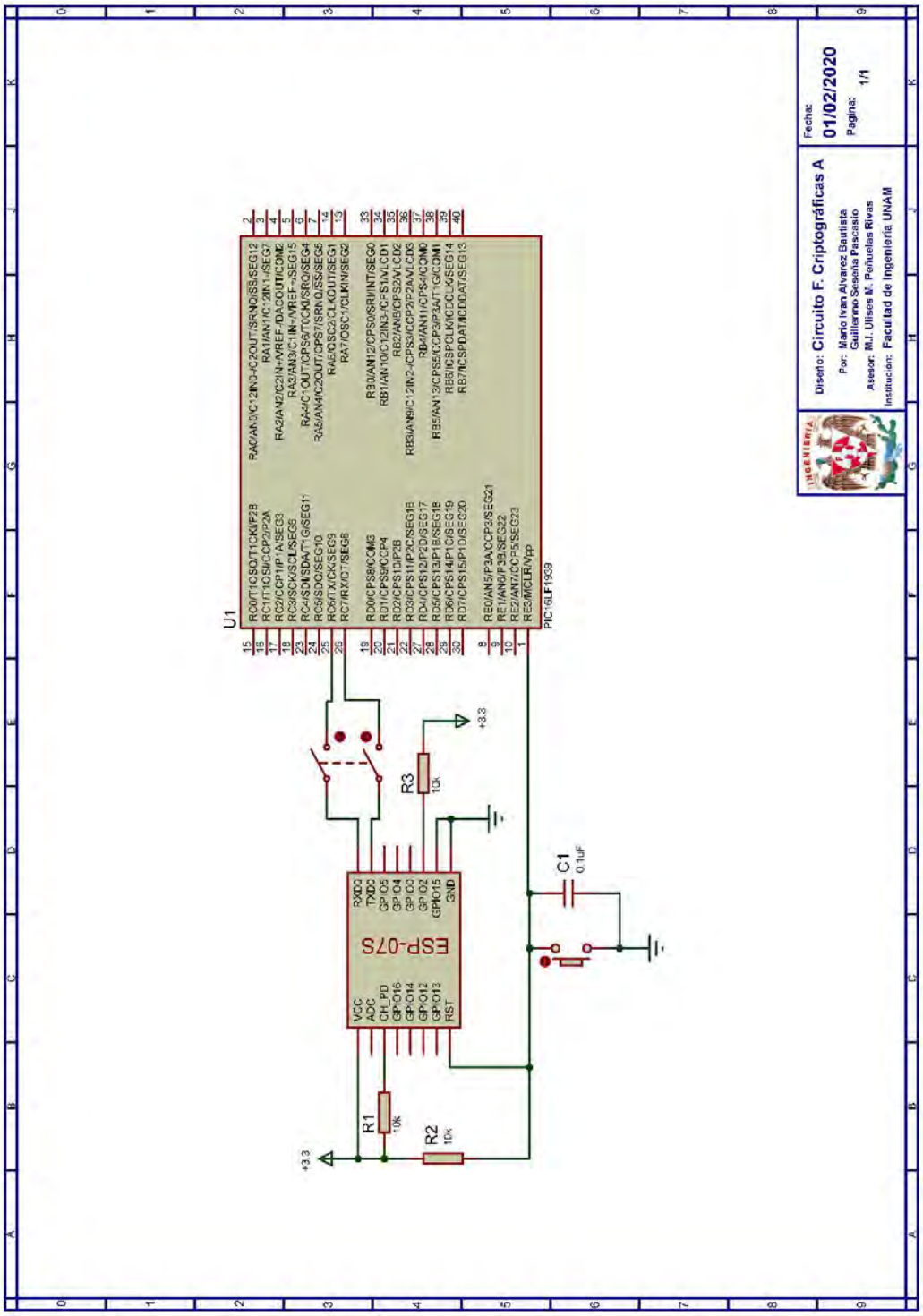


Figura P15-A.8 *Circuito de la práctica 15-A.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3hLNQrN

Notas

- Para cumplir con el tamaño del bloque, los mensajes deben ser rellenados con algún carácter.
- Se recomienda utilizar el programa Hterm para simular la terminal.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- Esta aplicación puede ejecutarse con el módulo en estación o punto de acceso.
- Con los métodos ejecutados en el módulo WiFi, se evita que se consuma una gran cantidad de memoria en el uC PIC y se realiza en un tiempo menor.

Referencias

1. Nayak, U. and U.H. Rao, *The InfoSec Handbook. An introduction to informaction Security*. 2014: Apress.
2. Werstén, B., *Implementing the Transport Layer Security Protocol for Embedded Systems.* 2007.

Práctica 15-B Funciones criptográficas: cifrado y descifrado de clave simétrica

Introducción

Con un mundo cada vez más conectado que gira en torno del internet y con un mayor uso de las tecnologías inalámbricas, los riesgos de seguridad de la información han aumentado, ya que la información que es transmitida puede ser fácilmente comprometida. La criptografía se ha convertido en una rama de la información utilizada para brindar la seguridad, principalmente en la transferencia de información confidencial, convirtiendo información sin formato en un mensaje codificado secreto, el cual si es interceptado no es legible por el captor. El proceso de conversión es conocido como cifrado y una vez que mensaje llega a su destino, se realiza sobre éste un proceso inverso conocido como descifrado para obtener el contenido original (ver figura P15-B.1) [1, 2].

Tanto para el cifrado como descifrado se utilizan algoritmos que hacen uso de una o más claves (*keys*), las cuales pueden ser palabras, números o frases, y dependiendo del algoritmo como de la clave seleccionada es la fortaleza del cifrado. Estos algoritmos, en función de sus claves, se clasifican en:

- Algoritmos de clave simétrica (Clave secreta)
- Algoritmos de clave asimétrica (Clave pública)
- Funciones hash.

En los algoritmos de clave simétrica tanto el emisor como el destinatario utilizan las mismas claves para cifrar y descifrar mensajes. Estas “claves usadas para cifrar y descifrar pueden ser diferentes, pero hay una transformación para ir de una clave hacia la otra”. Estos algoritmos operan en dos modos, continuo (*stream)* en el cual cada bit es cifrado de manera independiente, y en bloque en el cual se cifra por bloques de datos[1, 2]. Entre los modos de operación que utiliza un cifrado por bloques se tienen:

- Libro de códigos electrónico *(Electronic Codebook,* ECB), en el cual cada bloque es cifrado de

Figura P15-B.1 *Cifrado y descifrado[1].*

manera independiente (ver figura P15-B.2).

- Cadena de cifrado por bloques *(Cipher Block Chaining*, CBC), en el cual cada bloque se le aplica un XOR con el último bloque cifrado. Para esto se requiere de un bloque inicial, conocido como vector de inicialización (IV), para cifrar el primer bloque de datos (ver figura P15-B.2).

- Cifrado de realimentación *(Cipher Feedback,* CFB), el cual inicia cifrando el vector de inicialización y al resultado se le aplica un XOR junto con un bloque de datos. Este resultado es utilizado como el nuevo vector de inicialización para el siguiente bloque de datos (ver figura P15-B.2).

- Salida de realimentación (*Output Feedback* OFB), en el cual se cifra el vector de inicialización y al resultado se le aplica un XOR junto con un bloque de datos. El resultado de cifrar el vector de inicialización es utilizado como el nuevo vector de inicialización (ver figura P15-B.2).

En los algoritmos de clave asimétrica tanto el remitente como el destinatario cuentan con un par de claves que utilizan para cifrar y descifrar los mensajes[1, 2]. Por lo que cuando éstos quieren intercambiar mensajes de manera segura, se deben intercambiar previamente las claves con las que se

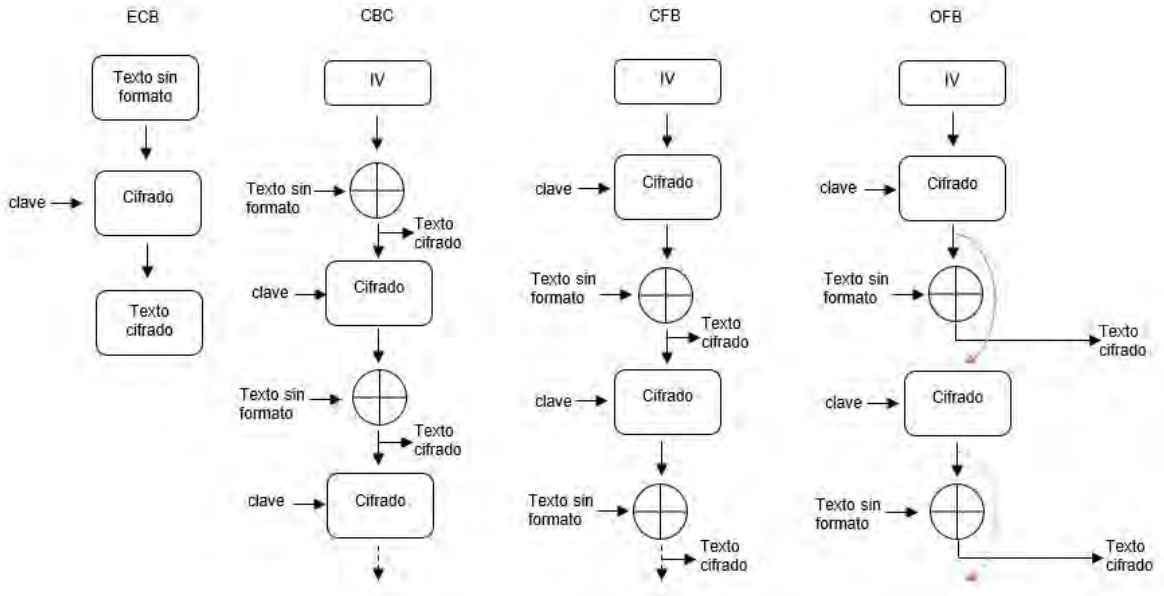


Figura P15-B.2 *Modos de cifrado de bloque. Modificado de [2].*

cifrará la información para cada uno, las cuales se conocen como claves públicas. Cuando el remitente desea enviar un mensaje al destinatario, éste cifra la información con la clave pública del destinatario y una vez que el remitente recibe el mensaje utiliza una clave para descifrar la información, la cual es conocida como clave privada y que sólo es del conocimiento del destinatario [1, 2]. El mismo proceso se repite cuando el destinatario quiere enviar un mensaje al remitente.

Las funciones hash, también conocidas como resúmenes de mensaje, toman como entrada un bloque de información de longitud variable y producen como salida un texto de longitud fija, conocido somo resumen de mensaje (*digest message*) o suma de verificación (*checksum*). Estas funciones tienen las siguientes propiedades:

- La salida es de tamaño fijo y no depende del tamaño de la entrada.
- El mensaje original no puede ser generado a partir del resumen de éste.
- Es muy poco probable que obtenga el mismo resumen a partir de dos entradas diferentes [1, 2].

Estas funciones suelen utilizarse para la comprobación de integridad de información, seguridad en procesos de identificación y en firmas digitales.

Los algoritmos criptográficos de clave simétrica que son soportados en la biblioteca ESP se muestran en la tabla P15-B.1, mientras que las funciones hash que son soportadas se muestran en la tabla P15-B.2.

Tabla P15-B.1 *Algoritmos de clave simétrica soportados.*

Algoritmo	**IV (bytes)**	**Claves (bytes)**	**Número de Claves**	**Bloque e/s (bytes)**
AES ECB	NA	16,24,32	1 para cifrar 1 para descifrar	16
AES CBC	16	16,24,32	1 para cifrar 1 para descifrar	16 y múltiplos de 16
ARC4	NA	1-58	1 para cifrar y descifrar	1-500
BLOWFISH ECB	NA	4-56	1 para cifrar y descifrar	8
BLOWFICH CBC	8	4-56	1 para cifrar y descifrar	8 y múltiplos de 8
XTEA ECB	NA	16	1 para cifrar y descifrar	8
XTEA CBC	8	16	1 para cifrar y descifrar	8 y múltiplos de 8
CAMELLIA ECB	NA	16,24,32	1 para cifrar 1 para descifrar	16
CAMELLIA CBC	16	16,24,32	1 para cifrar 1 para descifrar	16 y múltiplos de 16
AES CFB128	16	16	1 para cifrar y descifrar	16
AES CFB8	16	16,24,32	1 para cifrar y descifrar	16

Tabla P15-B.2 *Funciones hash soportadas.*

Algoritmo	Entrada (bytes)	Salida (bytes)
SHA1	500	20
SHA254	500	28
SHA256	500	32
SHA384	500	48
SHA512	500	64
MD5	500	16
MD4	500	16

El establecimiento del tamaño de la clave para cifrar o descifrar determinará sí el algoritmo será de 16, 24 o 32 bits.

Cabe mencionar que ninguno de los métodos criptográficos de cifrado en bloque cuenta con algún método de relleno (*padding*) por lo que se debe ingresar el bloque completo

Objetivo

Mostrar cómo utilizar los algoritmos criptográficos de clave simétrica con clave de tamaño fijo.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

- ACCESS_POINT, E_TCP_SERVER y E_CRYPTOGRAPHIC_FUNC en el módulo que funcione como servidor.
- STATION, E_SSL_TCP_UDP_CLIENT_UNICON y E_CRYPTOGRAPHIC_FUNC en el módulo que funcione como cliente.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 3 interruptores DPST
- 7 resistores de 10 kΩ
- 3 botones pulsadores n.o.
- 2 condensadores de 0.1 µF

Descripción

Se utilizarán dos módulos WiFi, uno funcionará como servidor TCP y AP, mientras que el otro funcionará como cliente TCP y estación. El servidor es creado en el módulo que funciona como AP para establecer una dirección IP personalizada que tomará el servidor, razón por la cual ya no se requiere la terminal para conocer la dirección IP a la que se debe conectar el cliente.

Cada vez que se presione un botón en el microcontrolador PIC asociado al cliente TCP, éste enviará el mensaje “hola mundo” al servidor TCP, pero este mensaje estará cifrado con AES ECB de 16 bytes (Para completar el tamaño del bloque, el mensaje se rellenará con espacios). El servidor al recibir el mensaje deberá imprimir el mensaje cifrado en una terminal, posteriormente lo descifrará e imprimirá el contenido del mensaje en la misma terminal.

Código

El código del microcontrolador asociado al módulo que funcionará como cliente TCP se muestra en figura P15-B.5, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña del AP creado por el otro módulo.
- Dirección IP y puerto del servidor credo por el otro módulo.

Mientras que el código del microcontrolador asociado al servidor TCP se muestra en la figura P15-B.6, en el que se deben ingresar los siguientes parámetros:

- Nombre que tomará el AP y su contraseña.
- Establecer dirección IP del AP que será la misma que la del servidor y el puerto.

Circuito

El circuito del módulo que funcionará como servidor TCP se muestra en la figura P15-B.7, mientras que el circuito del módulo que funcionará como cliente TCP se muestra en la figura P15-B.8.

Resultados

Se ejecutará primero el código del microcontrolador asociado al módulo que creará el servidor TCP. Al aparecer el mensaje de “*ready*” en la terminal (ver figura P15-B.4), significará que el servidor TCP ha sido creado, momento en el que se deberá ejecutar el código del microcontrolador asociado al módulo que funcionará como cliente TCP.

```
Moodulo esp listo para su uso
ready
```

Figura P15-B.4 *Mensaje para iniciar el código del uC PIC asociado al cliente TCP.*

Cuando en la terminal aparezca el mensaje “*connected*” *se* presionará el botón del microcontrolador asociado al módulo que funciona como cliente TCP para realizar el cifrado y el envío de la información. En la terminal se deberá imprimir tanto el mensaje cifrado como descifrado (ver figura P15-B.3).

```
connected
Cifrado:
o'o-oooo^HovEooo
Descifrado:
hola mundo
```

Figura P15-B.3 *Impresión del mensaje recibido*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2BntmVz

Notas

- En el circuito de la práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- Con los métodos ejecutados en el módulo WiFi, se evita que se consuma una gran cantidad de memoria en el uC PIC y se realiza en un tiempo menor.

```
#include <16lf1939.h>
#include "esp.h"

char message[]={"hola mundo     "};
char crypto[17]={0};
// Llave para los métodos criptográficos
int key[16]={0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1};
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    // Conexión al punto de acceso especificado
    begin_station("ESP","AP_FC","123456789");
    /* Se establece la llave para las funciones criptográficas, el
     método seleccionado determina si es utilizada para cirar o descifrar*/
    set_key(ENC,key,sizeof(key));
    // Conexion al servidor TCP
    create_tcp_pclient("192.168.4.1", 100);
    while(1)
    {
        if(1==input_state(PIN_A2)) // ¿Se ha presionado el botón?
        {
            while(1==input_state(PIN_A2))
            {
              // Espera a que se deje de preionar el botón
            }
            // Se cifra el mensaje  con e método seleccionado
            cryptographic(M_AES_ECB_E,sizeof(message), message,crypto);
            // Se envía el mensaje cifrado al servidor
            send_tcp(crypto);
        }
    }
}
```

Figura P15-B.5 *Código de la práctica 15-B, módulo como cliente TCP.*

```
#include <16lf1939.h>
#include "esp.h"

short flag=0; // Bandera de conexión
char server_buffer[20]={0};
char crypto[17]={0};
// Llave para el método criptográfico
int key[16]={0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1};
extern _bool new_message_server1;
extern uint16 message_server_length;

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //creación de un punto de acceso
    begin_ap(2,"AP_FC","123456789");
    //Establece la dirección IP del AP
    set_ip_ap("192.168.4.1");
    //Establece el buffer del servidor TCP para recibir mensajes
    set_buffer_server1(server_buffer,sizeof(server_buffer));
    // Crea un servidor TCP, con un único cliente en el puerto 100
    create_server1(100,1,900);
    fprintf(PIC, "ready\r\n");
    /* Se establece la llave para las funciones criptográficas, el
     método seleccionado determina si es utilizada para cirar o descifrar*/
    set_key(DEC,key,sizeof(key));
    new_message_server1=0;
    while(1)
    {
        if(1==client_connected(5) && flag==0)
        {
            delay_ms(1000);
            fprintf(PIC,"connected\r\n");
            flag=1;
        }
        if(0==client_connected(5))
            flag=0;
        //¿Se recibió un nuevo mensaje?
        if(1==new_message_server1)
        {
            // Se decifra el mensaje
            cryptographic(M_AES_ECB_D,message_server_length, server_buffer,crypto);
            delay_ms(10);
            fprintf(PIC,"Cifrado:\r\n");
            fprintf(PIC,"%s\r\n",server_buffer);
            fprintf(PIC,"Descifrado:\r\n");
            fprintf(PIC,"%s\r\n",crypto);
            new_message_server1=0; // Se limpia la bandera de nuevo mensaje
        }
    }
}
```

Figura P15-B.6 *Código de la práctica 15-B, módulo como servidor TCP.*

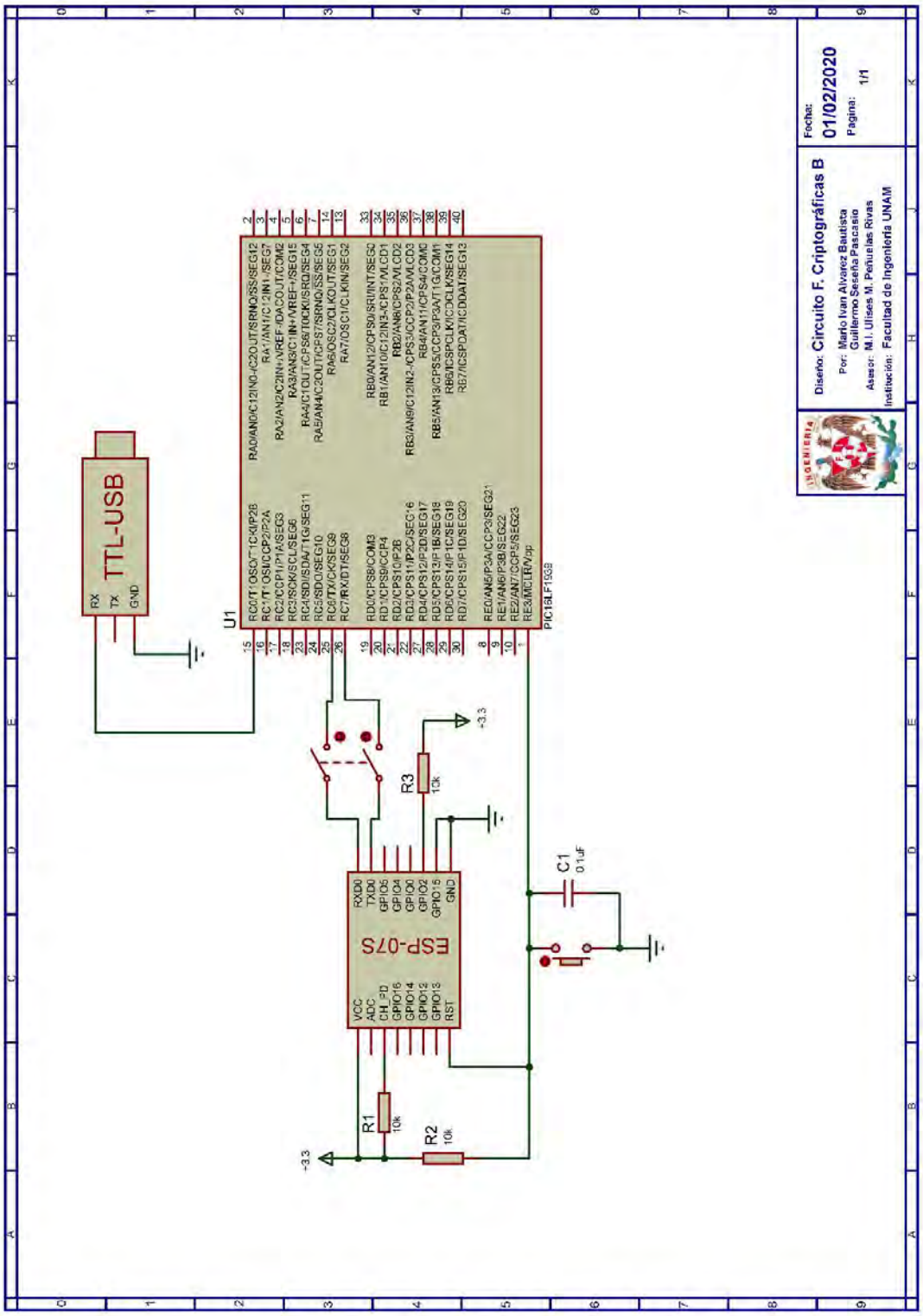


Figura P15-B.7 *Circuito de la práctica 15-B, módulo como servidor TCP.*

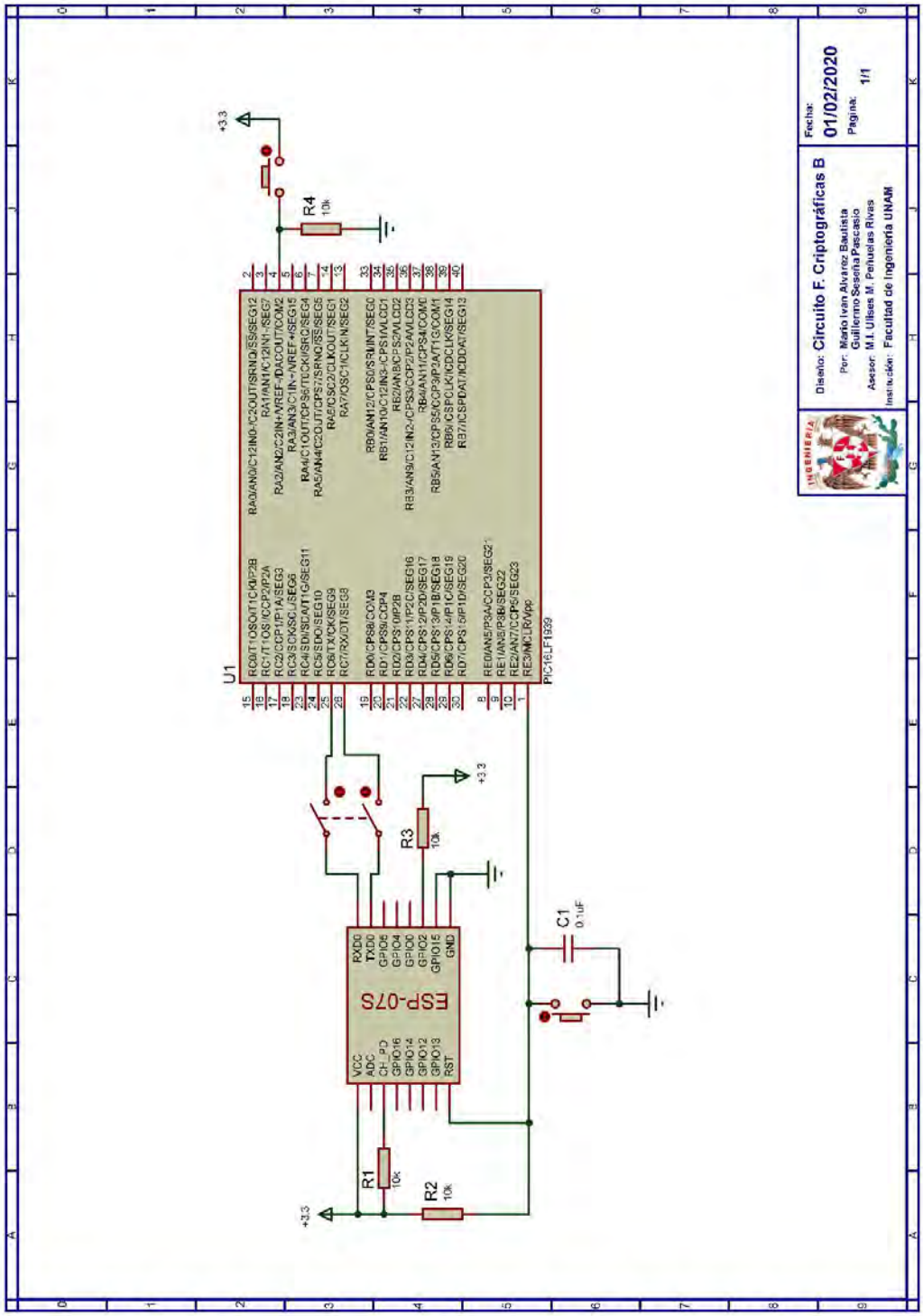


Figura P15-B.8 *Circuito de la práctica 15-B, módulo como cliente TCP.*

Referencias

1. Nayak, U. and U.H. Rao, *The InfoSec Handbook. An introduction to informaction Security*. 2014: Apress.
2. Werstén, B., *Implementing the Transport Layer Security Protocol for Embedded Systems.* 2007.

Práctica 15-C Funciones criptográficas: cifrado y descifrado de clave variable

Introducción

Con un mundo cada vez más conectado que gira en torno del internet y con un mayor uso de las tecnologías inalámbricas, los riesgos de seguridad de la información han aumentado, ya que la información que es puede ser fácilmente comprometida. La criptografía se ha convertido en una rama de la información utilizada para brindar la seguridad, principalmente en la transferencia de información confidencial, convirtiendo información sin formato en un mensaje codificado secreto, el cual si es interceptado no es legible por el captor. El proceso de conversión es conocido como cifrado y una vez que mensaje llega a su destino, se realiza sobre éste un proceso inverso conocido como descifrado para obtener el contenido original (ver figura P15-C.1) [1, 2].

Tanto para el cifrado como descifrado se utilizan algoritmos que hacen uso de una o más claves (*keys*), las cuales pueden ser palabras, números o frases, y dependiendo del algoritmo como de la clave seleccionada es la fortaleza del cifrado. Estos algoritmos, en función de sus claves, se clasifican en:

- Algoritmos de clave simétrica (Clave secreta)
- Algoritmos de clave asimétrica (Clave pública)
- Funciones hash[1, 2].

En los algoritmos de clave simétrica tanto el emisor como el destinatario utilizan las mismas claves para cifrar y descifrar mensajes. Estas “claves usadas para cifrar y descifrar pueden ser diferentes, pero hay una transformación para ir de una clave hacia la otra”. Estos algoritmos operan en dos modos, continuo (*stream)* en el cual cada bit es cifrado de manera independiente, y en bloque en el cual se cifra por bloques de datos [1, 2]. Entre los modos de operación que utiliza un cifrado por bloques se tienen:

- Libro de códigos electrónico *(Electronic Codebook,* ECB), en el

Figura P15-C.1 *Cifrado y descifrado [1].*

cual cada bloque es cifrado de manera independiente (ver figura P15-C.2).

- Cadena de cifrado por bloques *(Cipher Block Chaining*, CBC), en el cual cada bloque se le aplica un XOR con el último bloque cifrado. Para esto se requiere de un bloque inicial, conocido como vector de inicialización (IV), para cifrar el primer bloque de datos (ver figura P15-C.2).

- Cifrado de realimentación *(Cipher Feedback,* CFB), el cual inicia cifrando el vector de inicialización y al resultado se le aplica un XOR junto con un bloque de datos. Este resultado es utilizado como el nuevo vector de inicialización para el siguiente bloque de datos (ver figura P15-C.2).
- Salida de realimentación (*Output Feedback* OFB), en el cual se cifra el vector de inicialización y al resultado se le aplica un XOR junto con un bloque de datos. El resultado de cifrar el vector de inicialización es utilizado como el nuevo vector de inicialización (ver figura P15-C.2).

En los algoritmos de clave asimétrica tanto el remitente como el destinatario cuentan con un par de claves que utilizan para cifrar y descifrar los mensajes. Por lo que cuando éstos quieren intercambiar mensajes de manera segura, se deben intercambiar previamente las claves con las que se cifrará la información para cada uno, las cuales se conocen como

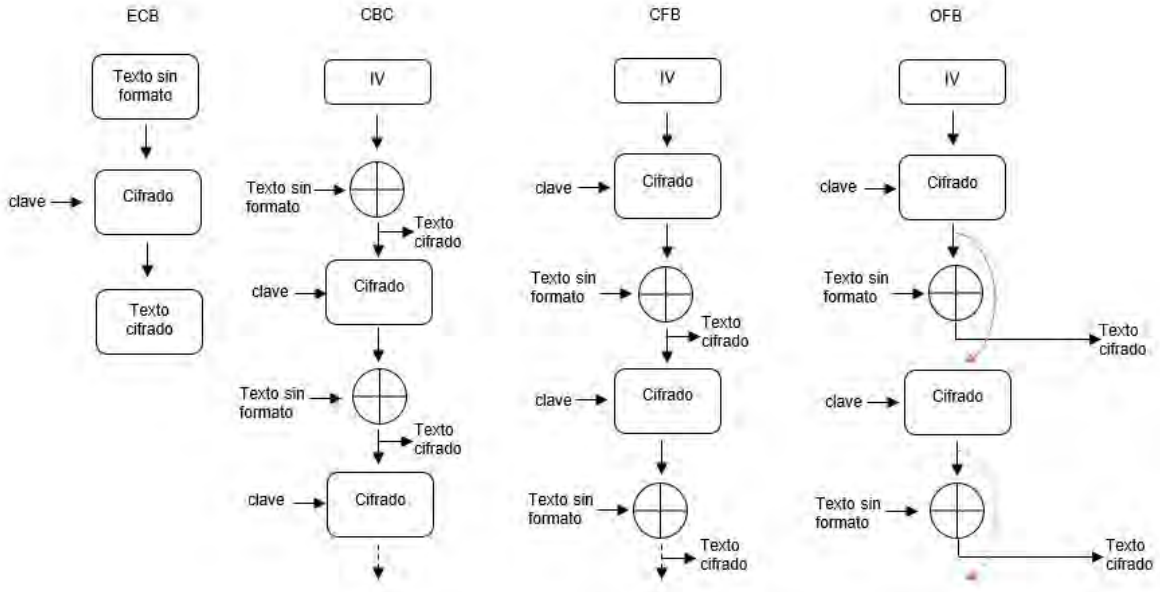


Figura P15-C.2 *Modos de cifrado de bloque. Modificado de [2].*

claves públicas [1, 2]. Cuando el remitente desea enviar un mensaje al destinatario, éste cifra la información con la clave pública del destinatario y una vez que el remitente recibe el mensaje utiliza una clave para descifrar la información, la cual es conocida como clave privada y que sólo es del conocimiento del destinatario. El mismo proceso se repite cuando el destinatario quiere enviar un mensaje al remitente.

Las funciones hash, también conocidas como resúmenes de mensaje, toman como entrada un texto de longitud variable y producen como salida un texto de longitud fija, conocido somo resumen de mensaje (*digest message*) o suma de verificación (*checksum*) [1, 2]. Estas funciones tienen las siguientes propiedades:

- La salida es de tamaño fijo y no depende del tamaño de la entrada.
- El mensaje original no puede ser generado a partir del resumen de éste.
- Es muy poco probable que obtenga el mismo resumen a partir de dos entradas diferentes [1, 2].

Estas funciones suelen utilizarse para la comprobación de integridad de información, seguridad en procesos de identificación y en firmas digitales.

Los algoritmos criptográficos de clave simétrica que son soportados en la biblioteca ESP se muestran en la tabla P15-C.1, mientras que las funciones hash que son soportadas se muestran en la tabla P15-C.2.

Tabla P15-C.1 *Algoritmos de clave simétrica soportados.*

Algoritmo	**IV (bytes)**	**Claves (bytes)**	**Número de Claves**	**Bloque e/s (bytes)**
AES ECB	NA	16,24,32	1 para cifrar 1 para descifrar	16
AES CBC	16	16,24,32	1 para cifrar 1 para descifrar	16 y múltiplos de 16
ARC4	NA	1-58	1 para cifrar y descifrar	1-500
BLOWFISH ECB	NA	4-56	1 para cifrar y descifrar	8
BLOWFICH CBC	8	4-56	1 para cifrar y descifrar	8 y múltiplos de 8
XTEA ECB	NA	16	1 para cifrar y descifrar	8
XTEA CBC	8	16	1 para cifrar y descifrar	8 y múltiplos de 8
CAMELLIA ECB	NA	16,24,32	1 para cifrar 1 para descifrar	16
CAMELLIA CBC	16	16,24,32	1 para cifrar 1 para descifrar	16 y múltiplos de 16
AES CFB128	16	16	1 para cifrar y descifrar	16
AES CFB8	16	16,24,32	1 para cifrar y descifrar	16

Tabla P15-C.2 *Funciones hash soportadas.*

Algoritmo	Entrada (bytes)	Salida (bytes)
SHA1	500	20
SHA254	500	28
SHA256	500	32
SHA384	500	48
SHA512	500	64
MD5	500	16
MD4	500	16

El establecimiento del tamaño de la clave para cifrar o descifrar determinará sí el algoritmo será de 16, 24 o 32 bits.

Cabe mencionar que ninguno de los métodos criptográficos de cifrado en bloque cuenta con algún método de relleno (*padding*) por lo que se debe ingresar el bloque completo

Objetivo

Mostrar cómo utilizar los algoritmos criptográficos de clave simétrica con clave de tamaño variable.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S.

Habilitación de aplicaciones

- ACCESS_POINT, E_TCP_SERVER y E_CRYPTOGRAPHIC_FUNC en el módulo que funcione como servidor
- STATION, E_SSL_TCP_UDP_CLIENT_UNICON y E_CRYPTOGRAPHIC_FUNC en el módulo que funcione como cliente.

Materiales

- 2 microcontroladores PIC
- 2 módulos WiFi
- 3 interruptores DPST
- 7 resistores de 10 kΩ
- 3 botones pulsadores n.o.
- 2 condensadores de 0.1 µF

Descripción

Se utilizarán dos módulos WiFi, uno funcionará como servidor TCP y AP, mientras que el otro funcionará como cliente TCP y estación. El servidor es creado en el módulo que funciona como AP para establecer una dirección IP personalizada que tomará el servidor, razón por la cual ya no se requiere la

terminal para conocer la dirección IP a la que se debe conectar el cliente.

Cada vez que se presione un botón en el microcontrolador PIC asociado al cliente TCP se deberá enviar el mensaje “hola mundo” al servidor TCP, pero este mensaje estará cifrado con ARC4 con una clave de 5 bytes. El servidor al recibir el mensaje deberá imprimir el mensaje cifrado en una terminal, posteriormente lo descifrará e imprimirá el contenido del mensaje en la misma terminal.

Código

El código del microcontrolador asociado al módulo que funcionará como cliente TCP se muestra en figura P15-C.5, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña del AP creado por el otro módulo.
- Dirección IP y puerto del servidor credo por el otro módulo.

mientras que el código del microcontrolador asociado al servidor TCP se muestra en la figura P15-C.6, en el que se deben ingresar los siguientes parámetros:

- Nombre que tomará el AP y su contraseña.
- Establecer dirección IP del AP que será la misma que la del servidor y el puerto.

Circuito

El circuito del módulo que funcionará como servidor TCP se muestra en la figura P15-C.7, mientras que el circuito del módulo que funcionará como cliente TCP se muestra en la figura P15-C.8.

Resultados

Se ejecutará primero el código del microcontrolador asociado al módulo que creará el servidor TCP. Al aparecer el mensaje de “*ready*” en la terminal (ver figura P15-C.4), significará que el servidor TCP está listo, momento en el que se deberá ejecutar el código del microcontrolador asociado al módulo que funcionará como cliente TCP.

```
\nModulo esp listo para su uso\r\n
ready\r\n
```

Figura P15-C.4 *Mensaje para ejecutar el código del uC PIC asociado al cliente TCP.*

Cuando en la terminal aparezca el mensaje “*connected*” *se* presionará el botón del microcontrolador asociado al módulo que funciona como cliente TCP para realizar el cifrado y el envío de la información. En la terminal se deberá imprimir tanto el mensaje cifrado como descifrado, tal y como se muestra en la figura P15-C.3.

```
connected\r\n
Cifrado:\r\n
ngbo8M\G]\iwFV\r\n
Descifrado:\r\n
hola mundo      \r\n
```

Figura P15-C.3 *Mensaje de recibido.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/30ZO48M

```
#include <16lf1939.h>
#include "esp.h"

char message[]={"hola mundo     "};
char crypto=[17]={0};
// Llave para los métodos criptográficos
int key[5]={1,1,1,1,1};
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    // Conexión al punto de acceso especificado
    begin_station("ESP","AP_FC","123456789");
    /* Se establece la llave variable para las funciones criptográficas, el
     método seleccionado determina si es utilizada para cirar o descifrar*/
    set_var_key(key, sizeof(key));
    // Conexion al servidor TCP
    create_tcp_pclient("192.168.4.1", 100);
    while(1)
    {
        if(1==input_state(PIN_A2)) // ¿Se ha presionado el botón?
        {
            while(1==input_state(PIN_A2))
            {
              // Espera a que se deje de preionar el botón
            }
            // Se cifra el mensaje  con e método seleccionado
            cryptographic(M_ARC4,sizeof(message), message,crypto);
            // Se envia el mensaje cifrado al servidor
            send_tcp(crypto);
        }
    }
}
```

Figura P15-C.5 *Código de la práctica 15-C, módulo como cliente TCP.*

```
#include <16lf1939.h>
#include "esp.h"

short flag=0; // Bandera de conexión
char server_buffer[20]={0};
char crypto[17]={0};
// Llave para los métodos criptográficos
int key[5]={1,1,1,1,1};
extern _bool new_message_server1;
extern uint16 message_server_length;

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //creación de un punto de acceso
    begin_ap(2,"AP_FC","123456789");
    //Establece la dirección IP del AP
    set_ip_ap("192.168.4.1");
    //Establece el buffer del servidor TCP para recibir mensajes
    set_buffer_server1(server_buffer,sizeof(server_buffer));
    // Crea un servidor TCP, con un único cliente en el puerto 100
    create_server1(100,1,200);
    fprintf(PIC, "ready\r\n");
    /* Se establece la llave para las funciones criptográficas, el
     método seleccionado determina si es utilizada para cifrar o descifrar*/
    set_var_key(key,sizeof(key));
    new_message_server1=0;
    while(1)
    {
        if(1==client_connected(5) && flag==0)
        {
            delay_ms(3000);
            fprintf(PIC,"connected\r\n");
            flag=1;
        }
        if(0==client_connected(5))
            flag=0;
        //¿Se recibió un nuevo mensaje?
        if(1==new_message_server1)
        {
            // Se decifra el mensaje
            cryptographic(M_ARC4,message_server_length, server_buffer,crypto);
            delay_ms(10);
            fprintf(PIC,"Cifrado:\r\n");
            fprintf(PIC,"%s\r\n",server_buffer);
            fprintf(PIC,"Descifrado:\r\n");
            fprintf(PIC,"%s\r\n",crypto);
            new_message_server1=0; // Se limpia la bandera de nuevo mensaje
        }
    }
}
```

Figura P15-C.6 *Código de la práctica 15-C, módulo como servidor TCP.*

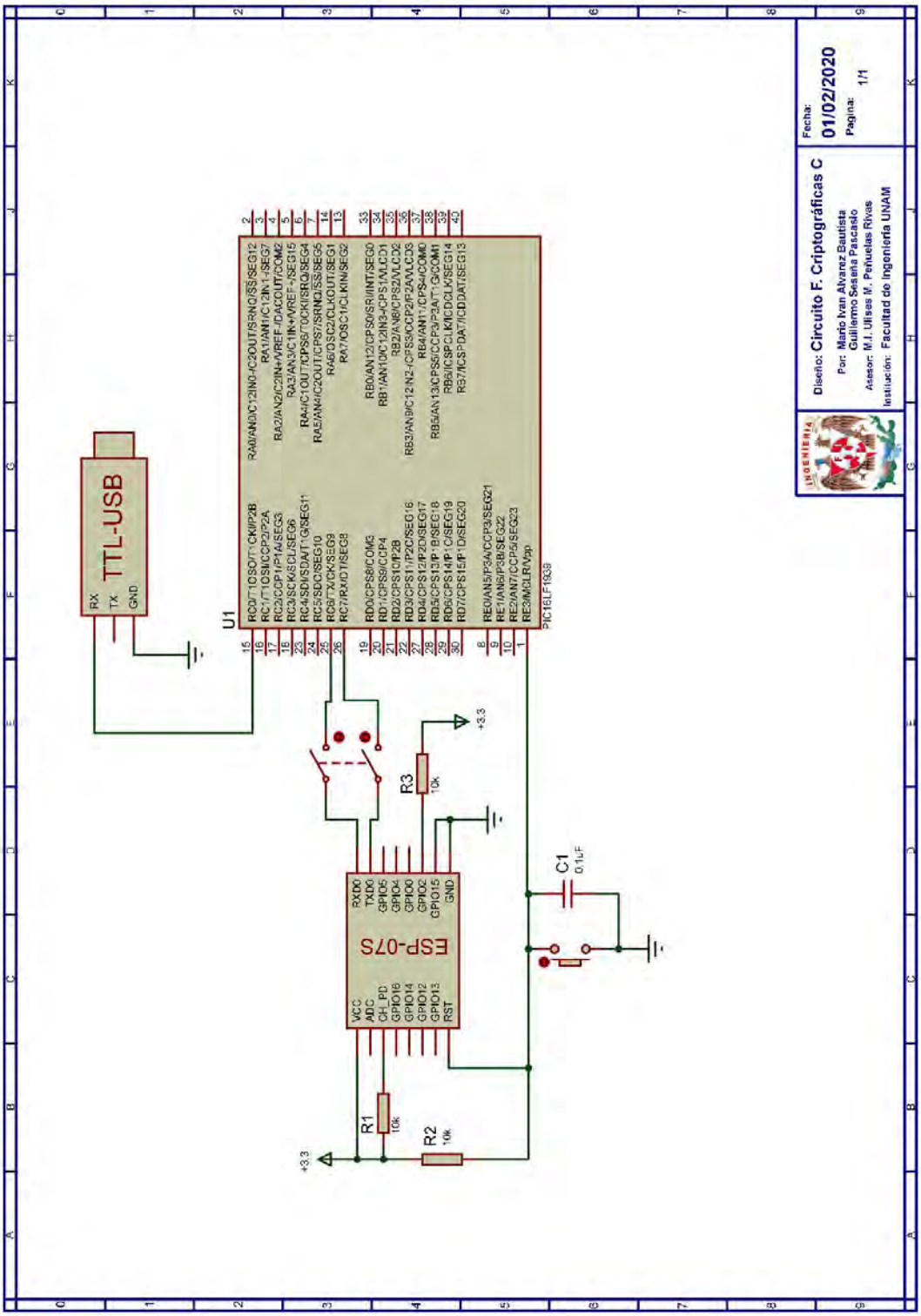


Figura P15-C.7 *Circuito de la práctica 15-C, módulo como servidor TCP.*

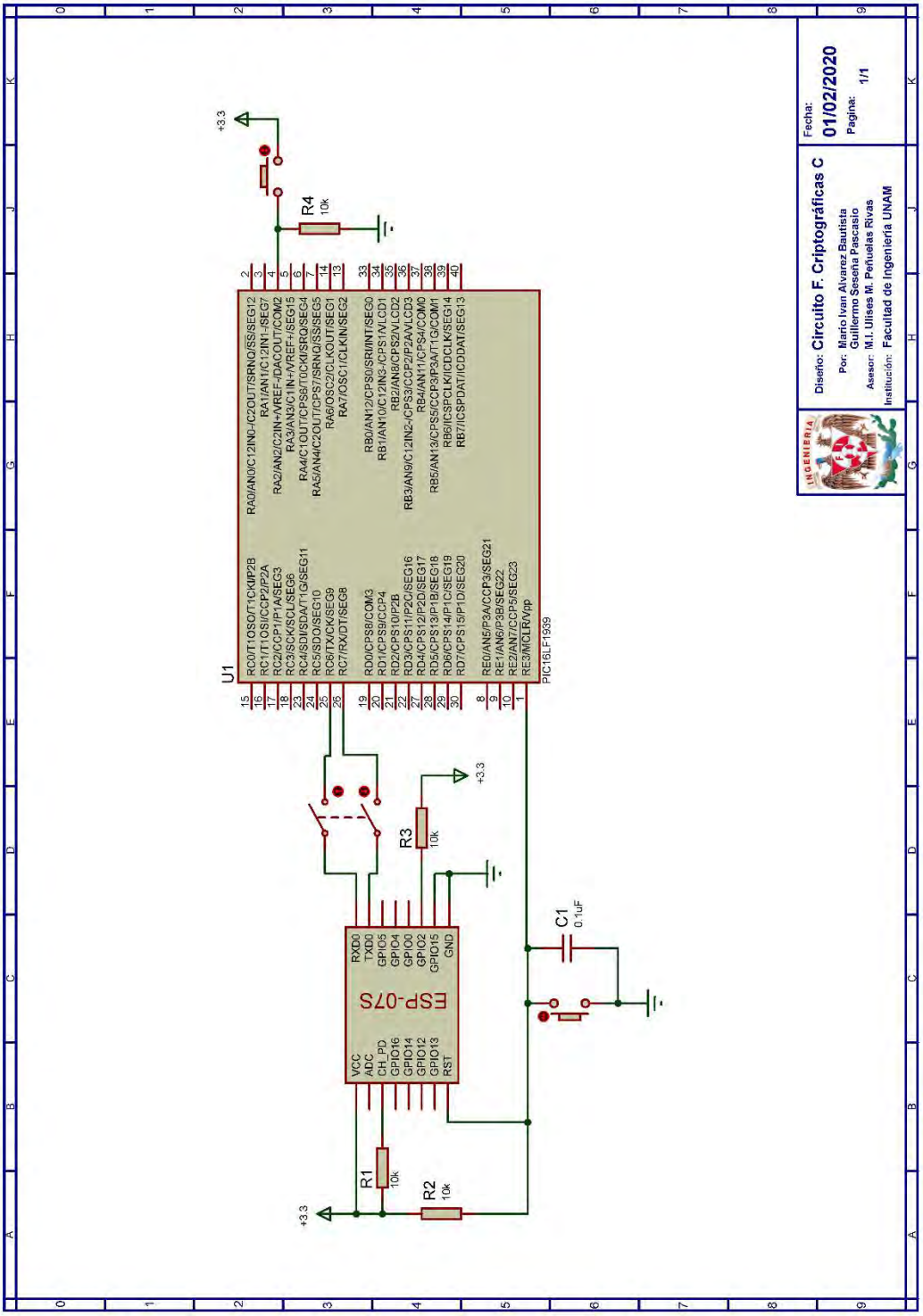


Figura P15-C.8 *Circuito de la práctica 15-C, módulo como cliente TCP.*

Notas

- En el circuito de la práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- Con los métodos ejecutados en el módulo WiFi, se evita que se consuma una gran cantidad de memoria en el uC PIC y se realiza en un tiempo menor.

Referencias

1. Nayak, U. and U.H. Rao, *The InfoSec Handbook. An introduction to informaction Security*. 2014: Apress.
2. Werstén, B., *Implementing the Transport Layer Security Protocol for Embedded Systems.* 2007.

Práctica 16 SNTP

Introducción

Uno de los protocolos de internet más antiguos y famosos es el Protocolo de tiempo de red (*Network Time Protocol,* NTP), diseñado para sincronizar el reloj de los dispositivos conectados a internet. NTP funciona por encima del protocolo de transporte UDP y sigue el modelo de cliente-servidor. Este protocolo cuenta con algoritmos para mitigar el desajuste generado por los retardos que existen entre la petición y la respuesta, ofreciendo una precisión de 1 a 50 *ms* [1].

Para dispositivos con poca capacidad computacional, NTP resulta muy complejo, por ello surgió una versión simplificada llamada Protocolo simple de tiempo de red (*Simple Network Time Protocol,* SNTP), este protocolo no cuenta con los algoritmos que mejoran el cronometraje de NTP, por lo que es muy ligero, pero ofrece una precisión del orden de fracciones de segundos [1].

Los módulos ESP pueden fungir como un cliente SNTP, solicitando la fecha y hora para una de las principales 25 zonas horarias del Tiempo universal coordinado (UTC). Estos datos los obtiene de los servidores:

- cn.ntp.org.cn
- ntp.sjtu.edu.cn
- us.pool.ntp.org.

Objetivo

Mostrar cómo utilizar el cliente SNTP.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_SNTP

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF
- 1 pantalla LCD 16x2

Descripción

En esta práctica se simulará un reloj, en donde el módulo solicitará cada segundo la fecha y hora de la zona horaria a la que pertenece la Ciudad de México (UTC-6),

esta información será mostrada en la pantalla LCD.

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica es el mostrado en la figura P16.1 y la figura P16.2. En el código, se deben ingresar nombre y contraseña de un AP con acceso a internet.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P16.3.

Resultados

Al ejecutarse el código del microcontrolador, en la LCD se mostrarán la fecha y hora actual para la Ciudad de México, esta información es mostrada en idioma inglés y actualizada cada segundo.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2Yf314L

Nota

- Es posible que en ocasiones aparezca Thu, Jan 01, 1970 00:00:00, esto es porque no obtuvo una respuesta exitosa de los servidores SNTP y el módulo indicará el valor inicial del sistema de tiempo POSIX.

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char fecha_hora[26];
char aux_[16];
void main()
{
    //Inicialización de la LCD
    lcd_init();
    //Posiciona escritura en la LCD
    lcd_gotoxy(3,1);
    //Imprime en la LCD
    lcd_putc("Fecha y hora:");
     //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Establece la zona horaria de la Cuidad de México (UTC-6)
    set_sntp(-6);
    while(1)
    {
        //Obtiene la fecha y hora
        get_sntp(fecha_hora, sizeof(fecha_hora));
        /*Ejemplo de como se obtiene: Thu Feb 27 13:20:01 2020
        Para  la LCD que tiene dos filas de 16 caracteres, se
        ordena y se muestra:
        Thu,Feb 27,2020
            13:20:01      */
        aux_[0]=fecha_hora[0];
        aux_[1]=fecha_hora[1];
        aux_[2]=fecha_hora[2];
        aux_[3]=',';
        aux_[4]=fecha_hora[4];
        aux_[5]=fecha_hora[5];
        aux_[6]=fecha_hora[6];
        aux_[7]=fecha_hora[7];
        aux_[8]=fecha_hora[8];
        aux_[9]=fecha_hora[9];
        aux_[10]=',';
        aux_[11]=fecha_hora[20];
        aux_[12]=fecha_hora[21];
        aux_[13]=fecha_hora[22];
        aux_[14]=fecha_hora[23];
        aux_[15]='\0';
        //Posiciona escritura en la LCD
        lcd_gotoxy(1,1);
```

Figura P16.1 *Primera parte del código de la práctica 16.*

```
        //Imprime en la LCD
        printf(lcd_putc,"%s",aux_);
        aux_[0]=fecha_hora[11];
        aux_[1]=fecha_hora[12];
        aux_[2]=fecha_hora[13];
        aux_[3]=fecha_hora[14];
        aux_[4]=fecha_hora[15];
        aux_[5]=fecha_hora[16];
        aux_[6]=fecha_hora[17];
        aux_[7]=fecha_hora[18];
        aux_[8]='\0';
        //Posiciona escritura en la LCD
        lcd_gotoxy(5,2);
        //Imprime en la LCD
        printf(lcd_putc,"%s",aux_);
        delay_ms(1000);
    }
}
```

Figura P16.2 *Segunda parte del código de la práctica 16.*

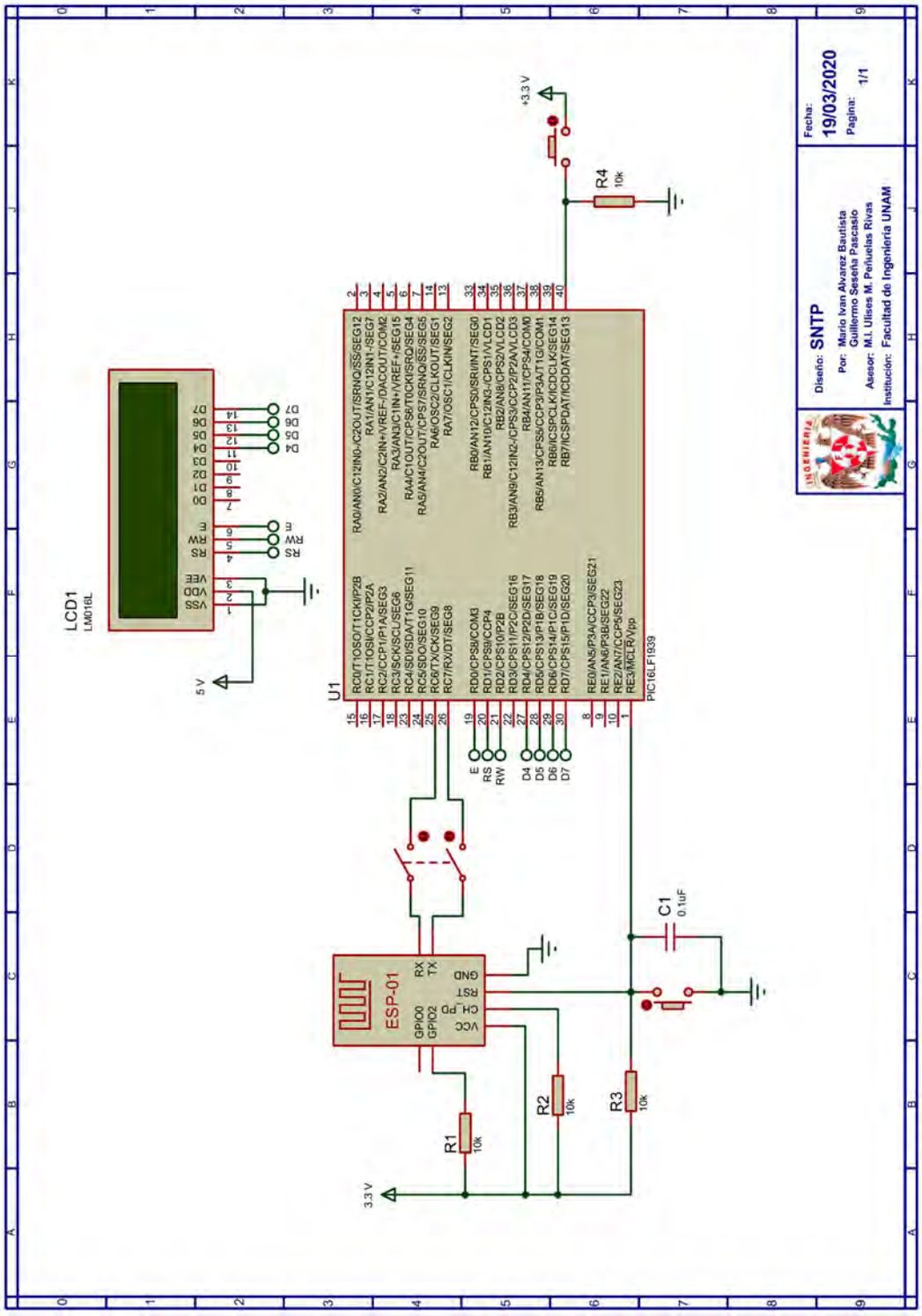


Figura P16.3 *Circuito de la práctica 16.*

Referencias

1. The Internet Society. *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*. 2006 [Citado 2020 Marzo]; Disponible en:https://tools.ietf.org/html/rfc4330.

Práctica 17-A SMTP: texto plano

Introducción

El correo electrónico ha sido una de las aplicaciones más populares de internet desde su surgimiento. En sus inicios estaba pensado únicamente para la transferencia de texto y fue evolucionando hasta que, en la actualidad, es posible enviar toda clase de archivos. El correo electrónico se ha mantenido vigente aun en el surgimiento de la mensajería instantánea y de las llamadas de voz [1].

El proceso "tradicional" de envío de correo electrónico se puede describir en la figura P17-A.1, en la que se señala que el proceso inicia con el agente de usuario del emisor, éste es un programa que se encarga de proporcionar una interfaz para que el usuario pueda interactuar con el sistema de correo electrónico, permitiéndole redactar y leer correos [1].

Una vez que el correo se ha redactado es llevado al agente de transferencia de mensajes, que tiene como encomienda transferir el correo al agente de transferencia de mensajes del destinatario mediante el Protocolo simple de transferencia de correo (*Simple Mail Transfer Protocol,* SMTP). Este protocolo funciona por encima de una conexión TCP o SSL, comúnmente establecida en el puerto 25 para TCP y en el 465 para SSL. El correo se transmite del cliente SMTP al servidor SMTP, mediante la ejecución de comandos de texto ASCII de 7 bits, por cada comando que se ejecute el servidor responde con un código de respuesta que indica el resultado del comando. En la actualidad es más común que los servidores usen SMTP con extensiones (*Extended SMTP*), que otorgan características como autentificación y transmisión de datos binarios [1].

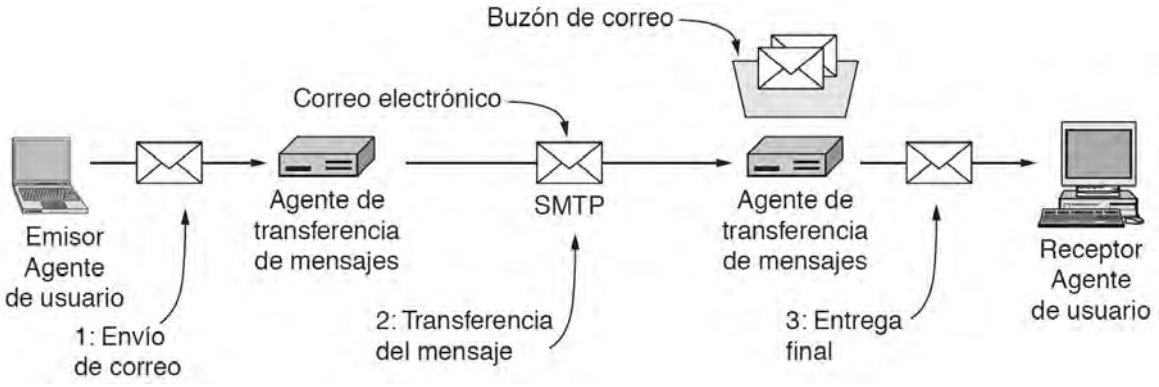


Figura P17-A.1 *Arquitectura del sistema de correo electrónico [1].*

En un principio los agentes de transferencia se ejecutaban dentro de la misma máquina del emisor y del receptor, por lo que el envío de correos solo era posible cuando el agente de transferencia del receptor estuviera disponible para recibirlo. Hoy en día, los agentes de transferencia son comúnmente servidores que siempre se encuentran conectados a internet, por lo que el envío de correos se puede efectuar en todo momento. De esta forma, el protocolo SMTP es utilizado en dos ocasiones: la primera es para llevar el correo del agente de usuario del emisor (cliente SMTP) al agente de transferencia del emisor (servidor SMTP), y la segunda es del agente de transferencia del emisor (cliente SMTP) al agente de transferencia de receptor (servidor SMTP) [1].

Finalmente, cuando el correo es almacenado por al agente de transferencia de mensajes del receptor, el usuario puede acceder a él directamente si el agente de transferencia se encuentra en la misma máquina o en caso contrario debe solicitarlo al servidor mediante protocolos como Protocolo de oficina de correos (*Post Office Protocol,* POP) o Protocolo de acceso a mensajes de internet (*Internet Message Access Protocol,* IMAP).

Un correo electrónico está constituido por una envoltura y un contenido. La envoltura contiene información relevante para transmitir el correo y es usada por el protocolo SMTP, como la dirección del emisor y la del receptor. En cambio, el contenido está dividido en dos partes: el encabezado que aporta información al agente de usuario de como mostrar el correo; y el cuerpo que contiene la información que se muestra al usuario.

El contenido de un correo electrónico comienza con los campos del encabezado, cada campo utiliza una línea de texto ASCII que especifica el nombre del campo seguido de dos puntos, el valor de dicho campo y finalmente de un retorno de carro (CR-*carriage return*) y un salto de línea (LF-*line feed*). Un correo al menos debe especificar tres encabezados: *from:* indica la dirección de quien escribió el correo; *to:* indica la dirección del destinatario; y *subject:* indica el asunto del correo. Posteriormente del campo de encabezados, el contenido del correo dispone de una línea en blanco (<CR> <LF>) para indicar el inicio del cuerpo del correo, que termina hasta encontrar una línea en blanco, un punto y otra línea en blanco (<CR><LF>.<CR><LF>) [1].

En la biblioteca ESP se han desarrollado funciones para que el módulo trabaje como un agente de usuario, que transfiere correos electrónicos a un servidor SMTP. Estas funciones establecen una conexión SSL con un servidor SMTP y ejecutan una secuencia

Tabla P17-A.1 *Secuencia de comandos y respuestas que son identificables por la biblioteca ESP.*

Paso	Comando SMTP	Código de respuesta esperado
1	(Se establece la conexión)	220
2	EHLO ESP	250
3	AUTH LOGIN	334
4	” Dirección email del emisor”	334
5	“Contraseña”	235
6	MAIL FROM:<” Dirección email del emisor”>	250
7	RCPT TO:<” Dirección email del receptor”>	250
8	DATA	354
9	“Contenido del correo electrónico”	250
10	QUIT	221

de comandos mínimos para enviar un correo electrónico, esta secuencia se muestra en la tabla P17-A.1, si el servidor no tiene la capacidad de ejecutar estos comandos fracasará el envío.

El usuario de la biblioteca puede optar por dos opciones para enviar un correo: una es utilizar la función *send_mail_smtp*, que internamente construye todo el contenido del correo solicitado, pero solo puede enviar texto plano; la otra opción es construir por completo el contenido del correo que se desee enviar haciendo uso de varias funciones.

Objetivo

Enviar un correo electrónico con contenido de texto plano a través del protocolo SMTP.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_SMTP

Materiales

- 1 computadora.
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 4 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 1 condensador de 0.1 µF

Descripción

En esta práctica cada vez que se oprima un botón, se enviará un correo electrónico con contenido de texto plano, para lo cual es necesario contar con una cuenta en algún servicio de correo electrónico que disponga de un servidor STMP. Con fines demostrativos esta práctica se realizará a través de Yahoo! Mail, cuyo servidor SMTP es smtp.mail.yahoo.com con puerto 465 para conexiones TLS.

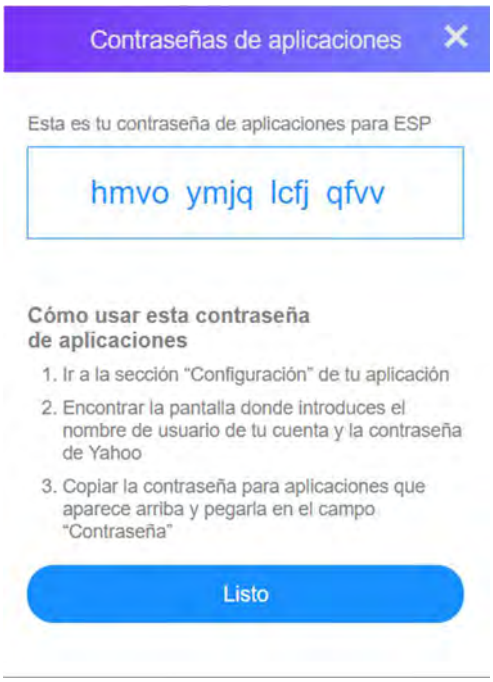


Figura P17-A.2 *Ejemplo de contraseña de aplicación para Yahoo! Mail.*

Para usar una cuenta de correo Yahoo! será necesario crear una contraseña de aplicación, para ello se ingresa a la página: https://login.yahoo.com/account/

En dicha página se accede al apartado de seguridad (ver figura P17-A.3) y se selecciona la opción de "Generar contraseña de aplicaciones", se selecciona la opción de "Otras aplicaciones", se especifica un nombre a la aplicación y se obtendrá una contraseña (ver figura P17-A.2) que debe ser colocada en el código del microcontrolador.

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica se muestra en la figura P17-A.4. En este código, se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet
- Correo emisor, dirección del correo electrónico de la cuenta a utilizar
- Contraseña de aplicación
- Correo receptor, dirección del correo electrónico de quien recibe el correo

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P17-A.5.

Resultados

Cada vez que se oprima el botón se enviará el correo electrónico, el cual debe aparecer en la bandeja de enviados de la cuenta utilizada y debe aparecer en el buzón de la dirección a la cual fue enviado el correo. En la figura P17-A.6 se muestra un ejemplo en que se utiliza una cuenta de Yahoo! Mail tanto de emisor como de receptor, aunque la dirección del receptor puede ser cualquier otro proveedor de servicio de correo electrónico.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3etlgJm

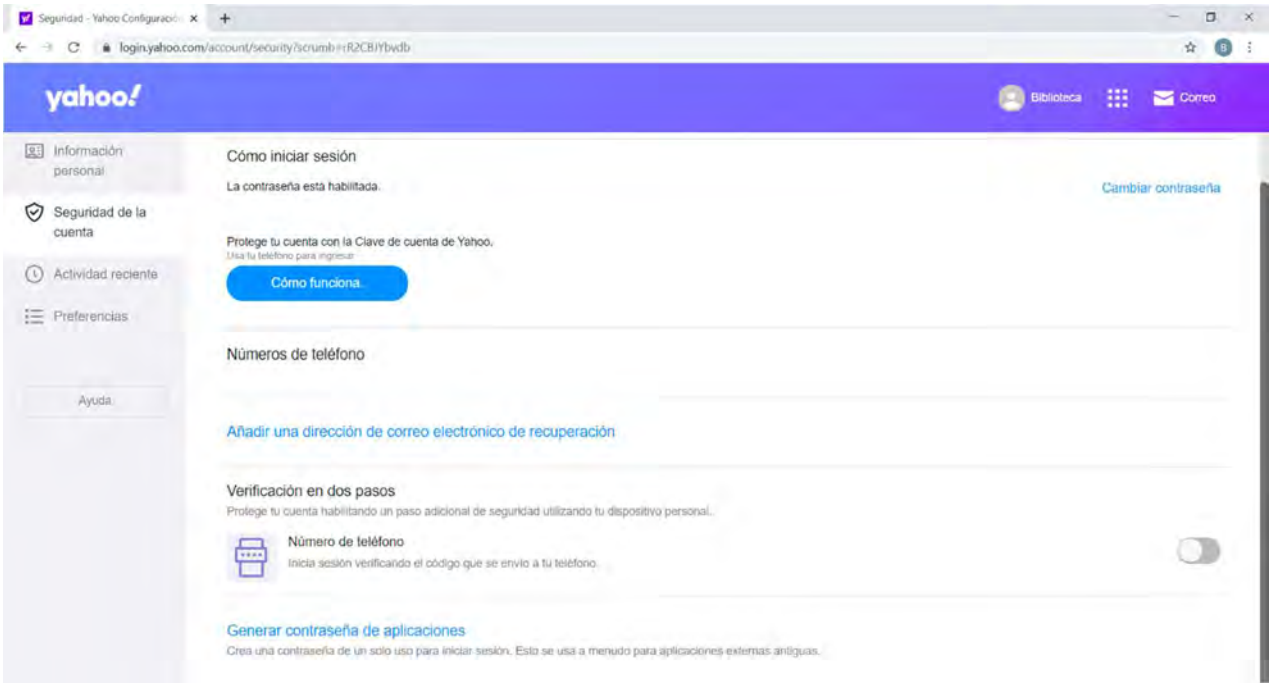


Figura P17-A.3 *Página para la generación de contraseñas de aplicación para Yahoo! Mail.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
/*
correo_emisor-> Dirección de correo electrónico de la cuenta a utilizar
contrasena->Contraseña de aplicación generada para la cuenta de correo a utilizar
correo_receptor->Dirección de correo electrónico a la cual se le envía el correo
*/
#define correo_emisor "correo_emisor@yahoo.com"
#define contrasena "contraseña_de_aplicación"
#define correo_receptor "correo_receptor@yahoo.com"

#define servidor_smtp "smtp.mail.yahoo.com"
#define puerto_smtp 465
#define asunto "Boton presionado."
#define contenido "El boton fue presionado."

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    while(1)
    {
      //Al presionar botón envia correo electrónico
      if(1==input_state(PIN_B7))
      {
        /*Envio de correo a traves del protocolo SNTP con contenido de texto plano.
        (Comunicación en el canal 0)*/
        send_mail_smtp(servidor_smtp,puerto_smtp,0,correo_emisor,contrasena,
        correo_receptor,asunto,contenido);
        while(1==input_state(PIN_B7))
        {
            //Se mantiene hasta soltar el botón
        }
      }
    }
}
```

Figura P17-A.4 *Código de la práctica 17-A.*

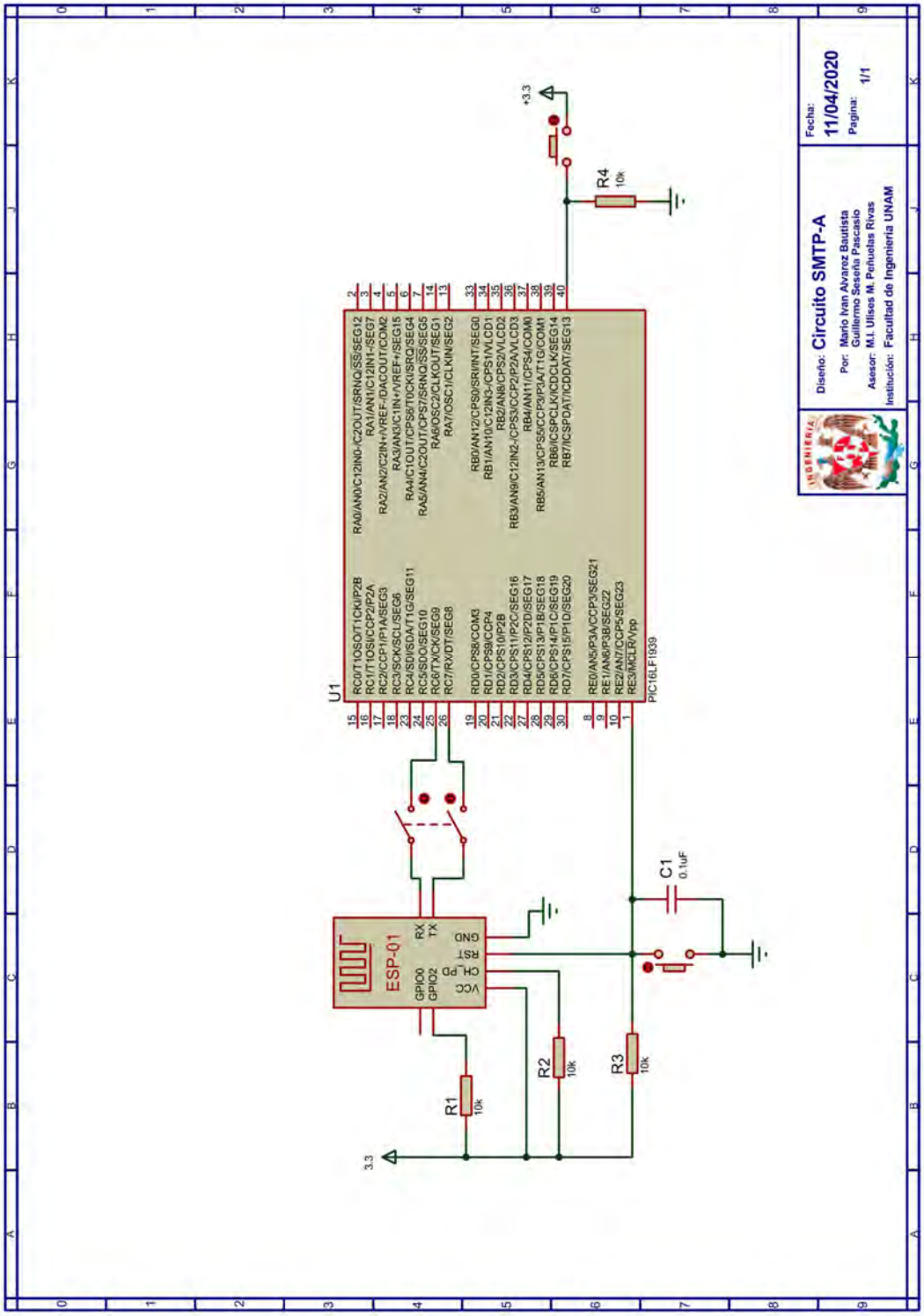


Figura P17-A.5 *Circuito de la práctica 17-A.*

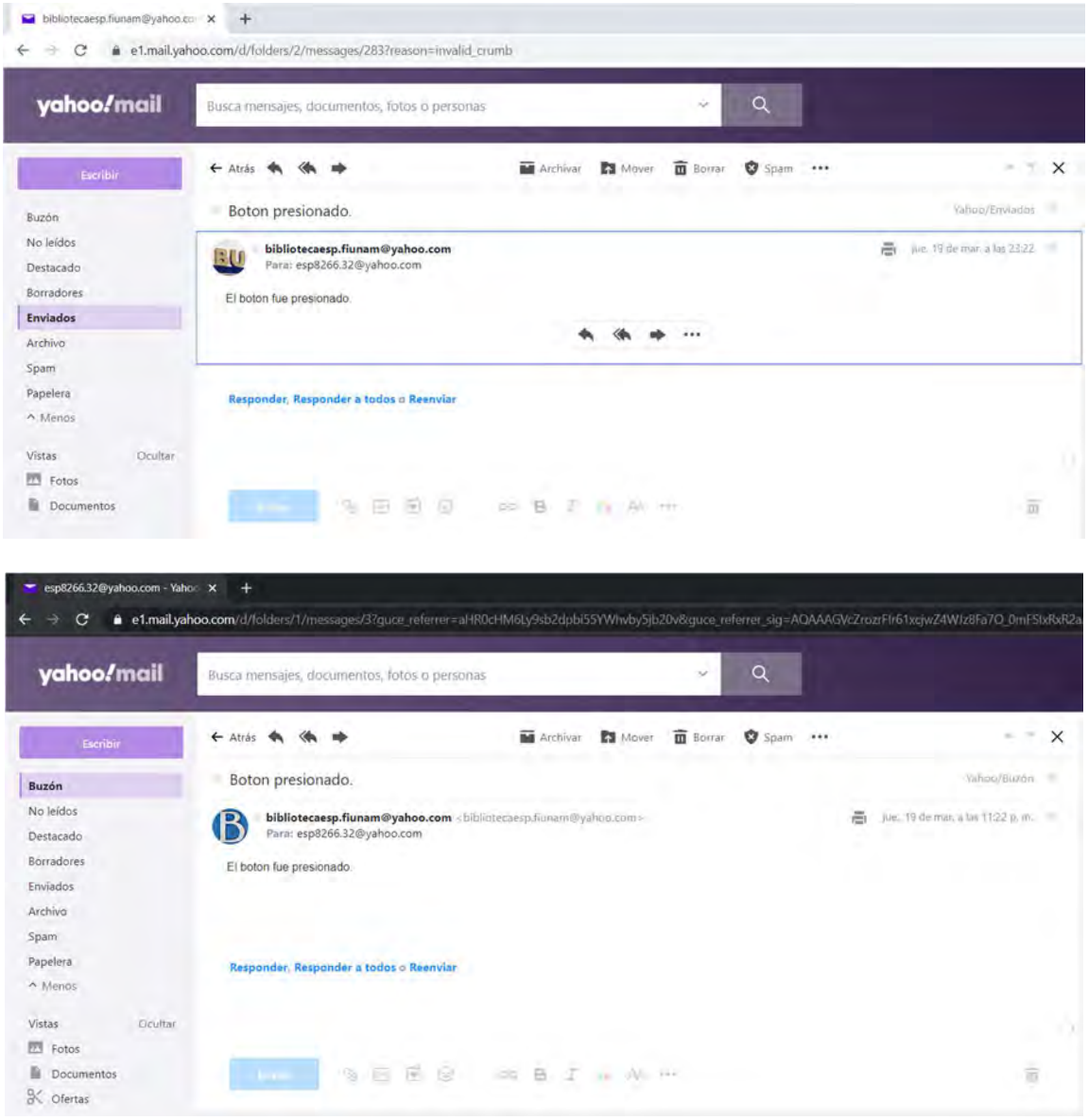


Figura P17-A.6 *Correo electrónico enviado por el módulo mostrado en la carpeta de enviados de la cuenta utilizada y mostrado también en el buzón del destinatario.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 17-B SMTP: multiformato

Introducción

En un principio al implementar un protocolo ASCII como transmisor (SMTP), los correos electrónicos únicamente contenían texto en ingles sin ningún formato, esta limitante fue solucionada mediante el desarrollo de Extensiones Multipropósito de Correo Internet (*Multipurpose Internet Mail Extensions,* MIME) que aporta encabezados para describir el contenido de correos electrónicos y de otras aplicaciones de internet como la navegación web [1].

El encabezado que permite utilizar MIME es *MIME-Version: 1.0*., y el encabezado Content-*Transfer-Encoding:* indica cómo se debe codificar el contenido, permitiendo cinco distintas formas:

- *7bits* es la codificación ASCII por defecto en la que utiliza caracteres de 7 bits
- *8 bits* es aquella que utiliza caracteres con valor binario de 0 a 255
- *binary* en la que los datos son completamente binarios, transmitidos en 8 bits
- *base64* en la que se codifica datos binarios a ASCII de 7 bits
- quoted-printable es la codificación entrecomillada imprimible, que es semejante a la codificación ASCII de 7 bits, pero incluye valores binarios que son representados en valor hexadecimal usando caracteres ASCII [1]

Tabla P17-B.1 *Tipos MIME.*

Tipo	Subtipos comunes
text	plain, html, xml, ccs, javascript
image	jpeg, png, gif, bmp, webp
audio	acc, midi, webm, ogg, x-wav
video	mpeg, mp4, webm, ogg
model	Vrml
application	octet-stream, pdf, javascript, zip, x-rar-compressed
multipart	mixed, alternative, parallel, digest

Para definir el tipo de contenido de un correo electrónico se usa el encabezado llamado *Content-Type,* en él se define un tipo y subtipo, los cuales están separados por una "/". En la tabla P17-B.1 se muestran algunos de los principales tipos y subtipos.

La biblioteca ESP permite enviar correos electrónicos a través del protocolo SMTP. El usuario tiene la posibilidad de construir por completo el contenido del correo, permitiendo incluir los encabezados que requiera, de esta forma, el correo que envíe el módulo puede contener más que simple texto e incluso puede adjuntar archivos. La biblioteca provee de tres funciones para personalizar el contenido del correo:

- `begin_smtp`
- `send_content_smtp`

- `finish_part_smtp`.

La primera de estas funciones sirve para ejecutar una secuencia de comandos SMTP, dejando abierta la conexión con el servidor para que se envíen fragmentos del contenido del correo con la segunda función, después de enviar todo el contenido se debe ejecutar la última de las funciones, la cual se encarga de enviar el finalizador del contenido (<CR><LF>.<CR><LF>) e indica al servidor que ha concluido y que puede cerrar la conexión.

Objetivo

Enviar un correo electrónico con contenido multi-formato a través del protocolo SMTP.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_SMTP

Materiales

- 1 computadora.
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 4 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 1 condensador de 0.1 µF

Descripción

En esta práctica, cada vez que se oprima un botón se enviará un correo electrónico con contenido multi-formato, por lo que será necesario contar con una cuenta en algún servicio de correo electrónico que disponga de un servidor SMTP.

Con fines demostrativos esta práctica se realizará a través de Gmail, cuyo servidor SMTP es smtp.gmail.com con puerto 465 para conexiones TLS. Para usar una cuenta de correo Gmail será necesario el Acceso de apps menos seguras, por lo que se debe ingresar a la página https://myaccount.google.com/lesssecureapps y permitir dicho acceso (ver figura P17-B.1).

En esta práctica se enviarán cuatro tipos de contenido en un solo correo electrónico, los cuales son:

- Texto plano (text/plain)
- Texto HTML (text/html)
- Una imagen png (image/png)

- Un archivo comprimido (application/x-rar-compressed)

El contenido del correo que se enviará en esta práctica se muestra en la figura P17-B.2. Como se observa en ella, lo primero que se coloca son cinco encabezados, tres de ellos son obligatorios para cualquier correo:

From: especifica la dirección de correo electrónico del emisor.

To: especifica la dirección de correo electrónico del receptor.

Subject: especifica el asunto del correo electrónico.

Los sobrantes encabezados son relacionados con MIME: *MIME-Version: 1.0* indica la versión MIME que se utiliza; y *Content-Type: multipart/mixed;* señala que el contenido del correo está compuesto de distintos tipos, también en este encabezado se indica *boundary= orue8rpq5itfbrt92skw*, este parámetro establece una cadena de menos de setenta caracteres ASCII que sirve para delimitar las múltiples partes del correo, este delimitador no debe aparecer en otra parte del contenido más que en los propios límites, ya que puede ser confundido e interpretado erróneamente, para esta práctica el *boundary* (frontera) fue elegido aleatoriamente.

Posteriormente de los encabezados se dejan tres líneas en blanco y se delimitan cuatro secciones usando el *boundary* antecedido por "--".

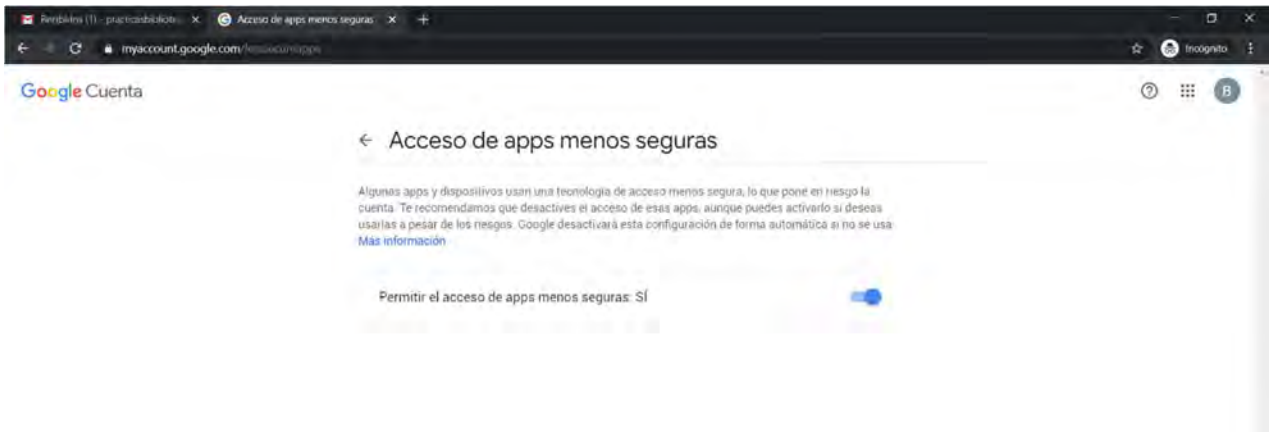


Figura P17-B.1 *Activación al acceso de apps menos seguras.*

From: biblioteca.esp.fi.unam@gmail.com
To: esp.3266.32@gmail.com
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=orue8rpq5itfbrt92skw
Subject: Correo electronico con contenido multi-formato

--orue8rpq5itfbrt92skw
Content-Type: text/plain; charset=utf-8

Contenido texto plano.

--orue8rpq5itfbrt92skw
Content-Type: text/html

<!DOCTYPE html><html><head><meta charset="utf-8"></head><body style="background-color:#811832;
color:white; text-align: center;"><h1>Contenido HTML</h1></body></html>

--orue8rpq5itfbrt92skw
Content-Type: image/png; name="Logo.png"
Content-Transfer-Encoding: base64

iVBORw0KGgoAAAANSUhEUgAAAIAAAACACAMAAAD04JH5AAAABGdBTUEAALGPC/xhBQAAAAFzUkdC
AK7OHOkAAAAzUExURUdwTA4LBF1UQjs0JCMdEF1UQVlOOIZ7ZhoVDQYEAU9FMDQtHktDMiwlGUM6KlV
OPgAAANNQyuIAAAAQdFJOUwD6Q8DoLhkI9f1g0l/fqXzNaAhSAAAEIUlEQVR42u1a627rIAwuBBLMLbz/056
qW84WMGAnDdWkfJP2i8Yf+laNH48bN278ScxaWhsN/EewVup5GSNcrt4pk3KACtFbfa3wZZbelbJ/INSTw3zZ
1v0UWtK/YJSLFxzEbNcgEhXCPDm8zSIWvUYXDCQG4KmMp0nIN5yE9EzZe3VEe+Ag5tU/t/yCUuKw9M0s1
denwuS9pEiPXzIhvf6fFP+jktd/EFPHNHRU6VoYJxsU1qvFv84hVgPFCmkEwM0flV9lYEUahgmLN2Gc/ARrKd+
nkTBFSJBmKlHkcwJjDyClkNnh4gYTgEwHerAGUoqf1cAzS+0JhOEE0p4ADJcPN4GbwE3g03EglzB9OhCtw+
W7rBwYroP8RhlHyzdZ2bhIMZZAzMuTZawZlnfCh1YjfXDF6oKBwchh9eE4JYDB6/QZuZiCKNH6dLkaSAbwza
C8mUZdYmqotlyNVHy+Wp773t21c4EFZGulf4d6H29RvXjRyRu2XF1EOLCNDkkk1LGtqF1+WxY2YFpNmnI5a
AaBcnNIxeWaTSLTX886AQ0ME8T1C57eSChOAPFsaPfqYt9rWzF77duraffpsDZB1tGx9NaLLxUAU6cxjJ3vng
GdgAfCReRB6RMEeYDAHCHxCVTCnLFsAhpveZjec0bl82KSlFaC2j6vPW6qEHrd82q3UHjCDX4LW7LqqlOP
QN3JVC1fIlEgkh2VEem/CWhB6L5FTrrK3CCeIQA9Aqb/ZuHfcQK+a6V8IwBXS5hInKtVehD7T0jVlqWnVJHfxz
RXdiEIr0Y1I9gM3FMIPPBdgKK841XeLcTMIFA7Rkd5xLPmOgKRQqBihVsM7RBYWpYCK0F+rULaTs+RmuD
2aBSoG8FGoF3DbqaKE3C0l/3ZnScgUTOyDxpiK43RCKDvD4r6mI7pYLOfxZAKfyyn9a6DHT+QtC6Ca4RCTx
5ZcPU8SiUQjsXhug6EZhHAArqjzzIg/L/z6EIkgEQisg/gfrAV9T0CoXqtUJxhjlIMlUD1BMCxJmdMTYFznJrYzlke9
wE04tPSSOslVPBGenI/YLhQxZUDc5QnHMpj9cocLPP3sd+s4n2A+3vLrKh6SnTcD2TX/+nB3gF0G44clzhNgG
sC+cWMT2CvAsWfq1ubrTJuIHDzuQ+wnSjvNvE3sPejAwR2wdQcmWzc6cBrLX/9zfiw7+81Up1xwqJZA2YHh+
zIht0ScS6MtGfLIAVd1LSQLTlPoJn5l2v1BQQcaxLrEIH2dBmPwBEvrFRoh1TAvk30JxwLI8Q701tJdHDK2iqDQ
yFzqcs61ZbH4+PuSwXvWH3jxo2/imXu4Vr50qke4oUUFixDQj7FMV0X+mijPgdz3/k7Cv1I7GoC/jICtJk7uE4FtC
OIF+bfxYsEHUx3/r9x4wYL/wDlE9Pxzb/37QAAAABJRU5ErkJggg==

--orue8rpq5itfbrt92skw
Content-Type:application/x-rar-compressed; name="Notas.rar"
Content-Transfer-Encoding: base64

UmFyIRoHAQAzkrXICgEFBgAFAQGAgADbXKY3JQIDC64ABK4AIMc0mhyAAAAJTm90YXMudHh0CgMCiYA
CP+/m1QFVbiBibG9jIGRlIG5vdGFzIGNvbXByaW1pZG8gZW4gdW4gYXJjaGl2byByYXIuHXdWUQMFBAA=

--orue8rpq5itfbrt92skw—

Figura P17-B.2 *Contenido del correo electrónico de la práctica 17-B.*

En cada sección se coloca uno o varios encabezados seguidos por una línea en blanco, el contenido de esa parte y finalmente otra línea en blanco.

En la primera sección se coloca el encabezado *Content-Type: text/plain; charset=utf-8* que establece que es texto plano que se codifica en el estándar UTF-8. El contenido de esta parte es "Contenido texto plano".

Para la segunda sección se coloca el encabezado *Content-Type: text/html* para establecer que está parte será un documento HTML, en el que se muestra un encabezado de nivel uno (h1), que menciona "Contenido HTML.", con letras blancas y fondo rojo.

En la tercera sección se coloca dos encabezados: *Content-Type: image/png; name="Logo.png"*, establece que corresponde a un archivo *png* cuyo nombre es Logo, este archivo corresponde a una imagen del logo deportivo de la UNAM; y *Content-Transfer-Encoding: base64* indica que la imagen está codificada en base64, método que codifica de binario a texto (caracteres ASCII). Base64 toma una secuencia de bytes y los divide en grupos de 24 bits (3 bytes) obteniendo de cada grupo 4 caracteres ASCII, es por ello que al codificar un archivo y colocarlo en el código del microcontrolador el tamaño de dicho archivo incrementará en un 33%. El archivo que se utiliza en esta parte tiene un tamaño de 1,234 bytes que al codificar produce 1648 caracteres, es decir que utiliza 1648 bytes de memoria del microcontrolador, que en este caso es proporcionada por la memoria ROM. Para codificar archivos se pueden utilizar páginas de internet que ofrecen ese servicio como: https://base64.guru/converter/encode/file la cual permite codificar gratuitamente archivos, sólo basta con seleccionar el archivo y se obtendrá su respectiva cadena de caracteres ASCII. Esta página también permite decodificar, si decodificamos el resultado de la codificación de la imagen Logo.png, se obtiene lo que se muestra en la figura P17-B.3, en que da detalles del archivo, esto puede ser útil si se quiere adjuntar un archivo y no se sabe cómo definir su tipo MIME.

En la cuarta sección cuenta con dos encabezados:

Content-Type:application/x-rar-compressed; name="Notas.rar"

Los cuales especifican que pertenece a un archivo comprimido tipo rar con nombre Notas, archivo que en su interior cuenta con un archivo tipo .txt, llamado Notas que en su contenido dice "Un bloc de notas comprimido en un archivo rar.". *Content-Transfer-Encoding: base64* indica que está codificado en dicho método, el archivo comprimido original

Figura P17-B.3 *Decodificación del archivo Logo.png a través de la página de Base64 Guru.*

tiene un tamaño de 119 bytes y codificado tiene un tamaño de 160 bytes.

Finalmente, en el contenido se encuentra el *boundary* antecedido y sucedido por "--".

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica se muestra en la figura P17-B.5 y en la figura P17-B.6. En este código, se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet.
- Correo emisor, dirección del correo electrónico de la cuenta a utilizar.
- Contraseña de la cuenta de correo.
- Correo receptor, dirección del correo electrónico de quien recibe el correo.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P17-B.7.

Resultados

Cada vez que se oprima el botón se enviará el correo electrónico, el cual debe aparecer en la bandeja de enviados de la cuenta utilizada y en la bandeja de

recibidos de la dirección a la cual fue enviado el correo. En la figura P17-B.4, se muestra que el correo enviado está dividido en cuatro partes, las primeras dos corresponde a texto en distinto formato y las otras dos son archivos adjuntos. En este ejemplo se utiliza una cuenta de Gmail tanto de emisor como de receptor, aunque la dirección del receptor puede ser de cualquier otro proveedor de servicio de correo electrónico.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3fF4xTH

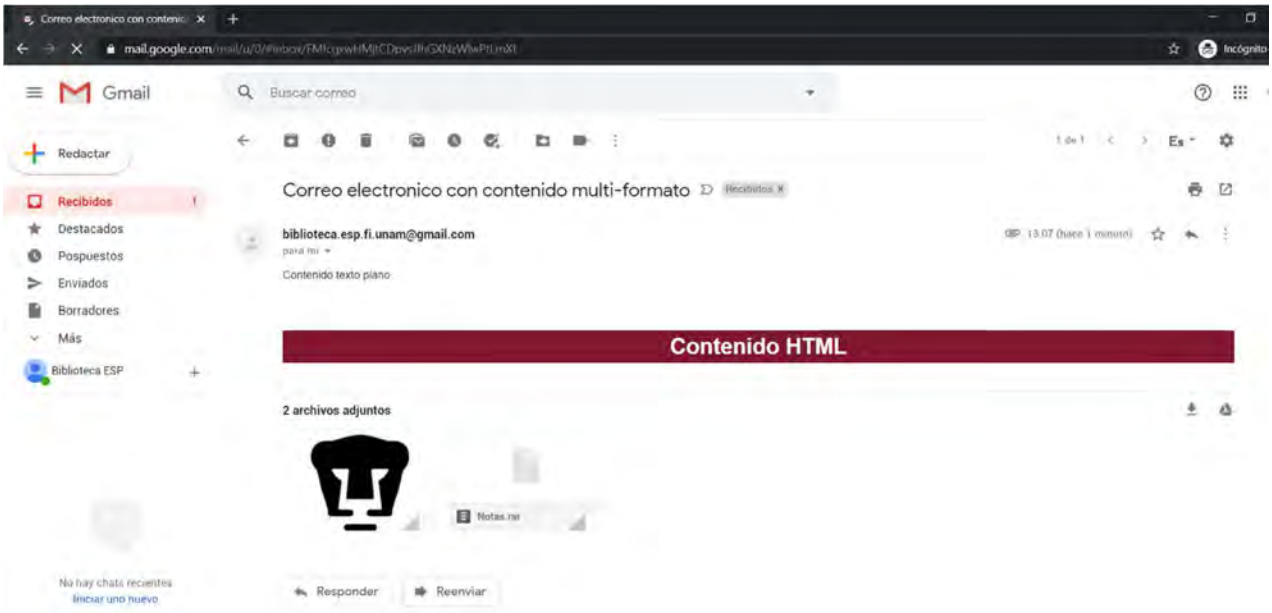


Figura P17-B.4 *Correo electrónico con contenido multi-fomato enviado por el módulo, mostrado en la bandeja de recibidos del destinatario.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
/*
correo_emisor-> Dirección de correo electrónico de la cuenta a utilizar
contrasena->Contraseña de la cuenta de correo a utilizar
correo_receptor->Dirección de correo electrónico a la cual se le envía el correo
*/
#define correo_emisor "correo_emisor@gmail.com"
#define contrasena "contraseña"
#define correo_receptor "correo_receptor@gmail.com"

#define servidor_smtp "smtp.gmail.com"
#define puerto_smtp 465

//Documento HTML
const char html[]=
"<!DOCTYPE html><html><head><meta charset=\"utf-8\"></head><body style=\"backgrou
nd-color:#811832; color:white; text-align: center;\"><h1>Contenido HTML</h1></bod
y></html>"
;
//Primer parte de archivo png codificado en base64
const char png_1[]=
"iVBORw0KGgoAAAANSUhEUgAAAIAAAACACAMAAAD04JH5AAAABGdBTUEAALGPC/xhBQAAAAFzUkdCAK7O
HOkAAAAzUExURUdwTA4LBF1UQjs0JCMdEF1UQVlOOIZ7ZhoVDQYEAU9FMDQtHktDMiwlGUM6KlVOPgAAA
NNQyuIAAAAQdFJOUwD6Q8DoLhkI9f1g0I/fqXzNaAhSAAAEIUlEQVR42u1a627rIAwuBBLMLbz/056qW8
4WMGAnDdWkfJP2i8Yf+IaNH48bN278ScxaWhsN/EewVup5GSNcrt4pk3KACtFbfa3wZZbelbJ/INSTw3z
Z1v0UWtK/YJSLFxzEbNcgEhXCPDm8zSIWvUYXDCQG4KmMp0nIN5yE9EzZe3VEe+Ag5tU/t/yCUuKw9M0s
1denwuS9pEiPXzIhvf6fFP+jktd/EFPHNHRU6VoYJxsU1qvFv84hVgPFCmkEwM0flV9lYEUahgmLN2Gc/
ARrKd+nkTBFSJBmKIHkcwJjDyClkNnh4gYTgEwHerAGUoqf1cAzS+0JhOEE0p4ADJcPN4GbwE3g03EgIz
B9OhCtw+W7rBwYroP8RhIHyzdZ2bhIMZZAzMuTZawZlnfCh1YjfXDF6oKBwchh9eE4JYDB6/QZuZiCKNH
6dLkaSAbwzaC8mUZdYmqotlyNVHy+Wp773t21c4EFZGulf4d6H29RvXjRyRu2XF1EOLCNDkkk1LGtqF1+
WxY2YFpNmnI5aAaBcnNIxeWaTSLTX886AQ0ME8T1C57eSChOAPFsaPfqYt9rWzF77duraffpsDZB1tGx9
NaLLxUAU6cxjJ3vngGdgAfCReRB6RMEeYDAHCHxCVTCnLFsAhpveZje"
;
//Segunda parte de archivo png codificado en base64
const char png_2[]=
"c0bl82KSlFaC2j6vPW6qEHrd82q3UHjCDX4LW7LqqlOPQN3JVC1fIIEgkh2VEem/CWhB6L5FTrrK3CCe
IQA9Aqb/ZuHfcQK+a6V8IwBXS5hInKtVehD7T0jVlqWnVJHfxzRXdiEIr0Y1I9gM3FMIPPBdgKK841XeL
cTMIFA7Rkd5xLPmOgKRQqBihVsM7RBYWpYCK0F+rULaTs+RmuD2aBSoG8FGoF3DbqaKE3C0l/3ZnScgUT
OyDxpiK43RCKDvD4r6mI7pYLOfxZAKfyyn9a6DHT+QtC6Ca4RCTx5ZcPU8SiUQjsXhug6EZhHAArqjzzI
g/L/z6EIkgEQisg/gfrAV9T0CoXqtUJxhjlIMlUD1BMCxJmdMTYFznJrYzlke9wE04tPSSOslVPBGenI/
YLhQxZUDc5QnHMpj9cocLPP3sd+s4n2A+3vLrKh6SnTcD2TX/+nB3gF0G44cIzhNgGsC+cWMT2CvAsWfq
1ubrTJuIHDzuQ+wnSjvNvE3sPejAwR2wdQcmWzc6cBrLX/9zfiw7+81Up1xwqJZA2YHh+zIht0ScS6MtG
fLIAVd1LSQLTlPoJn5I2v1BQQcaxLrEIH2dBmPwBEvrFRoh1TAvk30JxwLI8Q701tJdHDK2iqDQyFzqcs
61ZbH4+PuSwXvWH3jxo2/imXu4Vr50qke4oUUFixDQj7FMV0X+mijPgdz3/k7Cv1l7GoC/jICtJk7uE4F
tCOIF+bfxYsEHUx3/r9x4wYL/wDlE9Pxzb/37QAAAABJRU5ErkJggg=="
;
//Archivo rar codificado en base64
const char rar[]=
"UmFyIRoHAQAzkrXlCgEFBgAFAQGAgADbXKY3JQIDC64ABK4AIMc0mhyAAAAJTm90YXMudHh0CgMCiYAC
P+/m1QFVbiBibG9jIGRlIG5vdGFzIGNvbXByaW1pZG8gZW4gdW4gYXJjaGl2byByYXIuHXdWUQMFBAA="
;

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
```

Figura P17-B.5 *Primera parte del código de la práctica 17-B.*

```
    while (1)
    {
      //Al presionar botón envia correo electrónico
      if(1==input_state(PIN_B7))
      {
        /*Inicia proceso de envío de correo electrónico a través de SMTP
        (Comunicación en el canal 0)*/
        begin_smtp(servidor_smtp,puerto_smtp,0,correo_emisor,contrasena,
        correo_receptor);
        //Envío del contenido del correo electrónico
        send_content_smtp("From: ");
        send_content_smtp(correo_emisor);
        send_content_smtp("\r\nTo: ");
        send_content_smtp(correo_receptor);
        send_content_smtp("\r\nMIME-Version: 1.0\r\n");
        send_content_smtp("Content-Type: multipart/mixed; ");
        send_content_smtp("boundary=orue8rpq5itfbrt92skw\r\n");
        send_content_smtp("Subject: ");
        send_content_smtp("Correo electronico con contenido multi-formato\r\n\r\n");
        send_content_smtp("\r\n\r\n--orue8rpq5itfbrt92skw\r\n");
        send_content_smtp("Content-Type: text/plain; charset=utf-8\r\n\r\n");
        send_content_smtp("Contenido texto plano.");
        send_content_smtp("\r\n\r\n--orue8rpq5itfbrt92skw\r\n");
        send_content_smtp("Content-Type: text/html\r\n\r\n");
        send_content_smtp_const(html);
        send_content_smtp("\r\n\r\n--orue8rpq5itfbrt92skw\r\n");
        send_content_smtp("Content-Type: image/png; name=\"Logo.png\"\r\n");
        send_content_smtp("Content-Transfer-Encoding: base64\r\n\r\n");
        send_content_smtp_const(png_1);
        send_content_smtp_const(png_2);
        send_content_smtp("\r\n\r\n--orue8rpq5itfbrt92skw\r\n");
        send_content_smtp("Content-Type: application/x-rar-compressed; ");
        send_content_smtp("name=\"Notas.rar\"\r\n");
        send_content_smtp("Content-Transfer-Encoding: base64\r\n\r\n");
        send_content_smtp_const(rar);
        send_content_smtp("\r\n\r\n--orue8rpq5itfbrt92skw--\r\n");
        //Finaliza proceso de envío de correo electrónico
        finish_smtp();
        while(1==input_state(PIN_B7))
        {
            //Se mantiene hasta soltar el botón
        }
      }
    }
}
```

Figura P17-B.6 *Segunda parte del código de la práctica 17-B.*

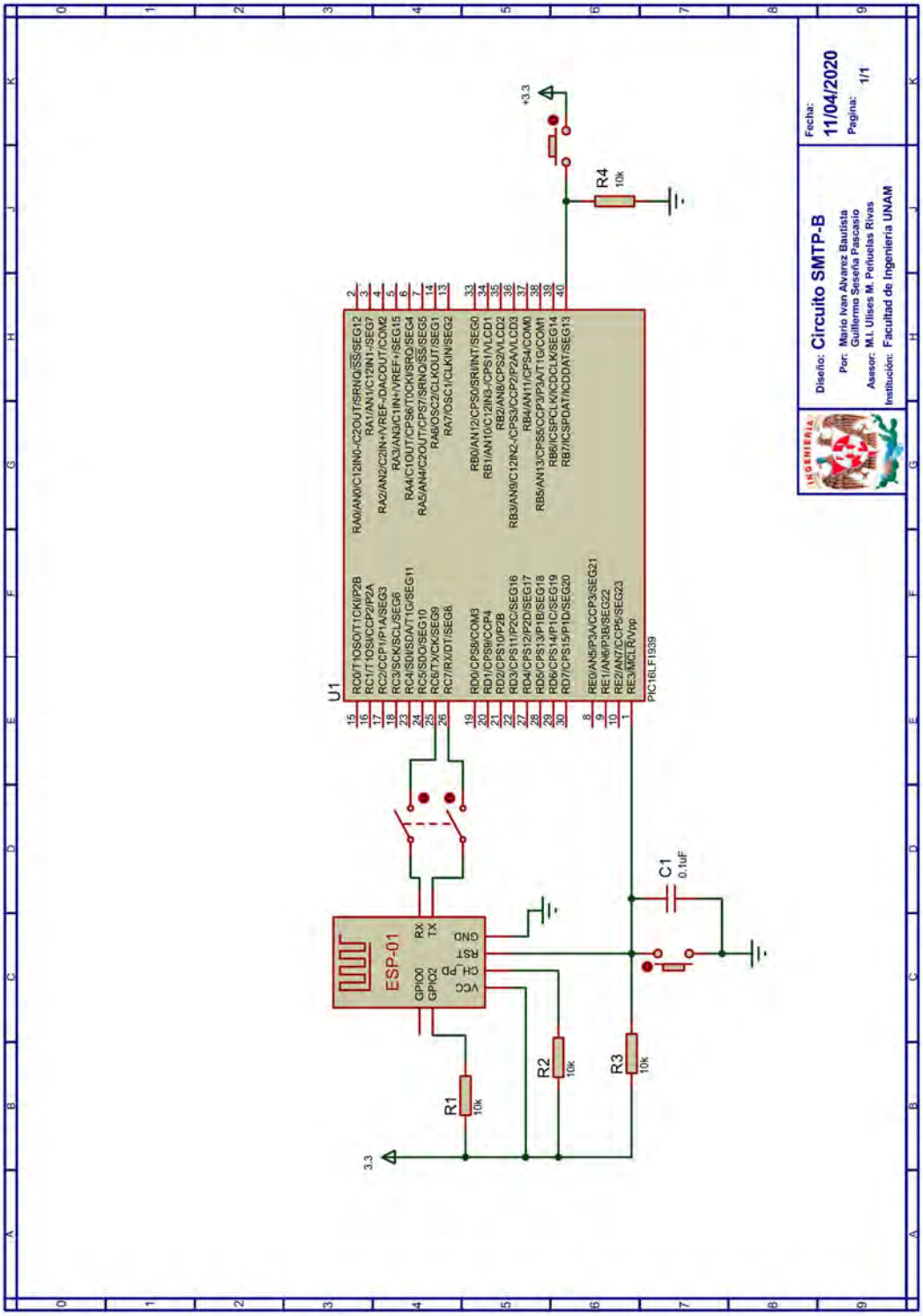


Figura P17-B.7 *Circuito de la práctica 17-B.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 18-A Servidor web: páginas estáticas

Introducción

La aplicación más importante del internet es la web que consiste principalmente en una inmensa cantidad de información almacenada en millones de máquinas en todo el mundo. Esta información se encuentra en forma de páginas web, las cuales son accedidas por usuarios a través de una interfaz gráfica que proporcionan los navegadores web.

Una de las características más relevantes desde la invención de las páginas web es que contienen vínculos para acceder a otras páginas, las cuales pueden estar alojadas en cualquier parte del mundo. Estos vínculos son conocidos como hipervínculos, que en sus inicios era común encontrarlos como textos subrayados, en la actualidad los diseñadores utilizan formas más sofisticadas para que el usuario pueda identificarlos, como uso de botones o imágenes que cambien su apariencia cuando el usuario interactúe con ellos [1].

Un usuario puede acceder de dos maneras a una página web: una es escribiendo la dirección especifica de la página en el navegador y la otra es accediendo con un hipervínculo. La dirección especifica de una página web está dada por un Localizador Uniforme de Recurso (URL-*Uniform Resource Locator*).

Un URL está constituido de tres partes: el protocolo por el cual se accede a un recurso, el nombre DNS del servidor donde se aloja (o dirección IP) y la ruta especifica donde se encuentra dicho recurso [1].

Por ejemplo, el URL de la página de inicio de la facultad de ingeniería es:

https://www.ingenieria.unam.mx/index.php .

En este URL se especifica que la página es accedida mediante el protocolo HTTPS, mientras que *www.ingenieria.unam.mx* es el nombre de dominio en que encuentra el servidor e index.php es el nombre del directorio de la página donde está alojado el recurso.

Los navegadores web son programas que se encargan de obtener el contenido de las páginas, interpretarlo y desplegarlo en pantalla, además son capaces de capturar la interacción que tiene el usuario con la página. Para obtener este contenido, los navegadores web funcionan como clientes que solicitan información a servidores web, esta interacción se realiza bajo el Protocolo de Transferencia de Hipertexto (*HyperText Transfer Protocol*, HTTP).

HTTP es un protocolo de aplicación diseñado especialmente para la trasferencia de información a través de la

web, está basado en la estructura solicitud-respuesta. Este protocolo trabaja sobre TCP y comúnmente a través del puerto 80. En la actualidad es más común la implementación de su versión segura llamada HTTPS (*HyperText Transfer Protocol* Secure) que trabaja por encima de conexiones SSL/TLS, las cuales cifran la información que intercambia el cliente y el servidor, además de autentificar al servidor y/o cliente mediante certificados [1].

HTTP define la sintaxis de las peticiones y respuestas que el cliente y el servidor envían respectivamente, y que hasta están constituidas completamente por líneas de texto ASCII para la versión 1.1 y anteriores[2]. En la actualidad la versión 1.1 aún sigue siendo la más empleada.

En la primera línea de las peticiones se especifica el método, el nombre del recurso que puede incluir su directorio y la versión del protocolo. Los métodos están predefinidos para ejercer cierto tipo de acción en un recurso, aunque siempre dependerá de cómo se haya programado el servidor para atender a las peticiones. En la tabla P18-A.1 se muestra los métodos para la versión 1.1.

En el caso de las respuestas, la primera línea señala la versión del protocolo y un código de estado que indica el resultado de la petición, estos códigos son números de tres cifras junto con una breve etiqueta, que como se muestra en la tabla P18-A.2, están agrupados en cinco tipos de respuestas para la versión 1.1.

Tabla P18-A.1 *Métodos HTTP en su versión 1.1 de acuerdo con RFC7231.*

Método	**Descripción**
GET	Solicita la representación de un recurso
HEAD	Igual que GET, pero con respuesta sin contenido
POST	Crea un nuevo recurso con el contenido de la petición
PUT	Reemplaza la representación de un recurso especifico con el contenido de la petición
DELETE	Borra la representación de un recurso en especifico
CONNECT	Establece un túnel a hacia el servidor identificado por el recurso
OPTIONS	Solicita la descripción de las opciones de comunicación para el recurso solicitado
TRACE	Realizar una prueba de bucle de mensaje a lo largo de la ruta

Tabla P18-A.2 *Códigos de estado HTTP en su versión 1.1, de acuerdo con RFC7231*

Código	Descripción
1xx	(Informativo) Se recibió la solicitud, proceso prosiguió
2xx	(Éxito) La petición se recibió, fue atendida y aceptada correctamente
3xx	(Redirección) Se deben tomar medidas adicionales para completar solicitud
4xx	(Error del cliente) La solicitud contiene una sintaxis errónea o no se puede realizar
5xx	(Error del servidor) el servidor no logró ejecutar una petición valida

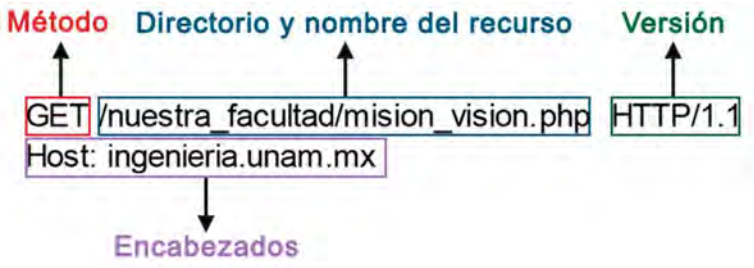


Figura P18-A.1 *Ejemplo de una petición HTTP para obtener una página web.*

Después de la primera línea las peticiones y las respuestas pueden incluir más información con los encabezados, cada uno de éstos está separado por un retorno de carro y salto de línea. Las peticiones y respuestas pueden incluir contenido, esto dependerá si se ha incluido encabezados que lo definen y si lo permite el método o código de estado utilizado.

HTTP envía contenido multimedia mediante MIME, por lo que una página web puede estar constituida de un solo tipo de archivo o una mezcla de diferentes tipos. En general, los navegadores web son intérpretes de documentos HTML capaces de procesar y mostrar cualquiera de los tipos de archivos que estos documentos puedan vincular, sin embargo, para otra clase de archivos MIME, el navegador tiene que recurrir a extensiones conocidas como *plug-ins* o a programas externos [1].

En la figura P18-A.1 se muestra un ejemplo de una petición que se realizaría para obtener la página:

https://www.ingenieria.unam.mx/nuestra_facultad/mision_vision.php

El método GET es el utilizado por los navegadores para obtener una página, el encabezado de *host* es obligatorio y especifica el nombre de dominio al cual

se envía la petición. La repuesta a la anterior solicitud es la mostrada parcialmente en la figura P18-A.2, en la cual se indica que la petición fue ejecutada exitosamente ya que regresa el código de estado igual a 200, muestra algunos encabezados relacionados a la máquina y *software* que generó la respuesta e indica que el contenido de la respuesta es un documento HTML. Al final de la respuesta se encuentra todo el documento HTML.

HTML (Lenguaje de Marcado de Hipertexto, *HyperText Markup Language*) es el lenguaje estándar de las páginas web, el cual no es un lenguaje de programación si no un lenguaje de marcado, que permite al navegador conocer la estructura y el formato de la información que se despliega al usuario [1].

Un documento HTML está constituido de elementos que están delimitados por etiquetas de inicio y de fin, a excepción de elementos vacíos. Cada una de las etiquetas está formada por una directiva que está dentro de corchetes angulares (<>). La diferencia entre una etiqueta de inicio y una de fin, es que las de fin incluyen una diagonal (/) entre el corchete angular a la derecha (<) y la directiva, además, en las etiquetas de

Figura P18-A.2 *Ejemplo de una respuesta de una petición HTTP para obtener una página web.*

inicio se pueden especificar algunos atributos que proporcionan más información de cómo debe ser presentado el elemento. Cada atributo es escrito después de la directiva colocando el nombre del atributo, el signo de igual (=) y su valor entre comillas. El texto que se escriba entre las etiquetas de inicio y fin es información que tomará el formato que se indican en dichas etiquetas, o en caso de que se coloquen elementos entre estas etiquetas se considera como un elemento "hijo" heredando los atributos del principal.

Un documento HTML está limitado por las etiquetas <html> y </html>, para la versión HTML 5.0, que es la versión más actual, es necesario colocar la etiqueta <!DOCTYPE html> al inicio del documento. Un documento HTML está divido en dos principales partes, uno es el encabezado y el otro es el cuerpo. El encabezado está delimitado por las etiquetas <head> y </head>, en esta sección se define la *metada*, información acerca del documento y de documentos externos que sean requeridos. Algunas de los elementos más comunes del encabezado son:

- <title>…</title> Elemento que especifica el nombre del documento HTML, el cual comúnmente es desplegado en la pestaña del navegador.
- <meta> Describe información diversa del documento como el esquema de codificación de caracteres, propiedades del documento o intervalo del tiempo en que se actualiza la página.
- <base> Sirve para definir la dirección URL base del documento. A partir de este URL harán referencia relativa todos los elementos del cuerpo que impliquen un URL.
- <link> Permite enlazar archivos externos al documento.
- <style>…</style> Anexa normas de apariencia de CSS.
- <script>…<script> Anexa una secuencia de comandos que se ejecutan del lado del cliente (en el navegador), también se pueden anexar las secuencias en el cuerpo del documento.

En el cuerpo de un documento HTML, contiene la información que el usuario puede ver, esta sección está delimitada por las etiquetas <body> y </body>. HTML proporciona una gran cantidad de etiquetas que le dan formato a los textos que se desean desplegar en pantalla. Incluyendo tablas, formularios, etc [3, 4].

Inicialmente HTML estaba destinado sólo a marcar la estructura de la información que se muestra al usuario, no poseía suficientes opciones para determinar la apariencia de las páginas, por lo que se agregaron nuevos atributos que especificaban con mayor precisión las fuentes de los textos y apariencia de los elementos de la página. Esto hizo que los

documentos HTML fueran más complejos, difíciles de modificar y especialmente menos portables entre distintos navegadores. Estos problemas se solucionaron al separar la apariencia del contenido, surgiendo el concepto de Hojas de Estilo en Cascada (Cascading Style Sheets , CSS) que fue introducido a HTML en su versión 4.0.

CSS permite definir la apariencia de los elementos del documento HTML de manera compacta, los cuales solo deben hacer referencia a un estilo definido en un archivo CSS, esta simplificación permite a los sitios web mantener un estilo consistente para sus páginas. Los archivos CSS pueden ser agregados a un documento HTML de dos formas: una es a través del elemento <link> como un documento externo al HTML; mientras que la otra es que el archivo esté definido dentro del documento HTML mediante las etiquetas <style> y </style> [1, 3, 4].

Las páginas web están clasificadas en estáticas y dinámicas. Las estáticas son aquellas que siempre que sean solicitadas mostraran el mismo contenido, mientras que, las dinámicas son aquellas que su contenido cambia en función del usuario que accede y/o la interacción que tenga éste con la página. También se consideran como páginas dinámicas aquellas que requieran realizar una consulta a una base de datos o la ejecución de algún programa en el lado del servidor.

La biblioteca ESP permite utilizar al PIC y al módulo como un servidor web, pero con características limitadas ya que el protocolo HTTP no fue diseñado para dispositivos de poca capacidad computacional como es el microcontrolador PIC. La biblioteca permite responder a solicitudes realizadas a través de los métodos GET y PUT, éstos fueron elegidos por ser de los más empleados, además de que apegándose a su uso predefinido son los que mejor se adecuan a las capacidades de programación del PIC. La sintaxis que fue implementada en la biblioteca es la HTTP/1.1, que, aunque permite ejecutar más de una petición por conexión, la biblioteca solo atiende una por conexión ya que después de responder a la petición desconectará al cliente.

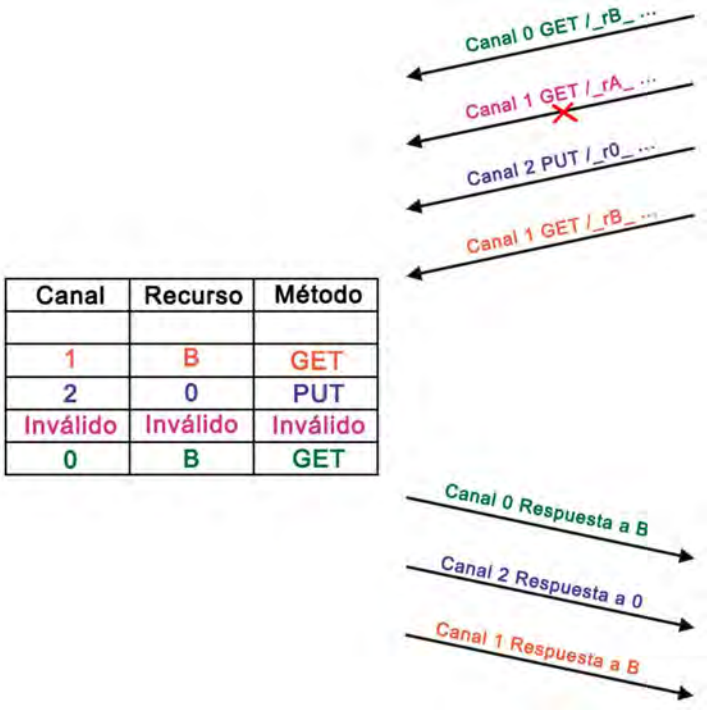
Los recursos web que el PIC-módulo ofrece son cadenas de caracteres que pueden ser documentos HTML o cualquier otro contenido que se pueda definir con *strings* alojados en la ROM y/o la RAM, también los recursos pueden ser acciones que se ejecutan en el programa del PIC. Estos recursos solo pueden ser consultados o actualizados. Los clientes deben referirse a los recursos del PIC-ESP con una URL que tenga la siguiente estructura:

http://{Dirección IP del módulo}:{puerto utilizado por el servidor web}/_R{un carácter ASCII de 7 bits}_

Para la versión IDF, se puede utilizar conexiones seguras y con ello cambiar http por https, sin embargo, el módulo cuenta con un certificado autofirmado, por lo que, al acceder a un recurso, el navegador indicará que no es seguro, aun cuando el intercambio de información este cifrada, ya que no posee con un certificado de una CA.

Los clientes pueden utilizar el servidor desde la red local o de manera remota, pero es necesario realizar un mapeo de puertos para este último, tal como se hizo en la segunda actividad de la práctica de servidor TCP. Para que la biblioteca atienda las peticiones de los clientes, se ha implementado una cola FIFO (*First In, First Out*), que cuando el módulo tiene múltiples conexiones simultaneas, permite la entrega correcta y ordenada de repuestas. Los elementos de la cola están compuestos por el canal de comunicación de procedencia, el recurso que se solicita y el método (GET o PUT).

Cada elemento es insertado al final de la cola durante la interrupción por recepción de datos al recibir una petición para los métodos válidos. Durante esta inserción, se comprueba que no exista algún elemento con el mismo canal de comunicación, en caso de existir, coloca valores inválidos al elemento antiguo (ya que indica que el cliente ha realizado una actualización del recurso que requiere). Cada elemento se desencola durante el envío de una respuesta, acción que se



Canal	Recurso	Método
1	B	GET
2	0	PUT
Inválido	Inválido	Inválido
0	B	GET

Figura P18-A.3 *Ejemplo de ejecución de la cola de atención de peticiones HTTP que ejecuta la biblioteca ESP.*

ejecuta en un *loop* del programa principal del microcontrolador, y que atiende al primer recurso almacenado en la cola. De acuerdo con el método, se otorga cierta respuesta que es enviada al canal que le corresponde, finalmente cierra la conexión del cliente atendido.

En la figura P18-A.3 se ilustra un escenario del funcionamiento de la cola de la biblioteca. En este escenario, el programa principal del microcontrolador está ejecutando otras tareas aparte de la atención de peticiones y en ese momento recibe tres nuevas peticiones, la cuales son almacenadas en la cola. En lo que la ejecución del programa llega a la sección de respuestas, el cliente conectado al canal 1 cierra su conexión antes ser atendido, liberando el canal para un nuevo cliente. Un nuevo cliente solicita un recurso totalmente diferente, la biblioteca se percata que en la cola ya existe un elemento con el mismo canal, por lo que, invalida el antiguo elemento y guardar la nueva petición al final de la cola. De esta forma cuando se atiendan las respuestas, se le entregará al nuevo cliente el recurso que solicitó y no aquel que había solicitado el anterior cliente que utilizó el mismo canal y no obtuvo respuesta a su petición.

Los recursos del PIC son definidos en un *loop* del programa principal. Este *loop* está constituido de un *switch* que evalúa la variable que le corresponde al recurso que se encuentra en el primer elemento de la cola de atención de peticiones. Cada uno de los casos utiliza un caracter ASCII que representa un recurso disponible y especifica la respuesta que se otorgará al cliente cuando sea solicitado dicho recurso. Es necesario un caso por defecto el cual se encargue recorrer la cola cuando se invalide algún elemento.

Cuando un cliente solicite al módulo un recurso con un URL de distinta composición a la mencionada o solicite un recurso que no haya sido definido en el *switch* de atención, se considera como un recurso inválido, por lo que la respuesta será que el recurso solicitado no fue encontrado. Cada recurso que sea definido podrá ser solicitado por un solo método ya sea GET o PUT.

Todas las posibles respuestas HTTP que la biblioteca ESP puede generar están descritas en la tabla P18-A.3. Para las respuestas que se construyen libremente se recomienda enviar *strings* alojados en la ROM del microcontrolador, los cuales no deben exceder los 2047 bytes, recordar que todos los *strings* cuentan con el carácter finalizador \0.

Tabla P18-A.3 *Respuestas HTTP que puede enviar la biblioteca ESP.*

Petición HTTP	**Respuesta HTTP**
GET para un recurso válido	Respuesta personalizable
	HTTP/1.1 200 OK Content-Type: text/plain; charset=UFT-8 Content-Length: 0
	HTTP/1.1 200 OK Content-Length: X Content-Type: application/json; charset=utf-8 {Contenido}
GET para un recurso invalido	HTTP/1.1 404 Not Found Content-Type: text/plain; charset=UFT-8 Content-Length: 0
GET para favicon.ico	HTTP/1.1 200 OK Content-Type: image/x-icon Content-Length: 0
GET para un recurso definido para PUT	HTTP/1.1 405 Method Not Allowed Content-Type: text/plain; charset=UFT-8 Content-Length: 0
PUT para un recurso válido	HTTP/1.1 204 No Content Content-Type: text/plain; charset=UFT-8 Content-Length: 0
PUT para un recurso invalido	HTTP/1.1 404 Not Found Content-Type: text/plain; charset=UFT-8 Content-Length: 0
PUT para un recurso definido para GET	HTTP/1.1 405 Method Not Allowed Content-Type: text/plain; charset=UFT-8 Content-Length: 0

Objetivo

Establecer al módulo como un servidor web para proporcionar una página web estática.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_HTTP_CLIENT.

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 capacitor de 0.1 μF
- 1 pantalla LCD 16x2

Descripción

Se establecerá al módulo ESP como un servidor web, el cual proporcionará a los clientes una página web estática. Para esta práctica se propone un documento HTML con el fin de mostrar que es posible otorgar páginas web que integren CSS.

La respuesta que el módulo otorgará a los clientes que soliciten la página se muestra parcialmente en la figura P18-A.4.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UFT-8
Content-Length: 1343

<!DOCTYPE html>

…

</html>
```

Figura P18-A.4 *Respuesta parcial a la solicitud de la página estática de la práctica 18-A.*

La página que se enviará está compuesta por los elementos visuales que se muestran en la figura P18-A.5. La estructura del *body* está constituida por un *header* en donde se muestra el nombre de la página; dos *img* que corresponde al escudo de la UNAM y al de la Facultad de Ingeniería respectivamente; un *nav* que contiene un *button* que funciona como un hipervínculo para la página https://www.ingenieria.unam.mx; *main*

Figura P18-A.5 *Composición de la página que se envía en la práctica 18-A*

que contiene un texto; y *footer* que contiene una fecha.

En el documento HTML, primero se define que la versión utilizada es HTML 5.0, posteriormente se establece “Inicio” como el nombre que se mostrará en la pestaña del navegador y se especifica las reglas de estilo para los elementos del documento HTML; que incluye una clase para modificar la apariencia de un botón al pasar el cursor sobre él (ver figura P18-A.9).

Posteriormente en el documento HTML se define el cuerpo de la página, utilizando los elementos ya mencionados. Las imágenes que se incluyen no se encuentran almacenadas en el documento si no que se encuentran alojadas en otros servidores. El escudo de la UNAM se encuentra en un servidor de la Facultad de Medicina y el escudo de la Facultad de Ingeniería pertenece a uno de sus servidores. El botón incrustado en el *nav* redirecciona al sitio de la Facultad de Ingeniería.

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica es el mostrado en la figura P18-A.10 y en la figura P18-A.11, en el que se debe ingresar nombre y contraseña de un AP. Se recomienda que el AP tenga acceso a internet para que los clientes locales puedan visualizar las imágenes alojadas en otros servidores.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P18-A.12.

Resultados

Al ejecutarse el código del uC PIC lo primero que se observará será la dirección IP que se desplegará en la LCD, con esta dirección se accederá desde un navegador o puede ser utilizada para mapear puertos y permitir su acceso de manera remota.

En el navegador web, se escribe http://{dirección IP}/_r0_, al realizarlo se mostrará la página, tal como se ilustra en la figura P18-A.6. Si se intenta acceder a otro recurso que no sea el de la página entonces se mostrará el error 401 correspondiente a recurso no encontrado (ver figura P18-A.7).

Si se accede a las herramientas de desarrolladores del navegador en la sección de red (*Network)*, al acceder a la página que otorga PIC-ESP, se visualizará la respuesta que ha emitido el módulo, además se puede observar que las imágenes que incluye la página son recursos independientes que son obtenidos de otros servidores (ver figura P18-A.8).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2YdRXF4

Figura P18-A.6 *Página que otorga el servidor web del PIC-ESP.*

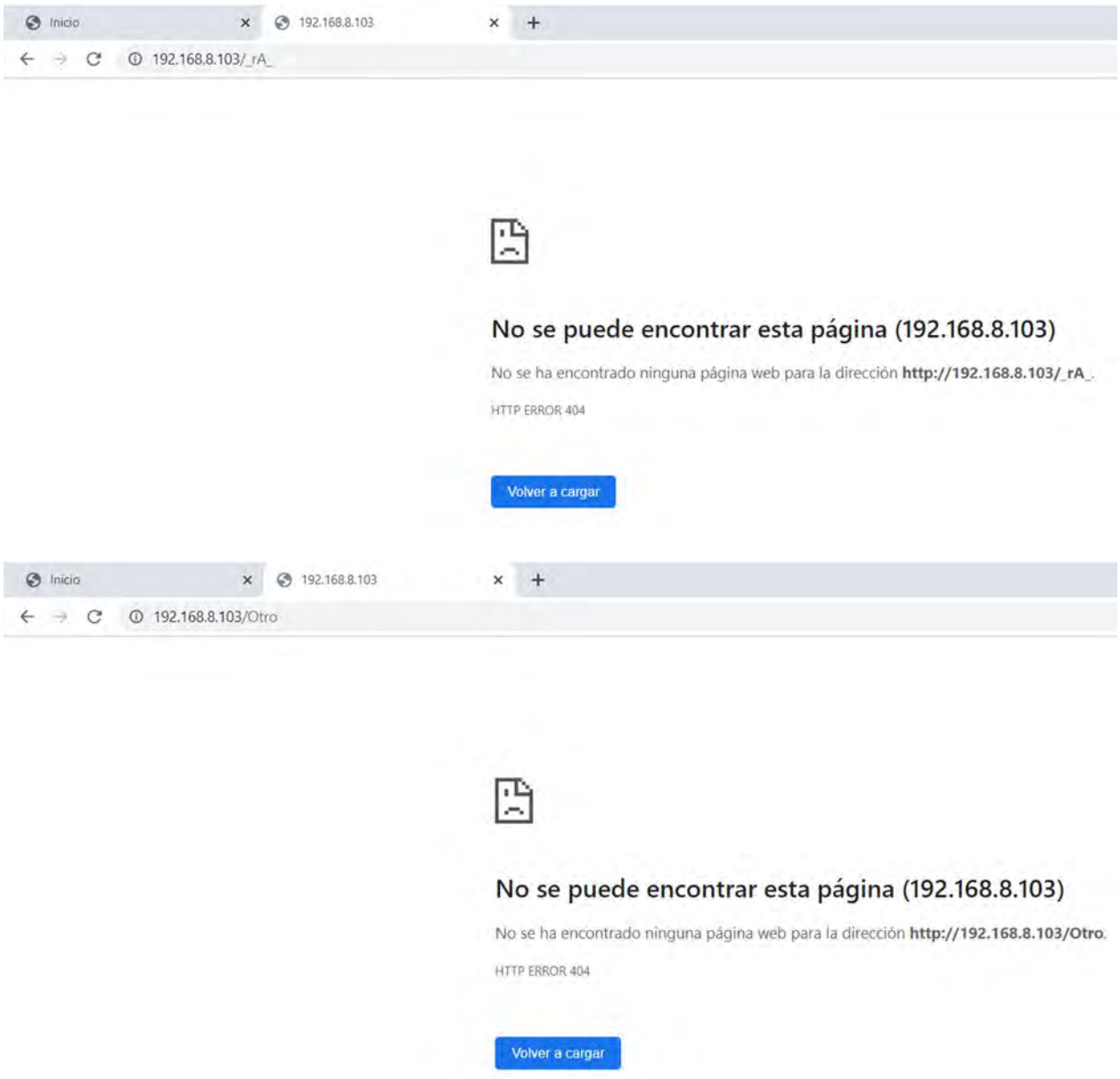


Figura P18-A.7 *Resultados de cuando se accede a un recurso no definido en el programa del PIC o cuando se ingresa a un recurso que no tiene la estructura permitida por la biblioteca ESP.*

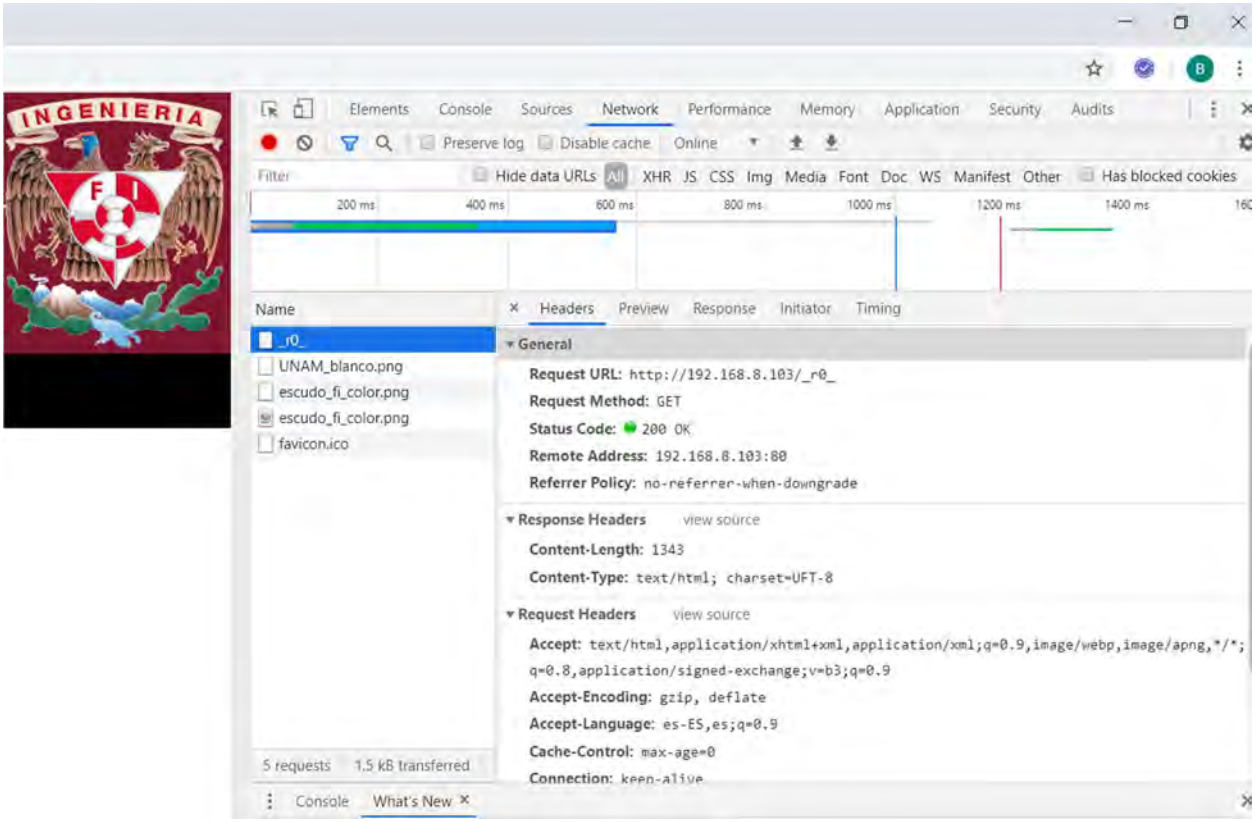


Figura P18-A.8 *La respuesta que generó el PIC-ESP mostrada en herramientas para desarrolladores del navegador Google Chrome.*

```
<!DOCTYPE html><html>
<head>
<meta charset="utf-8">
<title>Inicio</title>
<style type="text/css">

/*Define estilo de los elementos del documento HTML*/
body{background-color:#EBEBEB;margin: 0;height: 100vh;}
header{background-color:#811832; text-align: center;}
nav{background-color:black;text-align: left;}
main{background-color:white;font-size:22pt;text-align: center;min-height: 50vh;}
footer{background-color:#3F3F3F;text-align: center;}
h1{font-family:Georgia;font-size:40pt;color:white;}
h2{font-size:26pt;color:white;}

/*Define estilo de las clases*/
.c1{width:96%;margin-left:2%;overflow:hidden;}
.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:white;background-color:black;}
.b_1:hover{color:black;background-color:white;}
.t1 {table-layout:fixed;width:100%;border:0;}
.i1{width:182px;height:205px}

</style>
</head>
<body>

<header class="c1">
<table class="t1"><tr>
<td><img class="i1" src="http://www.facmed.unam.mx/img_b/UNAM_blanco.png" align="left"></td>
<td><h1>Biblioteca ESP</h1></td>
<td><img class="i1" src=
"http://www.ingenieria.unam.mx/nuestra_facultad/images/institucionales/escudo_fi_color.png"
align="right"></td>
</tr></table>
</header>

<nav class="c1">
<table><tr>
<td><button class="b_1" onclick="location='https://www.ingenieria.unam.mx/'"
>Facultad de Ingeniería</button></td>
</tr></table>
</nav>

<main class="c1">
<br>
<br>
<p>Contenido de la pagina.</p>
</main>

<footer class="c1">
<h2>Febrero 2020</h2>
</footer>

</body>
</html>
```

Figura P18-A.9 *Documento HTML utilizado en la práctica 18-A.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char IP[21]={0};
int recurso=0;
char str_content_length[]="0000";
const char respuesta[]=
"HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=UFT-8\r\nContent-Length: ";
const char pagina_0[]=
"<!DOCTYPE html><html><head><meta charset=\"utf-8\"><title>Inicio</title><style type=\"text/c
ss\">body{background-color:#EBEBEB;margin: 0;height: 100vh;}header{background-color:#811832;
 text-align: center;}nav{background-color:black;text-align: left;}main{background-color:whit
e;font-size:22pt;text-align: center;min-height: 50vh;}footer{background-color:#3F3F3F;text-al
ign: center;}h1{font-family:Georgia;font-size:40pt;color:white;}h2{font-size:26pt;color:whit
e;}.c1{width:96
%
;margin-left:2%;overflow:hidden;}.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:wh
ite;background-color:black;}.b_1:hover{color:black;background-color:white;}.t1 {table-layout:
fixed;width:100
%
;border:0;}.i1{width:182px;height:205px}</style></head><body><header class=\"c1\"><table clas
s=\"t1\"><tr><td><img class=\"i1\" src=\"http://www.facmed.unam.mx/img_b/UNAM_blanco.png\" al
ign=\"left\"></td><td><h1>Biblioteca ESP</h1></td><td><img class=\"i1\" src=\"http://www.inge
nieria.unam.mx/nuestra_facultad/images/institucionales/escudo_fi_color.png\" align=\"right\">
</td></tr></table></header><nav class=\"c1\"><table><tr><td><button class=\"b_1\" onclick=\"l
ocation='https://www.ingenieria.unam.mx/'\">Facultad de Ingeniería</button></td></tr></table>
</nav><main class=\"c1\"><br><br><p>Contenido de la pagina.</p></main><footer class=\"c1\"><h
2>Febrero 2020</h2></footer></body></html>"
;
```

Figura P18-A.10 *Primera parte del código de la práctica 18-A.*

```
void main()
{
   //Inicialización de la LCD
   lcd_init();
   //Posiciona escritura en la LCD
   lcd_gotoxy(2,1);
   //Imprime en la LCD
   lcd_putc("Direccion IP:");
   //Inicialización del módulo Wi-Fi
   begin_esp();
   //Conexión al punto de acceso indicado
   begin_station("Estacion",SSID,password);
   //Obtiene dirección IP local
   get_ip_station(IP,sizeof(IP));
   //Posiciona escritura en la LCD
   lcd_gotoxy(2,2);
   //Imprime en la LCD la dirección IP
   printf(lcd_putc,"%s",IP);
   /*Crea el servidor web para el puerto 80 (por defecto para navegadores),
   permite cuatro conexiones simultáneas con un timeout de 120s y asigna
   la variable que apunta al recurso del primer elemento de cola de atención
   de peticiones HTTP*/
   create_http_server(80,4,120,&recurso);
   while(1)
   {
      //Actualiza el recurso solicitado para el primer elemento de la cola
      http_upgrade_resource();
      switch (recurso)
      {
         //Envia respuesta para el recurso (página) _r0_
         case '0':
         http_send_const(respuesta);
         http_send(str_content_length);
         http_send("\r\n\r\n");
         http_send_const(pagina_0);
         http_finish(1);
         break;

         //Atiende elementos inválidos
         default:
         http_default_resource();
         break;
      }
      delay_ms(50);
   }
}
```

Figura P18-A.11 *Segunda parte del código de la práctica 18-A.*

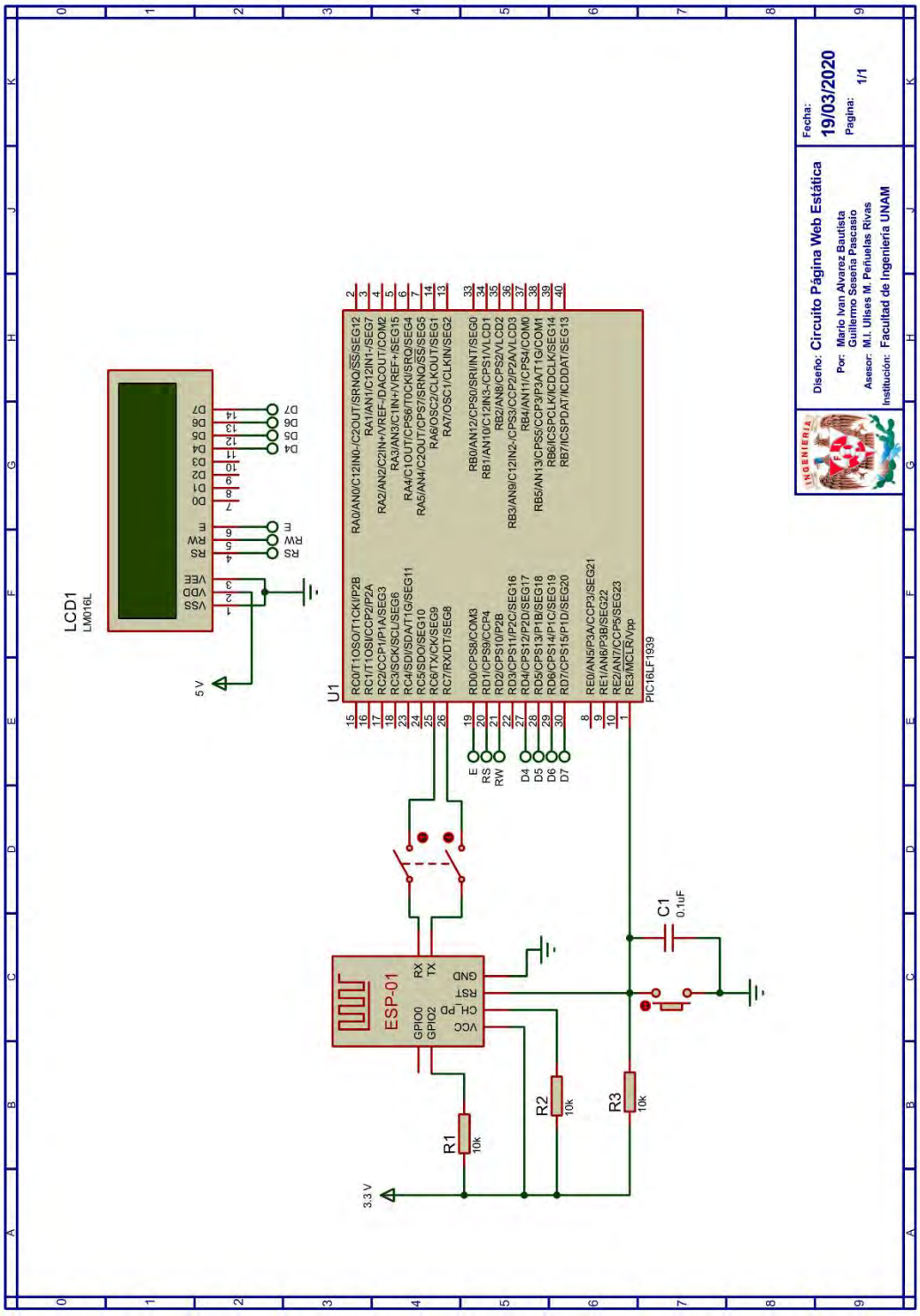


Figura P18-A.12 *Circuito de la práctica 18-A.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.
2. Colaboradores de MDN. *HTTP*. [Citado 2020 Marzo]; Disponible en:https://developer.mozilla.org/es/docs/Web/HTTP.
3. Macaulay, M., *Introduction to Web Interaction Design with HTML and CCS*. 2018: Taylor & Francis Group, LLC.
4. Deitel, P., H. Deitel, and D. Abbey, *Internet & World Wide Web Cómo programar*. Quinta ed. 2014.

Práctica 18-B Servidor web: páginas dinámicas

Introducción

Las páginas dinámicas permiten a los usuarios acceder a aplicaciones y servicios, como envío de correos, herramientas de trabajo colaborativo, compras en internet, computación en nube y servicios *streaming*. Este tipo de páginas son receptivas y responsivas, por lo que implementan la ejecución de secuencias de comandos llamados *scripts*, los cuales pueden ejecutarse en el navegador (cliente) o en el servidor.

Los *scripts* que se ejecutan en el lado del cliente (navegador) están definidos dentro del documento que recibe el navegador. En HTML éstos se definen entre las etiquetas <script> y </script>, siendo el lenguaje más popular para este tipo de *script* Javascript. Los *scripts* que se ejecutan en el servidor, evidentemente están definidos en el servidor, para este caso, en HTML sólo se incluye el nombre del *script* en el atributo *action* del elemento desencadenador de la ejecución, el lenguaje más popular para este tipo de *script* es PHP (*Hypertext Preprocessor*).

La elección de qué tipo de *script* se debe utilizar depende de la aplicación, si la página puede responder con solo la información que el usuario ingrese, entonces perfectamente se puede programar con *scripts* en el navegador. En el caso de que la aplicación necesite consultar información en el servidor entonces requerirá que los scripts se programen en el servidor, muchas otras aplicaciones requieren que se programe tanto en el cliente como en el servidor.

Para que las páginas web pudieran competir con las aplicaciones de escritorio, fue necesario mejorar la receptividad y el rendimiento de éstas. Las mejoras han sido posibles por la conjunción de varias tecnologías, uno de los conjuntos más populares es Javascript Asíncrono y Xml (*Asynchronous JAvascript and Xml*, AJAX) que permite al usuario interactuar con una página de manera independiente a la transferencia de información entre el navegador y el servidor [1].

En un diseño clásico (síncrono) una página puede actualizar su contenido con información del servidor sólo mediante la recarga completa de la página, obstaculizando al usuario usar la página durante dicha actualización, mientras que con AJAX sólo se actualizan las secciones de interés de manera paralela a la interacción del usuario [1, 2].

AJAX realiza peticiones HTTP de manera asíncrona mediante el uso de Javascript; intercambia información en formato de Lenguaje de Marcado Extensible (*eXtended Markup Language*, XML), aunque se puede usar otros formatos como JSON; y actualiza el documento

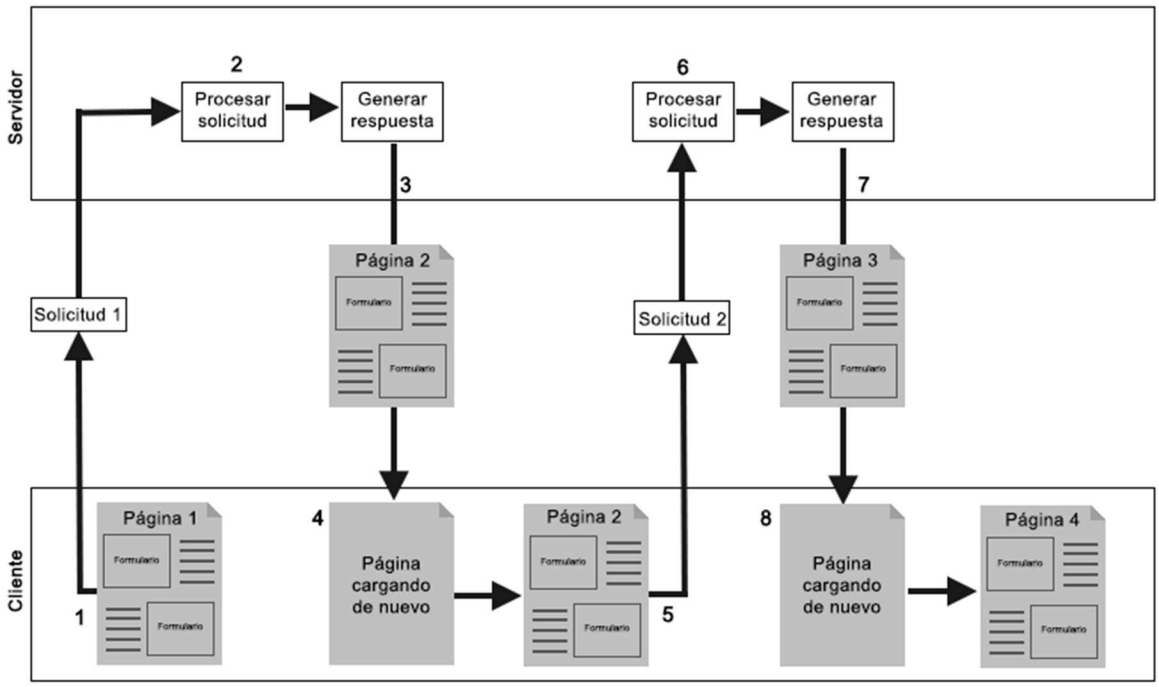


Figura P18-B.1 *Aplicación Web clásica que vuelve a cargar la página en cada interacción del usuario [2].*

HTML mediante el Modelo de Objetos de Documento (*Document Object Model*, DOM). Para comprender mejor la diferencia entre una página con un diseño tradicional y un diseño con AJAX se toma el ejemplo mostrado en [2], que consiste en el envío de formularios a un servidor web.

Para un diseño web clásico (ver figura P18-B.1) se siguen los siguientes pasos:

1. El navegador realiza una solicitud HTTP al servidor web, en la cual incluye la información que ha proporcionado el usuario en el formulario.
2. El servidor procesa la petición.
3. El servidor responde a la petición incluyendo el contenido de la página que se mostrará al usuario tras el llenado del formulario.
4. El navegador web carga la nueva página, bloqueando al usuario la interacción hasta que la página se haya cargado por completo.
5. El proceso anterior se repite cada vez que el usuario llene un formulario.

Para el diseño web con AJAX (ver figura P18-B.2) se siguen los siguientes pasos:

1. El navegador usa Javascript para crear un objeto XMLHttpRequest que controla la solicitud.
2. El navegador realiza la petición HTTP incluyendo la información que ha suministrado el usuario usando el formato XML.

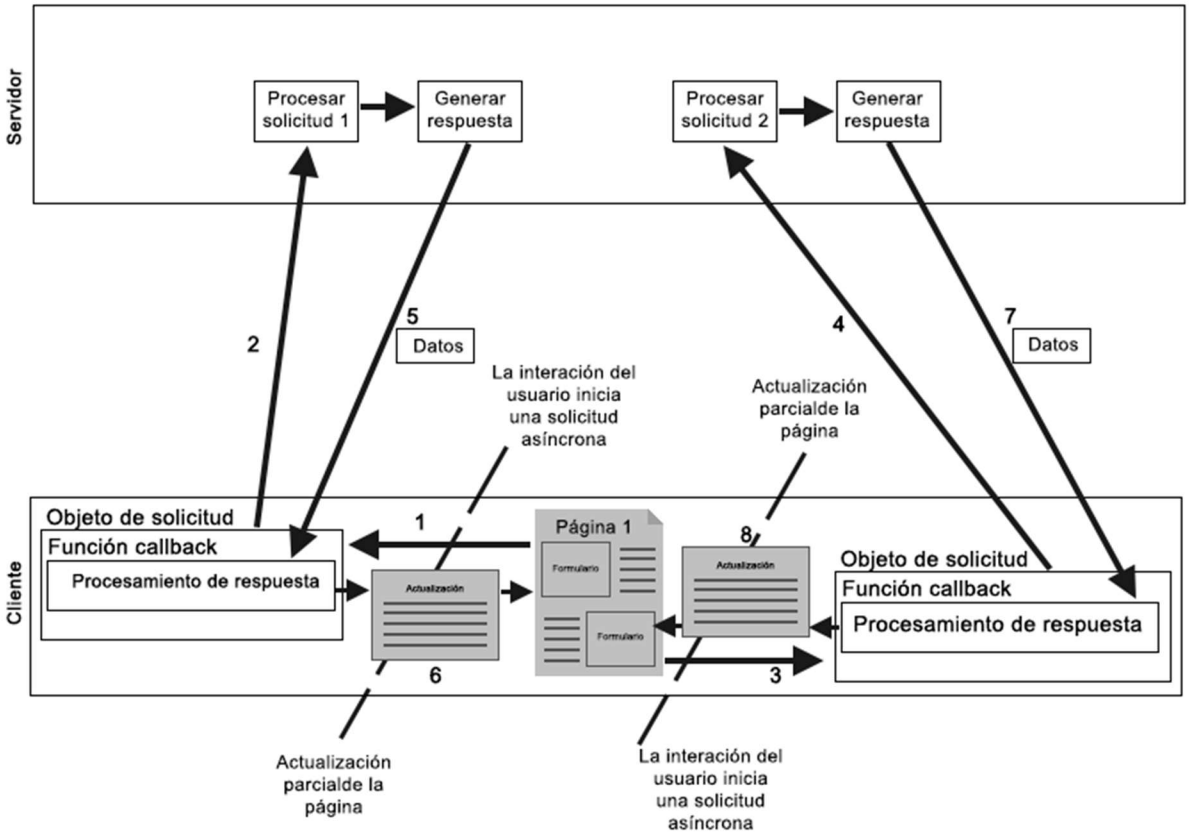


Figura P18-B.2 *Aplicación Web compatible con Ajax que interactúa con el servidor en forma asíncrona [2].*

3. Como es un proceso asíncrono el usuario puede seguir interactuando con la página y con ello generar una petición HTTP adicional sin esperar que la anterior solicitud ya haya sido atendida, por lo que el navegador genera un nuevo objeto XMLHttpRequest para manejar la nueva solicitud.
4. La nueva solicitud HTTP se envía al servidor.
5. El servidor responde a la primera petición.
6. El primer objeto XMLHttpRequest llama a una función *callback* que procesa la información recibida, mediante DOM la información modifica el contenido de los elementos del documento HTML que se actualizan, ejecutando así, una actualización parcial a la página, en la que en ningún momento el usuario pierde la interacción con la página.
7. El servidor responde a la segunda petición.
8. El segundo objeto XMLHttpRequest se encarga de

actualizar parcialmente a la página.

La biblioteca ESP puede ser utilizada para aplicaciones que requieran proporcionar páginas web dinámicas, siempre y cuando los *scripts* puedan incluirse en el documento HTML u otros archivos que puedan ser enviados como *strings*. Es importante mencionar que sólo se pueden utilizar *scripts* del lado del navegador. Lo equivalente a los *scripts* del lado del servidor sería la programación que se realice en el programa del microcontrolador PIC, pero se debe establecer alguna forma en el que el cliente le indique al PIC-ESP que ejecute algún proceso.

En el caso de páginas síncronas, la biblioteca puede enviar documentos HTML que han sido modificados de acuerdo con la interacción del usuario, sin embargo, el envío de información del usuario al servidor está restringido, ya que las peticiones POST no están implementadas en la biblioteca. Es posible enviar información a través del URL utilizando el método GET, ya que la biblioteca almacena el directorio y nombre del recurso solicitado en un *buffer*, no obstante, al no ser un recurso válido para la biblioteca, en el navegador se mostrará el error de recurso no encontrado. En consecuencia, con la biblioteca solo es posible que el cliente pueda indicarle al servidor que muestre un contenido diferente o actualizado cada vez que recargue la página u oprima algún botón que direccione al mismo u otro recurso disponible en el servidor, siendo posible que el cliente pueda controlar estados en el servidor.
Para las páginas asíncronas, es posible implementar aplicaciones que utilicen tecnologías como AJAX, ya que el *script* puede ser incluido en el documento que se le envía al cliente (navegador). Para esta situación es posible que el cliente envié información al servidor mediante peticiones PUT y utilizando JSON como formato de datos.

La biblioteca es capaz de almacenar y manejar datos en formato JSON al activar la herramienta E_JSON, pero con restricciones. Para manejar este formato, las peticiones que se realicen al módulo deben especificar dos encabezados:

- Content-type: debe tener el valor de application/json; charset=utf-8;
- Content-length: debe tener un valor diferente a nulo.

Sólo se puede alojar la representación en texto de pares (claves y valores) simples como *strings*, números y valores booleanos, no es posible manejar arreglos, ni matrices, ni listas, ni objetos.

Los valores de cada clave almacenada pueden ser consultados y comparados como strings, por ejemplo, el siguiente mensaje es válido para ser recibido: {num: 0, msj:”mensaje”, “valor”:true}

Del anterior mensaje, se puede extraer el *string* del valor de la clave msj, el cual es "mensaje".

Como se señaló en la práctica 18-A, la biblioteca dispone de funciones que atienden a peticiones que utilicen el método PUT con respuestas definidas, que facilita el diseño de este tipo de aplicaciones.

Objetivo

Establecer al módulo como un servidor web para proporcionar páginas web dinámicas.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_HTTP_CLIENT

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 pantalla LCD 16x2
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 56 Ω

Descripción

En esta práctica se establecerá al módulo ESP como un servidor web que otorgará páginas web dinámicas. La práctica está dividida en dos actividades, en la primera se mostrará como otorgar páginas dinámicas de manera síncrona, mientras que en la segunda se mostrará como otorgar una página web dinámica asíncrona utilizando la tecnología AJAX.

I Página dinámica síncrona

Para la primera actividad, el conjunto de páginas proporcionará al cliente dos opciones, una es el control de un LED y la otra es visualizar el valor de un contador, que está incrementando en el *loop* del programa principal del PIC. Este valor podría ser de alguna señal analógica, sin embargo, se seleccionó un contador para que se apreciará con facilidad la frecuencia con que se puede actualizar la información de una página síncrona.

```
<!DOCTYPE html><html>
<head>
<meta charset="utf-8">
<title>Inicio</title>
<style type="text/css">

/*Define estilo de los elementos del documento HTML*/
body{background-color:#EBEBEB;margin: 0;height: 100vh;}
header{background-color:#811832; text-align: center; min-height: 25vh;}
nav{background-color:black;text-align: left;}
main{background-color:white;font-size:22pt;text-align: center;min-height: 50vh;}
footer{background-color:#3F3F3F;text-align: center;}
h1{font-family:Georgia;font-size:60pt;color:white;}
h2{font-size:26pt;color:white;}

/*Define estilo de las clases*/
.c1{width:96%;margin-left:2%;overflow:hidden;}
.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:white;background-color:black;}
.b_1:hover{color:black;background-color:white;}
.b_2{padding:6px;font-family:Georgia;font-size:20pt;color:black;background-color:#fff57a;}
.b_2:hover{color:#fff57a;background-color:black;}
.t1 {table-layout:fixed;width:100%;border:0;}

</style>
</head>
<body>

<header class="c1">
<table class="t1"><tr>
<td><h1 onclick="location='_ri_'">Biblioteca ESP</h1></td>
</tr></table>
</header>

<nav class="c1">
<table><tr>
<!--Al presionar alguno de los botones redirecciona a otra página del servidor web-->
<td><button class="b_1" onclick="location='_rl_'">LED</button></td>
<td><button class="b_1" onclick="location='_rc_'">Contador</button></td>
</tr></table>
</nav>

<main class="c1">
<br>
<br>
<p>Diríjase al control del LED o al contador.</p>
</main>

<footer class="c1">
<h2>Febrero 2020</h2>
</footer>

</body>
</html>
```

Figura P18-B.3 *Documento HTML de página de inicio síncrona.*

El conjunto de páginas está conformado por 5 páginas que cuentan con estructura semejante y estilo que la página diseñada en la práctica 18-A. La diferencia entre cada una de ellas es el contenido del elemento *main* y el título de la página, de modo que cuando el cliente interactúe con ellas observe un contenido diferente.

Una de las páginas será utilizada como página de inicio, cuyo código se muestra en la figura P18-B.3. Esta página sólo mostrará el acceso a alguna de las dos opciones que se ofrecen. Si se presiona el botón de la opción del control del LED, la página redireccionará al cliente a otro recurso del servidor (PIC-ESP) para mostrarle la página correspondiente. Lo anterior es porque de manera síncrona, sólo se le puede solicitar al servidor que realice alguna acción mediante la solicitud de un recurso a través del método GET o mediante el envío de información a través del método POST, el cual no fue implementado en la biblioteca.

Para ambas opciones el contenido de la página se deberá enviar al cliente (navegador) y éste deberá desplegarlo nuevamente. Para que el servidor pueda distinguir las acciones del cliente, se le asignó un recurso a cada acción (una página). En la tabla P18-B.1 se muestra la página que le corresponde a cada recurso al cual el cliente puede acceder.

El código de la página del control tiene el mismo código HTML que la de inicio, sólo cambia el elemento *main* (ver figura P18-B.4), en el que se agregaron dos botones que redireccionan a las páginas de encender o apagar led, las cuales otorgarán al cliente la misma página, pero con el párrafo del estado del LED modificado.

Tabla P18-B.1 *Recursos utilizados para las páginas dinámicas síncronas.*

Recurso	Página
_ri_	Página de inicio
_rl_	Página de control de estado del LED mencionando el estado del LED
_re_	Página de control de estado del LED mencionando que el LED está encendido (Enciende LED)
_ra_	Página de control de estado del LED mencionando que el LED está apagado (Apaga LED)
_rc_	Página de visualización del contador, solicita actualizarse cada 5 segundos para mostrar el valor

```
<main class="c1">
<br>
<br>
<!--Para encender o apagar el LED redirecciona a otra página del servidor web-->
<button class="b_2" onclick="location='_re_'">Encender</button>
<button class="b_2" onclick="location='_ra_'">Apagar</button>
<br>
<br>
<p>LED Encendido/Apagado</p>
</main>
```

Figura P18-B.4 *Main de la página que controla el estado del LED.*

El código de la página que muestra el valor del contador es similar a la de la página de inicio, pero cuenta con un *main* distinto (ver figura P18-B.5).

```
<main class="c1">
<br>
<br>
<p>Contador:</p>
<p>0000</p>
</main>
```

Figura P18-B.5 *Main de la página que muestra el valor del contador.*

Además, para que la página se recargue cada 5 segundos en el encabezado del documento HTML se ha agregado:

```
<meta http-equiv="refresh" content="5">
```

El valor mínimo de recarga que se puede establecer es de un segundo, sin embargo, resulta muy complicado que el cliente pueda controlar la página con valores pequeños ya que durante la carga del contenido de la página se bloquea la interacción del cliente. Cada vez que el navegador realiza la petición de actualización, el servidor leerá el valor actual del contador y lo incluirá en el documento HTML.

Ya que el código HTML es casi igual en todas las páginas de esta actividad, en el código de microcontrolador PIC se ha seccionado ésta en varias partes para que no se tenga que almacenar individualmente el contenido de cada uno de los recursos, lo que aumentaría el consumo de memoria.

Cada vez que se solicite el recurso, el servidor enviará secciones del código hasta completar el correspondiente documento HTML. Para cada recurso solicitado se utiliza la misma respuesta HTTP que la práctica 18-A, evidentemente con longitudes diferentes, las cuales se calculan e incluyen a la respuesta durante la ejecución del código del PIC. Es posible enviar directamente el documento HTML sin definir la respuesta HTTP, dejando que el navegador lo interprete, sin embargo, no es recomendable ya que algún

navegador podría no interpretarlo como un documento HTML, que es el caso como se desea aquí.

II Página dinámica asíncrona

Para la segunda actividad el módulo proporcionará una página web dinámica, la cual actualiza su contenido mediante peticiones HTTP asíncronas, que se ejecutan utilizando la tecnología AJAX. En esta actividad el módulo proporcionará un recurso para otorgar el documento HTML completo de la página y ofrecerá otros recursos para proveer de secciones de código HTML que sirven para actualizar el contenido de la página, además ofrece la ejecución de algunas acciones en el servidor. Los recursos utilizados en esta práctica se muestran en la tabla P18-B.2, que señala el método con que deben ser solicitados, el contenido de la petición, la descripción del recurso y el contenido de la respuesta que otorga el PIC-ESP.

La página principal tendrá el mismo estilo y estructura que el de las dinámicas asíncronas, solo cambia el *main* en el que se ha agregado un *div* cuyo contenido será sustituido mediante las peticiones asíncronas.

Tabla P18-B.2 *Recursos utilizados para las páginas dinámicas asíncronas.*

Recurso	Método con el que se accede	Contenido de la petición	Descripción	Contenido de la respuesta
_r0_	GET	Sin contenido	Proporciona el código completo de la página	HTML
_rl_	GET	Sin contenido	Proporciona el contenido de la página para controlar el LED	HTML
_rp_	PUT	Mensaje en formato JSON {LED:”on/off”}	Enciende o apaga el LED	Sin contenido
_rc_	GET	Sin contenido	Proporciona contenido de la página para observar el valor del contador	HTML
_re_	GET	Sin contenido	Proporciona contenido de la página para enviar mensajes que se muestran en la LCD	HTML
_rv_	GET	Sin contenido	Proporciona valor del contador	HTML
_rm_	PUT	Mensaje en formato JSON {Msj:”mensaje”}	Registra el mensaje en un *string* y lo muestra en la LCD	Sin contenido

Para que la página pueda utilizar AJAX, en el encabezado del documento HTML se ha incluido un *script* de Javascript, cuyo funcionamiento sólo se explica de manera general, para más información de cómo utilizar Javascript puede consultar [2]. Todo el código del documento HTML se muestra en la figura P18-B.6, figura P18-B.7 y figura P18-B.8.

La página proporcionará tres opciones, una es el control de un LED, otra muestra el valor de un contador del programa del PIC y la tercera es el envío de mensajes que se mostrarán en la LCD. Al presionar el botón de algunas de las tres opciones, se ejecutará una función en Javascript llamada `rsqrCont`, la cual realiza una petición HTTP asíncrona mediante el método GET a uno de los recursos que proporciona código HTML para actualizar la sección del *main* de la página. Si la petición se ejecutó correctamente sustituirá el contenido del *div* por el contenido de la respuesta.

Para el caso del control del LED, se mostrarán dos botones, uno para encenderlo y otro para apagarlo. Al presionar alguno de estos botones se ejecuta la función `rsqrSend`, que ejecuta una petición HTTP asíncrona mediante el método PUT al recurso que controla el estado de LED, la petición tendrá contenido en formato JSON cuyo mensaje es {Msj:"on"} para encender el LED y {Msj:"off"} para apagarlo, si la petición se ejecuta adecuadamente entonces el texto que indica el estado del LED cambiará.

Cuando se solicite la opción de contador y se obtenga correctamente el contenido de esta opción, se activará la función `ini`, que establece que en 500 milisegundos se ejecutará la función que realizará una petición HTTP asíncrona mediante el método GET para solicitar el valor actual del contador, si se ejecuta bien, sustituye el texto de un párrafo con el valor obtenido de la respuesta. Cada vez que se cambie de opción, el temporizador es limpiado evitando que la función sea ejecutada aun cuando ya no se muestre esta opción en la página.

Finalmente, en la opción de envío de mensajes se mostrará un elemento tipo *input*, en donde se ingresarán los mensajes que se enviarán al PIC, además, se muestra un botón que realiza dicho envío. Al presionar el botón se ejecutará la función `rsqrSend`, la cual ejecuta una petición HTTP asíncrona, cuyo contenido está en formato JSON, la información que se envía es {Msj:"mensaje”} donde mensaje es el texto que se encontraba en el *input* al momento de presionar el botón.

```
<!DOCTYPE html><html>
<head>
<meta charset="utf-8">
<title>AJAX</title>
<script type="text/javascript">
var inte;
var obj=new Object();

//Establece que la función rqstV se ejecutará dentro de 500 milisegundos
function ini(){
    if(inte)
        clearInterval(inte);
    inte=setInterval(rqstV,500);
}

//Envía petición para conocer el valor actual del contador del PIC
function rqstV(){
    rqstCont('_rv_');
}

/*Envía petición mediante PUT para actualizar los recursos del PIC.
Cambiar el estado del LED o modificar registro del mensaje que se muestra
en la LCD*/
function rqstSend(n,str){
    //Define el conteo y cancela la ejecución de rqstV
    clearInterval(inte);
    //Caso de control del LED
    if(n==0){
        obj.LED=str;
        delete obj.Msj;
    }
    //Caso del envío de mensaje
    else {
        obj.Msj=str;
        delete obj.LED;
    }
    var json=JSON.stringify(obj);
    var xhr=new XMLHttpRequest();
    //Petición asincrona
    if(n==0)
        xhr.open("PUT","_rp_",true);
    else
        xhr.open("PUT","_rm_",true);
    //Envía información en formato JSON
    xhr.setRequestHeader('Content-type','application/json; charset=utf-8');
    //Envía {LED:"{on/off}"} o {Msj:"{Mensajes}"}
    xhr.send(json);
    xhr.onreadystatechange=function(){
        //Si la petición se ejecutó bien (código de estado 2xx)
        if(xhr.readyState==4 && xhr.status>199 && xhr.status<300){
            if(n==0 && str=="on")
                //Modifica HTML para indicar que se ha encendido el LED
                document.getElementById("LED").textContent="LED Encendido";
            else if(n==0 && str=="off")
                //Modifica HTML para indicar que se ha apagado el LED
                document.getElementById("LED").textContent="LED Apagado";
        }
    }
}
```

Figura P18-B.6 *Primera parte del documento HTML de página dinámica asíncrona.*

```
/*Envía petición mediante GET para solicitar parte del contenido de la página.
De la respuesta actualiza alguno de los elementos del documento HTML*/
function rqstCont(rsrc){
    //Define el conteo y cancela la ejecución de rqstV
    clearInterval(inte);
    var xhr=new XMLHttpRequest();
    //Petición asíncrona
    xhr.open("GET",rsrc,true);
    xhr.onreadystatechange=function(){
        //Si la petición se ejecutó bien (código de estado 200)
        if(xhr.readyState==4 && xhr.status==200) {
            /*Recursos que actualizan el contenido del div "d_cont"
            que está en el main de la página*/
            if(rsrc=="_re_" || rsrc=="_rl_" || rsrc=="_rc_"){
                var d_c=document.getElementById('d_cont');
                if(xhr.responseText!=null && d_c!=null)
                    //Actualiza elemento HTML
                    d_c.innerHTML=xhr.responseText;
            }
            /*Si se solicitó valor del contador, actualiza el
            párrafo cntr que contiene el valor del contador en la página*/
            else if(rsrc=="_rv_"){
                var c=document.getElementById('cntr');
                if(xhr.responseText!=null && c!=null){
                    //Actualiza valor del contador
                    c.innerHTML=xhr.responseText;
                    //Inicia a temporizar para ejecutar rsqtV
                    ini();
                }
            }
            //Si se solicitó el contenido que muestra el valor del contador
            if(rsrc=="_rc_")
                //Inicia a temporizar para ejecutar rsqtV
                ini();
        }
    }
    xhr.send(null);
}
</script>

<style type="text/css">
/*Define estilo de los elementos del documento HTML*/
body{background-color:#EBEBEB;margin: 0;height: 100vh;}
header{background-color:#811832; text-align: center; min-height: 25vh;}
nav{background-color:black;text-align: left;}
main{background-color:white;font-size:22pt;text-align: center;min-height: 50vh;}
footer{background-color:#3F3F3F;text-align: center;}
h1{font-family:Georgia;font-size:60pt;color:white;}
h2{font-size:26pt;color:white;}
/*Define estilo de las clases*/
.c1{width:96%;margin-left:2%;overflow:hidden;}
.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:white;background-color:black;}
.b_1:hover{color:black;background-color:white;}
.b_2{padding:6px;font-family:Georgia;font-size:20pt;color:black;background-color:#fff57a;}
.b_2:hover{color:#fff57a;background-color:black;}
.t1 {table-layout:fixed;width:100%;border:0;}
</style>

</head>
```

Figura P18-B.7 *Segunda parte del documento HTML de página dinámica asíncrona*

```
<body>

<header class="c1">
<table class="t1"><tr>
<td><h1 onclick="location='_r0_'">Biblioteca ESP</h1></td>
</tr></table>
</header>

<nav class="c1">
<table><tr>
<!--Al presionar alguno de los botones ejecuta la función rqstCont
para solicitar al servidor el respectivo contenido del main (actualiza div "d_cont")-->
<td><button class="b_1" onclick="rqstCont('_rl_');">LED</button></td>
<td><button class="b_1" onclick="rqstCont('_rc_');">Contador</button></td>
<td><button class="b_1" onclick="rqstCont('_re_');">Envío</button></td>
</tr></table>
</nav>

<main class="c1"><div id="d_cont">
<br><br>
<p>Diríjase al control del LED, al Contador o al envío de mensajes.</p>
<p id="cntr"></p>
</div></main>

<footer class="c1">
<h2>Febrero 2020</h2>
</footer>

</body>
</html>


<!--Se muestra el contenido del main (div "d_cont") que el PIC-ESP envia para las tres opcion
es-->
<!--Para la opción del control del LED-->
<!--
    <br><br><br>
    <button class="b_2" onclick="rqstSend(0, 'on');">Encender</button>
    <button class="b_2" onclick="rqstSend(0, 'off');">Apagar</button><br>
    <p id="LED">LED Encendido/Apagado</p>
-->

<!--Para la opción de visualizar el valor del contador-->
<!--
    <br><br>
    <p>Contador:</p>
    <p id="cntr"></p>
-->

<!--Para la opción de envío de mensajes al PIC-->
<!--
    <br>
    <p>Escriba lo que desea enviar:</p>
    <input type="text" ID="str"><br><br>
    <button class="b_2" onclick="rqstSend(1,str.value);">Enviar</button>
-->
```

Figura P18-B.8 *Tercera parte del documento HTML de página dinámica asíncrona*

Código

El código que deberá ejecutar el microcontrolador PIC para la primera actividad es el mostrado en la figura P18-B.17, figura P18-B.18 y figura P18-B.19. Mientras que para la segunda actividad se ejecuta el código mostrado en la figura P18-B.20, figura P18-B.21 y figura P18-B.22. En ambas actividades se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P18-B.23.

Resultados

Para la primera actividad, en la LCD se mostrará la dirección IP, con la que se accede desde un navegador al recurso _ri_. Se mostrará la página de inicio, la que ofrece dos opciones, una es el control del LED y la otra el visualizar el valor del contador (ver figura P18-B.9).

Al presionar la opción del control del LED, se mostrará la página del recurso _rl_, por lo que se carga completamente el contenido de la página (ver figura P18-B.10). En esta opción hay dos botones uno para encender y otro para apagar el LED. Al presionar alguno de éstos se encenderá o apagará el LED y se accederá a la página del recurso _re_ o _ra_ respectivamente.

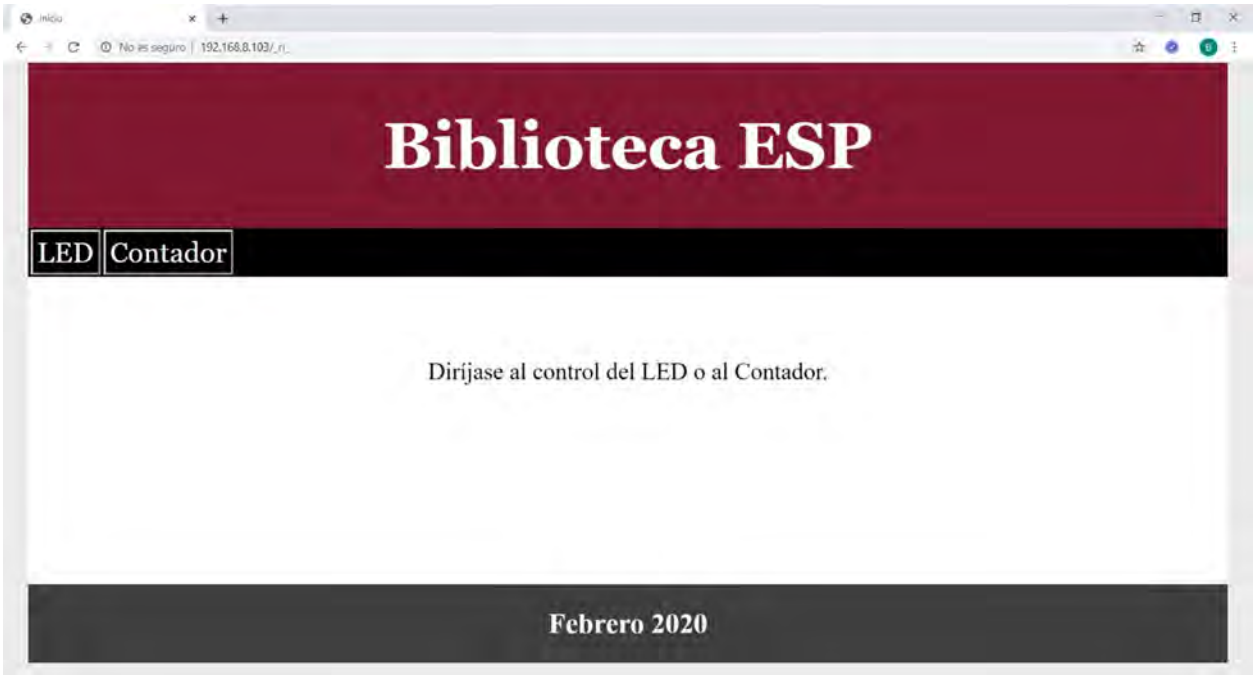


Figura P18-B.9 *Página de inicio para la actividad de páginas dinámicas síncronas.*

Figura P18-B.10 *Se carga una nueva página al encender el LED.*

Si accede a la opción de contador se mostrará el valor de éste. La página se recargará cada 5 segundos, durante ese momento no se puede utilizar la página hasta que haya cargado por completo, si se accede a herramientas de desarrolladores del navegador, en la opción de red (*network*), se observa el tiempo en que tarda ejecutarse la petición que solicita el documento HTML, además se muestra el tamaño de dicho archivo (ver figura P18-B.11). También se observará que durante la recarga se solicita al servidor el *favicon* (icono de la pestaña de la página).

Para la segunda actividad se accede al recurso _r0_ usando la dirección IP que se muestra en la LCD, al hacerlo se mostrará una página con tres opciones, una para controlar el LED, otra para ver el valor del contador y otra para enviar mensajes (ver figura P18-B.13). Al acceder a alguna de estas opciones se notará que sólo se actualiza la sección del *main* de la página y en ningún momento se bloquea la página.

Al entrar a la opción de control del LED, se actualizará el contenido de la página, aparecerán dos botones para cambiar de estado dicho elemento. Al presionar alguno de éstos se encenderá o apagará el LED y cambiará el texto asociado, tampoco se bloqueará la pantalla ni

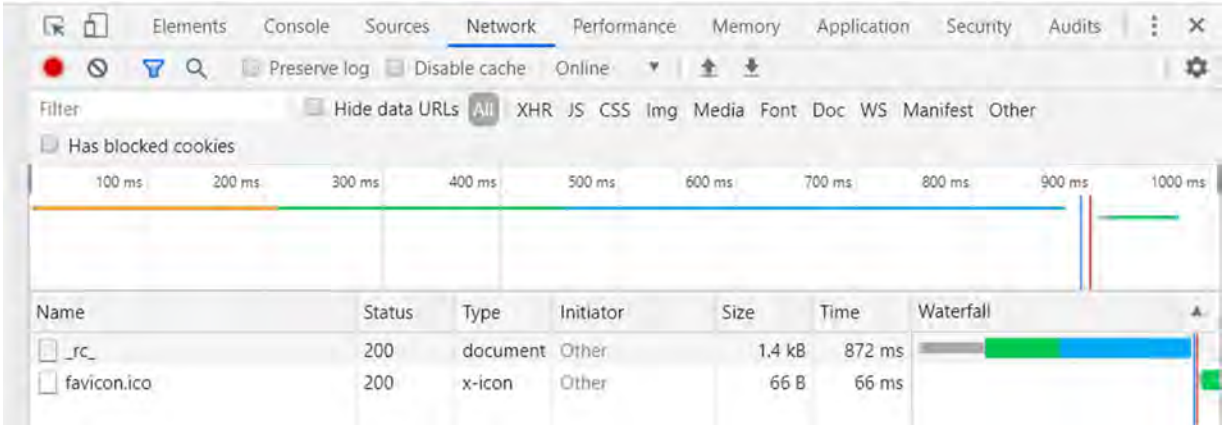


Figura P18-B.11 *Tamaño de la página de contador y tiempo de recarga para mostrar un nuevo valor.*

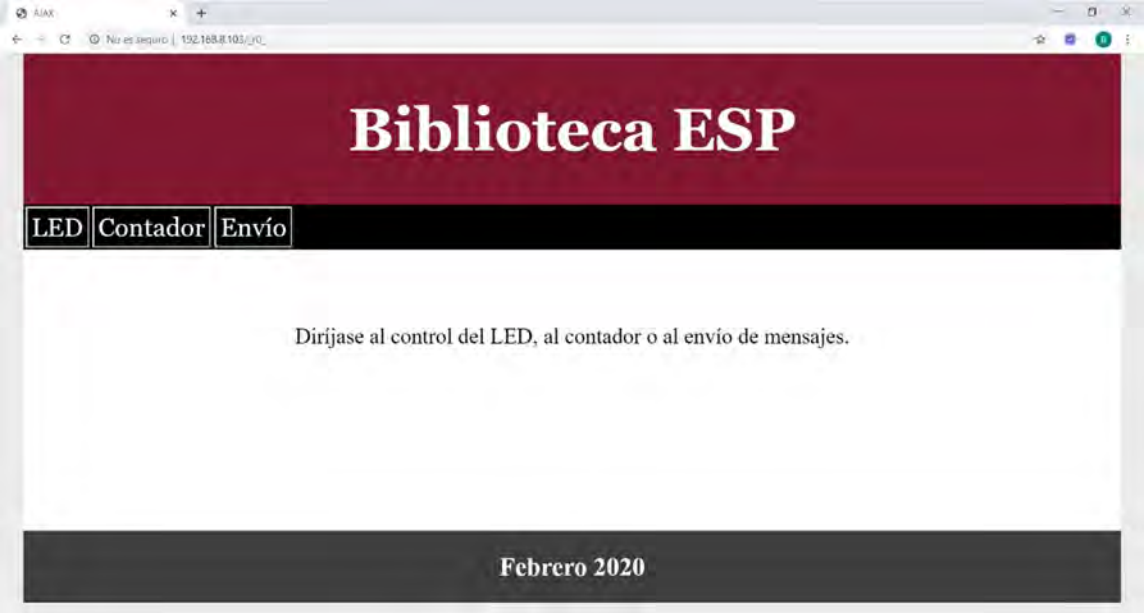


Figura P18-B.13 *Página principal para la actividad de páginas dinámicas con AJAX (asíncrona).*

direccionará a otro recurso (ver figura P18-B.12). Si accede a la herramienta de desarrolladores del navegador en la opción de red, se mostrará que, al encender y apagar el LED, se ejecuta una petición PUT que si se realiza correctamente obtiene el código de estado 204 (ver figura P18-B.15).

Figura P18-B.12 *Página encender LED AJAX*

En la opción del contador se notará que la actualización del valor se logra de manera más fluida que en el síncrona (ya que no se bloquea la página, sólo se actualiza el texto), además de que en el *script* se solicitó un tiempo muestreo menor al establecido en recarga de la síncrona. Finalmente, en la opción de envío de mensajes, cada mensaje que se envíe se mostrará en la LCD.

Al observar el tiempo en que tarda en ejecutarse las peticiones asíncronas, se notará que generalmente es menor que el tiempo correspondiente a las síncronas, esto debido a que los contenidos de las respuestas son de menor tamaño al no contener los documentos HTML completos, sino sólo fragmentos de éstos. Además, que las peticiones que controlan el LED y envían

mensajes, responden sin contenido (ver figura P18-B.16). También se puede observar que para este tipo de peticiones se envía al servidor (PIC-ESP) mensajes en formato JSON (ver figura P18-B.14).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y los códigos de la práctica:

https://bit.ly/2YgjnKF

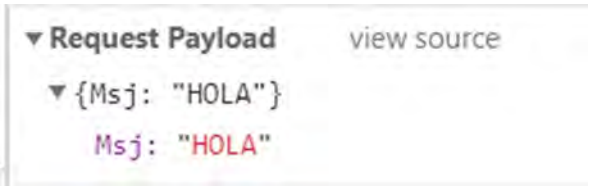


Figura P18-B.14 *Mensajes que se envían al PIC en formato JSON.*

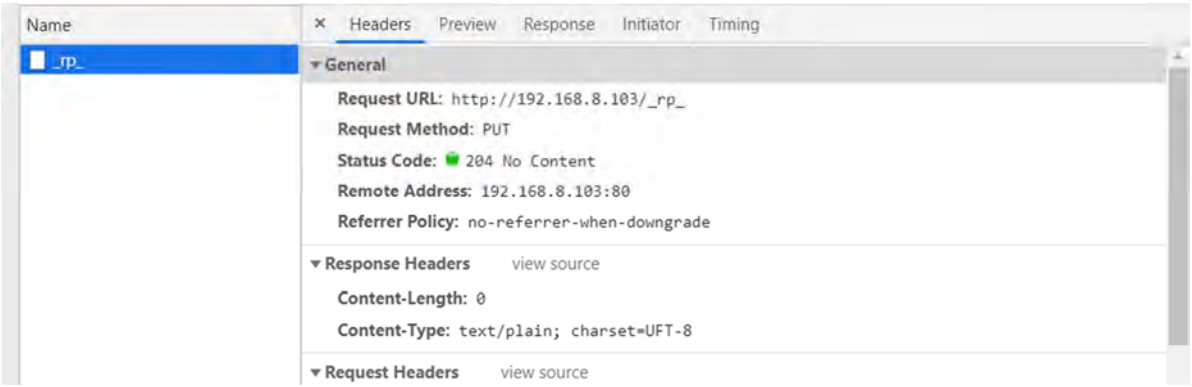


Figura P18-B.15 E*nvío de información utilizando el método PUT al recurso _rp_ en control del LED.*

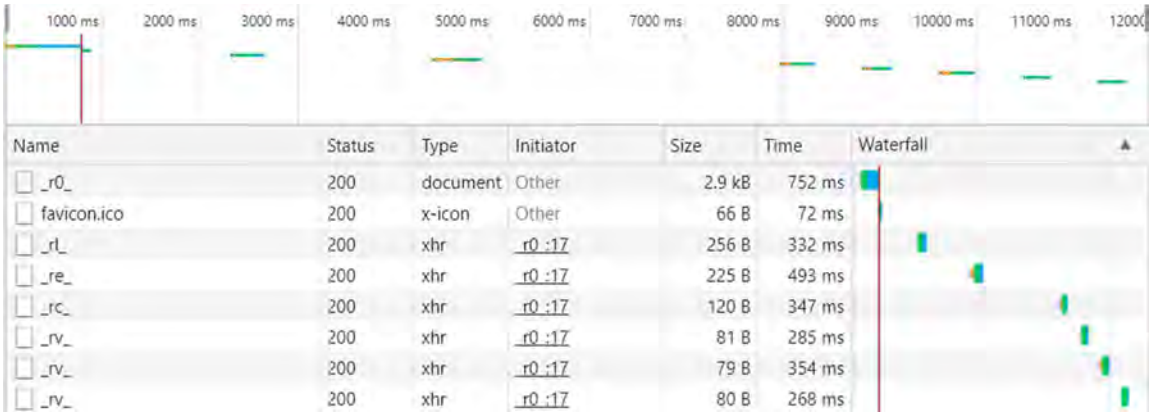


Figura P18-B.16 *Tamaño y tiempo del contenido de las peticiones asíncronas que ejecuta el navegador para las diferentes opciones de la página.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

unsigned int contador=0;
char str_contador[4];
char IP[21]={0};
volatile int recurso=0;
char str_content_length[]="0000";
const char respuesta[]=
"HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=UFT-8\r\nContent-Length: ";

const char meta[]="<!DOCTYPE html><html><head><meta charset=\"utf-8\">";
const char meta_refresh[]="<meta http-equiv=\"refresh\" content=\"5\">";

const char title_i[]="<title>Inicio</title>";
const char title_c[]="<title>Contador</title>";
const char title_l[]="<title>LED</title>";
const char style[]=
"<style type=\"text/css\">body{background-color:#EBEBEB;margin: 0;height: 100vh;}header{backg
round-color:#811832; text-align: center; min-height: 25vh;}nav{background-color:black;text-al
ign: left;}main{background-color:white;font-size:22pt;text-align: center;min-height: 50vh;}fo
oter{background-color:#3F3F3F;text-align: center;}h1{font-family:Georgia;font-size:60pt;colo
r:white;}h2{font-size:26pt;color:white;}.c1{width:96
%
;margin-left:2%;overflow:hidden;}.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:wh
ite;background-color:black;}.b_1:hover{color:black;background-color:white;}.b_2{padding:6px;f
ont-family:Georgia;font-size:20pt;color:black;background-color:#fff57a;}.b_2:hover{color:#fff
57a;background-color:black;}.t1 {table-layout:fixed;width:100
%;border:0;}</style>";
const char header_nav[]=
"<header class=\"c1\"><table class=\"t1\"><tr><td><h1 onclick=\"location='_ri_'\">Biblioteca
 ESP</h1></td></tr></table></header><nav class=\"c1\"><table><tr><td><button class=\"b_1\" on
click=\"location='_rl_'\">LED</button></td><td><button class=\"b_1\" onclick=\"location='_rc
_'\">Contador</button></td></tr></table></nav>"
;
const char main_i[]=
"<main class=\"c1\"><br><br><p>Diríjase al control del LED o al Contador.</p></main>";
const char main_c1[]="<main class=\"c1\"><br><br><p>Contador:</p><p>";
const char main_c2[]="</p></main>";
const char main_l[]=
"<main class=\"c1\"><br><br><button class=\"b_2\" onclick=\"location='_re_'\">Encender</butto
n><button class=\"b_2\" onclick=\"location='_ra_'\">Apagar</button><br><br><p>LED "
;
const char encendido[]="Encendido</p></main>";
const char apagado[]="Apagado</p></main>";
const char footer[]="<footer class=\"c1\"><h2>Febrero 2020</h2></footer></body></html>";

//Cálculo de longitud de páginas que otorga el servidor
long ri_content_length=sizeof(meta)-1+sizeof(title_i)-1+sizeof(style)-1+sizeof(header_nav)-1
+sizeof(main_i)-1+sizeof(footer)-1;
long rl_content_length=sizeof(meta)-1+sizeof(title_l)-1+sizeof(style)-1+sizeof(header_nav)-1
+sizeof(main_l)-1+sizeof(footer)-1;
long re_content_length=sizeof(meta)-1+sizeof(title_l)-1+sizeof(style)-1+sizeof(header_nav)-1
+sizeof(main_l)-1+sizeof(footer)-1+sizeof(encendido)-1;
long ra_content_length=sizeof(meta)-1+sizeof(title_l)-1+sizeof(style)-1+sizeof(header_nav)-1
+sizeof(main_l)-1+sizeof(footer)-1+sizeof(apagado)-1;
long rc_content_length=sizeof(meta)-1+sizeof(meta_refresh)-1+sizeof(title_c)-1+sizeof(style)-
1+sizeof(header_nav)-1+sizeof(main_c1)-1+sizeof(main_c2)-1+sizeof(footer)-1;
```

Figura P18-B.17 *Primera parte del código de la actividad de páginas síncronas.*

```
void main()
{
   set_tris_b(0b01111111);
   output_b(0);
   lcd_init();
   lcd_gotoxy(2,1);
   lcd_putc("Direccion IP:");
   //Inicialización del módulo Wi-Fi
   begin_esp();
   //Conexión al punto de acceso indicado
   begin_station("Estacion",SSID,password);
   //Obtiene dirección IP local
   get_ip_station(IP,sizeof(IP));
   lcd_gotoxy(2,2);
   printf(lcd_putc,"%s",IP);
   /*Crea el servidor web para el puerto 80 (por defecto para navegadores),
   permite cuatro conexiones simultáneas con un timeout de 60s y asigna
   la variable que apunta al recurso del primer elemento de cola de atención
   de peticiones HTTP*/
   create_http_server(80,4,60,&recurso);
   while(1)
   {
      //Actualiza el recurso solicitado para el primer elemento de la cola
      http_upgrade_resource();
      switch (recurso)
      {
         //Responde con la página de inicio (_ri_)
         case 'i':
         http_send_const(respuesta);
         sprintf(str_content_length,"%lu",ri_content_length);
         http_send(str_content_length);
         http_send("\r\n\r\n");
         http_send_const(meta);
         http_send_const(title_i);
         http_send_const(style);
         http_send_const(header_nav);
         http_send_const(main_i);
         http_send_const(footer);
         http_finish(1);
         break;

         //Responde con la página que control del LED (_rl_)
         case 'l':
         http_send_const(respuesta);
         if(input_state(PIN_B7))
         {
            sprintf(str_content_length,"%lu",rl_content_length+sizeof(encendido)-1);
         }
         else
            sprintf(str_content_length,"%lu",rl_content_length+sizeof(apagado)-1);
         http_send(str_content_length);
         http_send("\r\n\r\n");
         http_send_const(meta);
         http_send_const(title_l);
         http_send_const(style);
         http_send_const(header_nav);
         http_send_const(main_l);
         if(input_state(PIN_B7))
         {
            http_send_const(encendido);
         }
         else
            http_send_const(apagado);
         http_send_const(footer);
         http_finish(1);
         break;
```

Figura P18-B.18 *Segunda parte del código de la actividad de páginas síncronas.*

```
      //Responde con la página de LED encendido (_re_)
       case 'e':
       output_bit(PIN_B7,1);
       http_send_const(respuesta);
       sprintf(str_content_length,"%lu",re_content_length);
       http_send(str_content_length);
       http_send("\r\n\r\n");
       http_send_const(meta);
       http_send_const(title_1);
       http_send_const(style);
       http_send_const(header_nav);
       http_send_const(main_1);
       http_send_const(encendido);
       http_send_const(footer);
       http_finish(1);
       break;

       //Responde con la página de LED apagado (_ra_)
       case 'a':
       output_bit(PIN_B7,0);
       http_send_const(respuesta);
       sprintf(str_content_length,"%lu",ra_content_length);
       http_send(str_content_length);
       http_send("\r\n\r\n");
       http_send_const(meta);
       http_send_const(title_1);
       http_send_const(style);
       http_send_const(header_nav);
       http_send_const(main_1);
       http_send_const(apagado);
       http_send_const(footer);
       http_finish(1);
       break;

       //Envía respuesta para el recurso (página) _rc_
       case 'c':
       sprintf(str_contador,"%u",contador);
       sprintf(str_content_length,"%lu",rc_content_length+get_str_length(str_contador));
       http_send_const(respuesta);
       http_send(str_content_length);
       http_send("\r\n\r\n");
       http_send_const(meta);
       http_send_const(meta_refresh);
       http_send_const(title_c);
       http_send_const(style);
       http_send_const(header_nav);
       http_send_const(main_c1);
       http_send(str_contador);
       http_send_const(main_c2);
       http_send_const(footer);
       http_finish(1);
       break;

       //Atiende elementos inválidos
       default:
       http_default_resource();
       break;
    }
    contador++;
    delay_ms(50);
  }

}
```

Figura P18-B.19 *Tercera parte del código de la actividad de páginas síncronas.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>
/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char IP[21]={0};
char json[15]={0};
volatile int recurso=0;
char mensaje[10];
char str_contador[4];
unsigned int contador=0;
char str_content_length[]="0000";
const char respuesta[]=
"HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=UFT-8\r\nContent-Length: ";
const char pag_princ0[]=
"<!DOCTYPE html><html><head><meta charset=\"utf-8\"/><title>AJAX</title><script type=\"text/j
avascript\">
var inte;var obj=new Object(); function ini(){ if(inte) clearInterval(inte);inte=setInterval
(rqstV,500);} function rqstV(){ rqstCont('_rv_');} function rqstSend(n,str){ clearInterval(in
te); if(n==0){ obj.LED=str;delete obj.Msj;} else { obj.Msj=str;delete obj.LED;} var json=JSO
N.stringify(obj);var xhr=new XMLHttpRequest();if(n==0) xhr.open(\"PUT\",\"_rp_\",true); else
 xhr.open(\"PUT\",\"_rm_\",true);
xhr.setRequestHeader('Content-type','application/json; charset=utf-8');xhr.send(json); xhr.on
readystatechange=function(){ if(xhr.readyState==4 && xhr.status>199 && xhr.status<300){
if(n==0 && str==\"on\") document.getElementById(\"LED\").textContent=\"LED Encendido\";
else if(n==0 && str==\"off\") document.getElementById(\"LED\").textContent=\"LED Apagado\";
}
}
}
function rqstCont(rsrc){ clearInterval(inte);var xhr=new XMLHttpRequest();xhr.open(\"GET\",rs
rc,true); xhr.onreadystatechange=function(){ if(xhr.readyState==4 && xhr.status==200) {
if(rsrc==\"_re_\" || rsrc==\"_rl_\" || rsrc==\"_rc_\"){ var d_c=document.getElementById('d_co
nt'); if(xhr.responseText!=null && d_c!=null) d_c.innerHTML=xhr.responseText;}
else if(rsrc==\"_rv_\"){ var c=document.getElementById('cntr'); if(xhr.responseText!=null &&
 c!=null){ c.innerHTML=xhr.responseText; ini();
}
}
if(rsrc==\"_rc_\") ini();
}
}
xhr.send(null);}
</script>";
const int pag_princ1[]=
"<style type=\"text/css\">body{background-color:#EBEBEB;margin: 0;height: 100vh;}header{backg
round-color:#811832; text-align: center; min-height: 25vh;}nav{background-color:black;text-al
ign: left;}main{background-color:white;font-size:22pt;text-align: center;min-height: 50vh;}fo
oter{background-color:#3F3F3F;text-align: center;}h1{font-family:Georgia;font-size:60pt;colo
r:white;}h2{font-size:26pt;color:white;}.c1{width:96
%
;margin-left:2%;overflow:hidden;}.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:wh
ite;background-color:black;}.b_1:hover{color:black;background-color:white;}.b_2{padding:6px;f
ont-family:Georgia;font-size:20pt;color:black;background-color:#fff57a;}.b_2:hover{color:#fff
57a;background-color:black;}.t1 {table-layout:fixed;width:100
%
;border:0;}</style><body><header class=\"c1\"><table class=\"t1\"><tr><td><h1 onclick=\"locat
ion='_r0_'\">Biblioteca ESP</h1></td></tr></table></header><nav class=\"c1\"><table><tr><td><
button class=\"b_1\" onclick=\"rqstCont('_rl_');\">LED</button></td><td><button class=\"b_1\"
 onclick=\"rqstCont('_rc_');\">Contador</button></td><td><button class=\"b_1\" onclick=\"rqst
Cont('_re_');\">Envío</button></td></tr></table></nav><main class=\"c1\"><div id=\"d_cont\"><
br><br><p>Diríjase al control del LED, al contador o al envío de mensajes.</p><p id=\"cntr\">
</p></div></main><footer class=\"c1\"><h2>Febrero 2020</h2></footer></body></html>"
;
```

Figura P18-B.20 *Primera parte del código de la actividad de página asíncronas.*

```
const char cont_led[]=
"<br><br><br><button class=\"b_2\" onclick=\"rqstSend(0, 'on');\">Encender</button><button cl
ass=\"b_2\" onclick=\"rqstSend(0, 'off');\">Apagar</button><br>"
;
const char led_ence[]="<p id=\"LED\">LED Encendido</p>";
const char led_apag[]="<p id=\"LED\">LED Apagado</p>";
const char cont_envio[]=
"<br><p>Escriba lo que desea enviar:</p><input type=\"text\" ID=\"str\"><br><br><button class
=\"b_2\" onclick=\"rqstSend(1,str.value);\">Enviar</button>"
;
const char cont_contador[]="<br><br><p>Contador:</p><p id=\"cntr\"></p>";
void main()
{
    set_tris_b(0b01111111);
    output_b(0);
    lcd_init();
    lcd_gotoxy(2,1);
    lcd_putc("Direccion IP:");
    begin_esp();
    begin_station("Estacion",SSID,password);
    get_ip_station(IP,sizeof(IP));
    lcd_gotoxy(2,2);
    printf(lcd_putc,"%s",IP);
    //Establece buffer en que se alojan mensajes en formato JSON provenientes de los clientes
    set_http_sjsonbuffer(json,sizeof(json));
    /*Crea el servidor web para el puerto 80 (por defecto para navegadores), permite
    cuatro conexiones simultáneas con un timeout de 60s y asignala variable que apunta
    al recurso del primer elemento de cola de atención de peticiones HTTP*/
    create_http_server(80,4,60,&recurso);
    while(1)
    {
        //Actualiza el recurso solicitado para el primer elemento de la cola
        http_upgrade_resource();
        switch (recurso)
        {
            //Responde con la página principal (_r0_)
            case '0':
            http_send_const(respuesta);
            sprintf(str_content_length,"%lu",sizeof(pag_princ0)+sizeof(pag_princ1)-2);
            http_send(str_content_length);
            http_send("\r\n\r\n");
            http_send_const(pag_princ0);
            http_send_const(pag_princ1);
            http_finish(1);
            break;

            //Responde con la sección de página de control del LED (_r1_)
            case '1':
            http_send_const(respuesta);
            if(input_state(PIN_B7))
            {
                sprintf(str_content_length,"%lu",sizeof(cont_led)+sizeof(led_ence)-2);
            }
            else
                sprintf(str_content_length,"%lu",sizeof(cont_led)+sizeof(led_apag)-2);
            http_send(str_content_length);
            http_send("\r\n\r\n");
            http_send_const(cont_led);
            if(input_state(PIN_B7))
            {
                http_send_const(led_ence);
            }
            else
                http_send_const(led_apag);
            http_finish(0);
            break;
```

Figura P18-B.21 *Segunda parte del código de la actividad de página asíncronas.*

```
            //Recurso encender y apagar LED (_rp_), responde "204 No Content"
            case 'p':
            //Revisa que en el buffer JSON exista {LED:"on"}
            unsigned int i=compare_http_sjson("LED","on",0);
            if(i==1)
            {
                output_bit(PIN_B7,1);
            }
            else
                output_bit(PIN_B7,0);
            http_response_put();
            break;

            //Responde con la sección de página del contador (_rc_)
            case 'c':
            http_send_const(respuesta);
            sprintf(str_content_length,"%lu",sizeof(cont_contador)-1);
            http_send(str_content_length);
            http_send("\r\n\r\n");
            http_send_const(cont_contador);
            http_finish(0);
            break;

            //Responde con la sección de página de envio de mensajes(_re_)
            case 'e':
            http_send_const(respuesta);
            sprintf(str_content_length,"%lu",sizeof(cont_envio)-1);
            http_send(str_content_length);
            http_send("\r\n\r\n");
            http_send_const(cont_envio);
            http_finish(0);
            break;

            //Responde con la sección de página que tiene el valor del contador (_rv_)
            case 'v':
            sprintf(str_contador,"%u",contador);
            http_send_const(respuesta);
            sprintf(str_content_length,"%lu",get_str_length(str_contador));
            http_send(str_content_length);
            http_send("\r\n\r\n");
            http_send(str_contador);
            http_finish(0);
            break;

            /*Recurso del registro de mensaje  (_rm_), responde "204 No Content"*/
            case 'm':
            //Obtiene el valor la clave Msj en el buffer JSON
            get_value_http_sjson("Msj",mensaje,sizeof(mensaje));
            printf(lcd_putc,"\f");
            lcd_gotoxy(5,1);
            printf(lcd_putc,"Mensaje:");
            lcd_gotoxy(1,2);
            printf(lcd_putc,"%s",mensaje);
            http_response_put();
            break;

            default:
            http_default_resource();
            break;
        }
        contador++;
      delay_ms(50);
    }
}
```

Figura P18-B.22 *Tercera parte del código de la actividad de página asíncronas.*

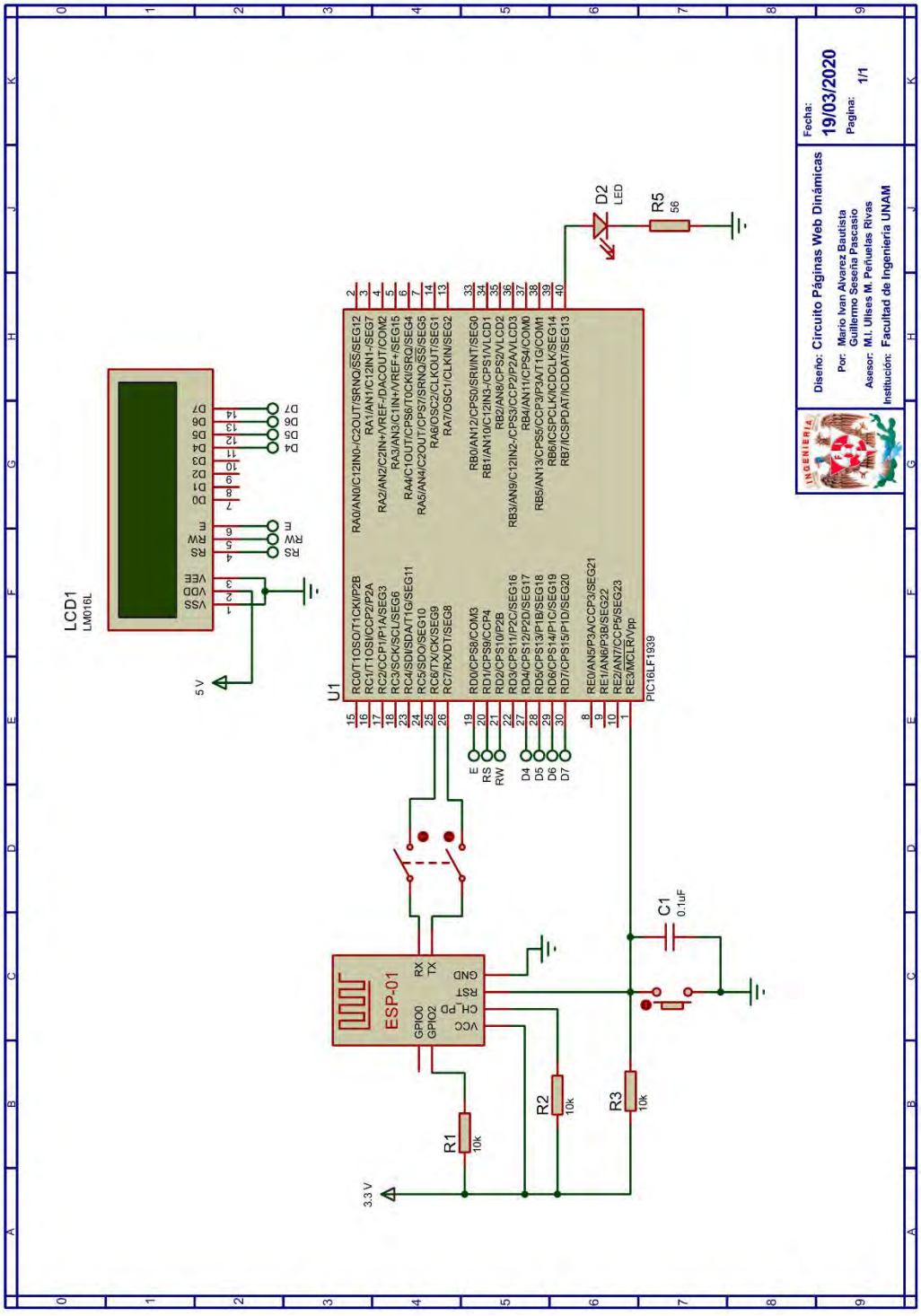


Figura P18-B.23 *Circuito de la práctica 18-B.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.
2. Deitel, P., H. Deitel, and D. Abbey, *Internet & World Wide Web Cómo programar*. Quinta ed. 2014.

Práctica 18-C Servidor Web: envío de información mediante el URL

Introducción

Como se ha mostrado en prácticas anteriores, existen distintas formas de comunicarse con el módulo ESP y en consecuencia de controlar al microcontrolador PIC, sin embargo, todas requieren de algún software que ofrezca alguna de las opciones de comunicación del módulo. De todos estos programas, lo más fáciles de acceder son los navegadores web, que están presentes en la mayoría de las computadoras y teléfonos inteligentes.

Utilizar un navegador web implica que la información es transmitida a través del protocolo HTTP [1]. En prácticas previas, se mostró que el módulo puede proveer páginas web para controlar algún recurso del PIC desde un navegador, no obstante, implica el diseño de una página web.

Una alternativa para enviar información utilizando un navegador sin diseñar una página es enviar mensajes a través del URL, en vez de colocar el directorio y nombre de una página se coloca el mensaje que se desee enviar. Lo anterior es posible ya que, al establecer al módulo como un servidor web, la biblioteca ESP puede almacenar en un *buffer* el directorio y nombre de los recursos que se soliciten mediante los métodos GET y PUT, sin que necesariamente los recursos solicitados sean válidos para la biblioteca o para el programa del microcontrolador, por consiguiente, los directorios y nombre de recursos pueden ser utilizados como mensajes para desencadenar alguna acción en el PIC.

Al enviar un mensaje utilizando el URL, el navegador obtendrá el error 404, que indica que la página (recurso) solicitada no ha sido encontrada, ya que es la respuesta por defecto que la biblioteca otorga a los clientes cuando solicitan recursos considerados inválidos.

Es importante señalar que esta forma de enviar información no hace un uso propio de lo que el protocolo HTTP establece, ya que el método que ejecuta el navegador al solicitar una página es GET, que su función es la recuperar la representación de un recurso y no el envío de información, además de que la URL sirve para localizar recursos y no como medio para enviar información. Sin embargo, esta es una alternativa de fácil acceso para usuarios que deseen controlar al PIC sin *software* adicional.

La estructura del URL para enviar mensajes al módulo es la siguiente:

IP del módulo:[puerto utilizado por el servidor web]/[mensaje a enviar]

El envío de mensajes se puede realizar de manera remota, es decir desde un

dispositivo conectado a una red diferente a la del módulo, pero se requiere de un mapeo de puertos en la red del módulo, así como se realizó en la segunda actividad de la práctica de servidor TCP. Para ese caso, la dirección IP del módulo y puerto utilizado por el servidor web son sustituidos por la IP pública de la red del módulo y el puerto que se asignó en la configuración del mapeo.

Es relevante tener en cuenta que algunos caracteres no pueden ser incluidos en el URL de manera directa y, por lo tanto, no pueden ser enviados como mensajes. Los caracteres que no se pueden incluir en URL son comúnmente sustituidos por su representación hexadecimal de acuerdo con ASCII antecedido por un "%". Por ejemplo, si se envía el mensaje "<hola mundo>", se recibiría como "%3Chola%20mundo%3E".

Para este tipo de mensajes, la biblioteca ESP solo puede almacenarlos y compararlos con un *string*, así que el usuario le puede dar cualquier formato para utilizarlos, por ejemplo, podría implementar cadenas de consulta (*Query string*) para enviar en un solo mensaje distintos datos.

Objetivo

Enviar información al módulo a través del URL utilizando un navegador web.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_WEB_SERVER

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 pantalla LCD 16x2
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 56 Ω

Descripción

Para esta práctica se enviarán mensajes al módulo desde un servidor web a través del URL. Al enviar el mensaje *on* se deberá encender un LED y para apagarlo

se enviará *off*. Todos los mensajes que se envían al PIC-ESP son mostrados en una LCD.

Código

El código que deberá ejecutar para esta práctica se muestra en la figura P18-C.2, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P18-C.3.

Resultados

Al iniciar la ejecución del programa del PIC, lo primero que se observará en la LCD es la dirección IP que el AP le asignó al módulo. Esta dirección se colocará como URL en el navegador junto con el mensaje *on*, al oprimir la tecla *enter* se encenderá el LED y se mostrará en el navegador el error 401 (ver figura P18-C.1). Al enviar el mensaje *off* se apagará el LED. Todos los mensajes que se envíen al módulo se mostrarán en la LCD.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de esta práctica:

https://bit.ly/2BkLiju

Desarrollo de herramientas de comunicación

Ya que la biblioteca ESP ha implementado parcialmente el protocolo HTTP1.1 es más fácil involucrar al microcontrolador PIC en aplicaciones web, esto a su vez facilita el desarrollo de herramientas que se comuniquen con el microcontrolador PIC, ya que al usar un protocolo tan popular no es necesario idear desde cero las reglas para intercambiar información, caso que se presenta si se utiliza directamente sockets TCP o UDP.

Una opción que permite diseñar herramientas es MIT App inventor, el cual es un entorno de programación visual e intuitivo de tipo *drag-drop* desarrollado por Google para la creación de aplicaciones Android, el cual está caracterizado por permitir un rápido desarrollo gracias a que se basa en un modelo de programación en bloques. Como App Inventor es una herramienta basada en la nube, las aplicaciones se desarrollan desde un navegador web ingresando al siguiente enlace:

ai2.appinventor.mit.edu.

Para crear aplicaciones en este entorno que puedan interactuar con un módulo WiFi mediante HTTP, se tienen dos opciones. La primera opción es hacer uso del componente *webviewer*, el cual permite ver páginas web especificando la URL de una página (utiliza el método

GET ya que funciona como un navegador web). La segunda opción consiste hacer uso del componente no visible *web*, cual habilita el uso de los métodos HTTP GET, POST, PUT y DELETE para enviar y/o recibir datos a un servidor.

Un ejemplo una aplicación que se desarrolló y creada en este entorno puede ser descargado en el siguiente enlace:

https://bit.ly/2BsLtJt

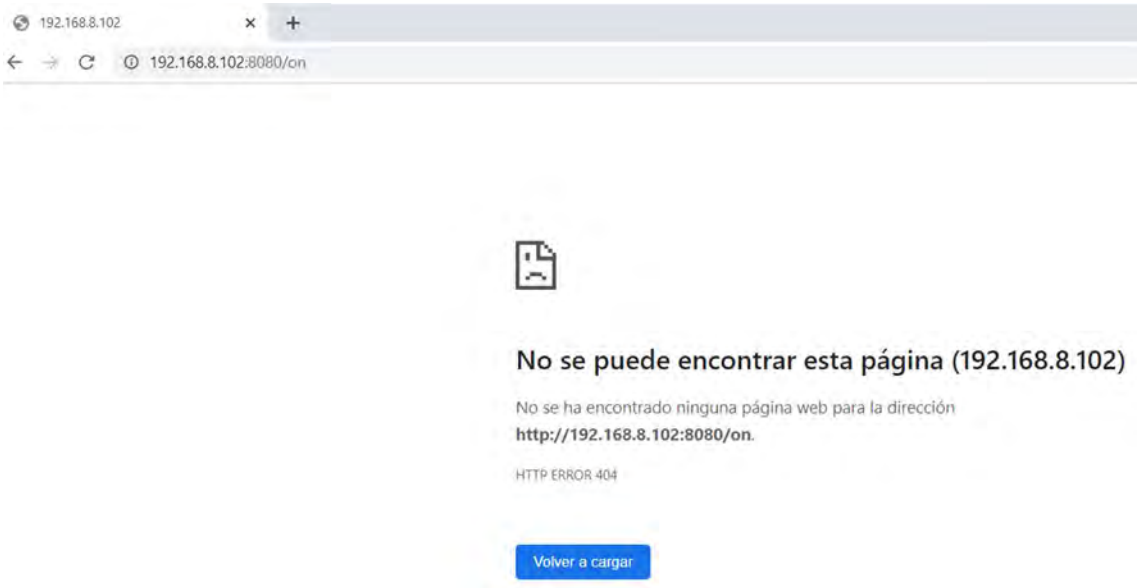


Figura P18-C.1 *Vista del navegador al enviar un mensaje en el URL.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char IP[21]={0};
char buffer[21]={0};
int recurso=0;

void main()
{
    set_tris_b(0b01111111);
    output_b(0);
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Obtiene dirección IP del módulo
    get_ip_station(IP,sizeof(IP));
    lcd_gotoxy(2,1);
    lcd_putc("Direccion IP:");
    lcd_gotoxy(2,2);
    printf(lcd_putc,"%s",IP);
    //Establece buffer en que se alojan mensajes enviados dentro de la URL
    set_buffer_httpserver(buffer,sizeof(buffer));
    //Crea el servidor web para el puerto 8080 con un timeout de 120s
    create_http_server(8080,4,120,&recurso);
    while(1)
    {
        //Actualiza el recurso solicitado para el primer elemento de la cola
        http_upgrade_resource();
        switch (recurso)
        {
            default://Atiende elementos inválidos
            http_default_resource();
            break;
        }
        if(compare_httpSbuffer("on",1))//Enciende LED
        {
            output_bit(PIN_B7,1);
        }
        else if(compare_httpSbuffer("off",1))//Apaga LED
        {
            output_bit(PIN_B7,0);
        }
        if(buffer[0]!='\0')//Muestra en la LCD el mensaje recibido
        {
            printf(lcd_putc,"\f");
            lcd_gotoxy(5,1);
            printf(lcd_putc,"Mensaje:");
            lcd_gotoxy(1,2);
            printf(lcd_putc,"%s",buffer);
            delay_ms(500);
        }
    }
}
```

Figura P18-C.2 *Código de la práctica 18-C.*

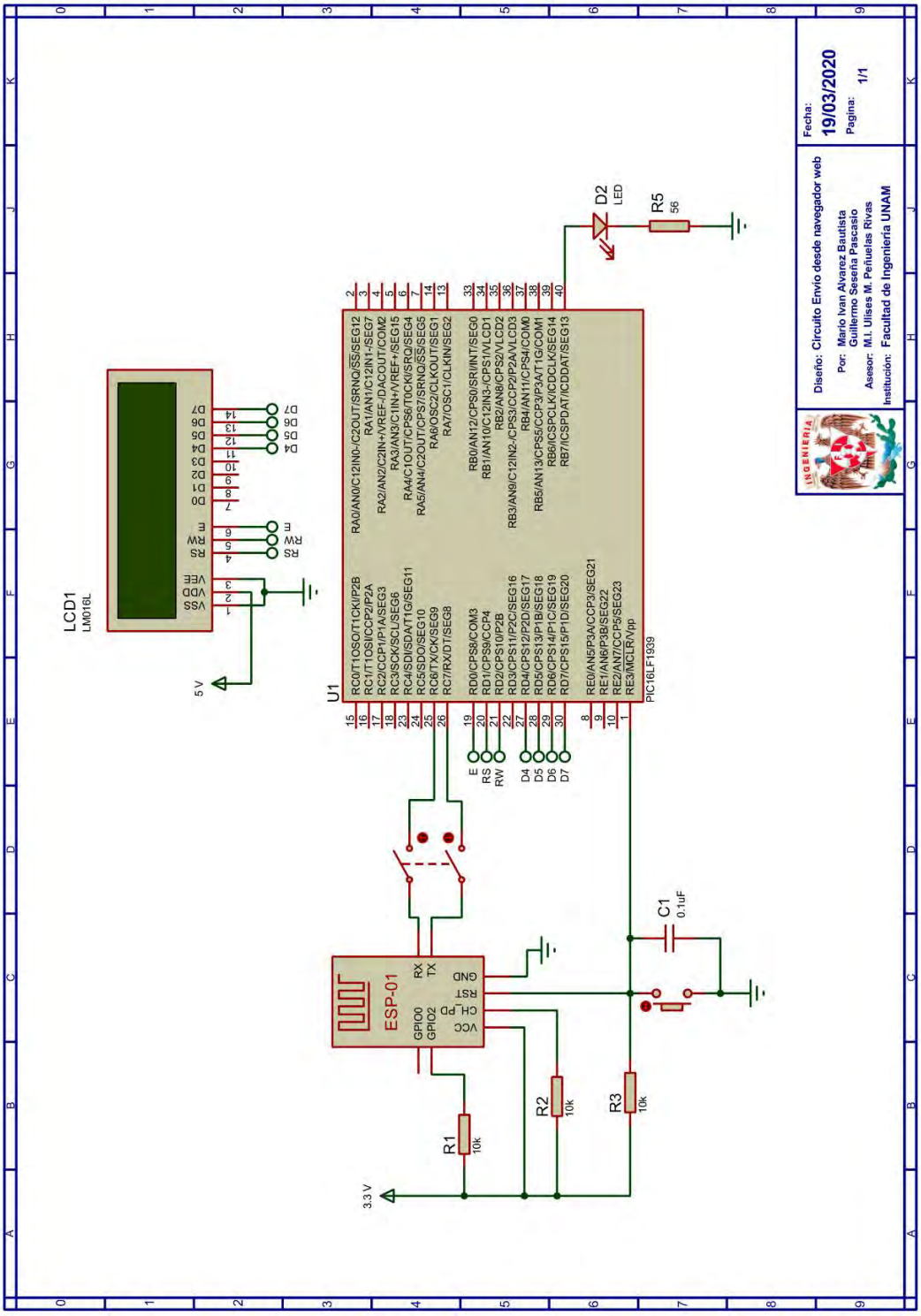


Figura P18-C.3 *Circuito de la práctica 18-C.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 19-A Cliente HTTP: consumiendo un servicio web

Introducción

Inicialmente el uso de HTTP estaba limitado a mostrar información en páginas web, pero en la actualidad su uso se ha extendido a la transferencia de información entre distintas aplicaciones sin importar que estén programadas en distintos lenguajes o sistemas operativos, esto beneficia la interoperabilidad de las aplicaciones disponibles en internet [1].

Las nuevas aplicaciones aprovechan las características de HTTP para trasmitir información de cualquier tipo. A este tipo de aplicaciones se les han denominado como servicios Web, las cuales están basados en distintas arquitecturas como Protocolo simple de acceso a objetos (*Simple Object Access Protocol,* SOAP), Lenguaje de descripción de servicios web (*Web Services Description Language,* WSDL) y Transferencia de representación de estado (*Representational State Transfer,* REST) [1, 2].

Los servicios web basados en REST son los que han gozado de mayor popularidad en los últimos años, por su facilidad de implementación en comparación con otras arquitecturas. Esta arquitectura está basada en tres principios: utilizar HTTP de manera explícita, controlar solicitudes sin estado y usar URL como directorio del recurso.

Usar HTTP de manera explícita es utilizar los métodos de tal forma que se adecúe mejor al uso que se había previsto en la especificación del protocolo. Los métodos han sido predefinidos para ejecutar cierta acción en los recursos, pero siempre dependen de cómo se programe el servidor web. Algunos servicios web utilizan los métodos de manera “incorrecta” de acuerdo a la idea original, por ejemplo, el método GET que es uno de los métodos más populares ha sido utilizado para actualizar, crear recursos, crear nuevos registros a base de datos, pero en realidad fue diseñado para realizar consultas [2].

La arquitectura REST hace uso principalmente de cuatro métodos para realizar tareas concisas:

- GET, método para consultar el contenido de un recurso.
- POST, método para crear un nuevo recurso o registro en el servidor.
- PUT, método para actualizar un recurso ya existente.
- DELETE, método para borrar un recurso.

En REST, la información que fluye a través de las peticiones debe ser transmitida en la carga útil del contenido de dichas solicitudes y no a través del

URL. Esta información utiliza un formato MIME, de los cuales destacan XML y JSON [2].

En REST se busca que las solicitudes sean completas e independientes, para que el servidor no tenga que almacenar el estado de las peticiones. También, los URL deben apuntar a un recurso en específico, indicando el directorio de dicho recurso, para lo que comúnmente se establecen jerarquías, que permite a los clientes referirse a los recursos del servidor de manera intuitiva [2].

La arquitectura REST está siendo ampliamente utilizada en el desarrollo de APIs (Interfaz de programación de aplicaciones o *Application Programming Interface*) de muchos servicios en línea. Empresas importantes como Google, Facebook, Twitter y Amazon han desarrollado APIs basadas en esta arquitectura, permitiendo a las aplicaciones hacer uso de sus servicios.

El módulo ESP es capaz de consumir servicios web basados en REST ya que la biblioteca ESP le permite funcionar como un cliente HTTP, ya sea mediante una conexión TCP o una SSL/TLS. El cliente puede construir y ejecutar peticiones HTTP libremente. Al ejecutar cada petición, la biblioteca solo indica el código de estado HTTP que ha otorgado la respuesta.

Es posible almacenar y manejar datos en formato JSON al activar la herramienta E_JSON, pero con restricciones. Las respuestas deben especificar dos encabezados:

- Content-type: debe tener el valor de application/json; charset=utf-8;
- Content-length: debe tener un valor diferente a nulo.

La biblioteca sólo puede alojar la representación en texto de pares (claves y valores) simples como *strings*, números y valores booleanos, no es posible manejar arreglos, ni matrices, ni listas, ni objetos. Los valores de cada clave almacenada pueden ser consultados y comparados como *strings*. Por ejemplo, el siguiente mensaje es válido para ser recibido:

{num: 0, msj:”mensaje”, “valor”:true}.

De este mensaje se puede extraer el *string* del valor de la clave msj, el cual es “mensaje”. Para información más compleja la biblioteca sólo ofrece captura, por lo que el usuario debe extraer “manualmente” la información útil de éste, que comúnmente estará contenido junto con más información adicional que envían los servidores.

Con lo mencionado anteriormente, es por lo que la biblioteca no proporciona una opción generalizada para facilitar el

manejo del contenido de las respuestas HTTP.

Objetivo

Establecer al módulo ESP como un cliente HTTP para consumir algún servicio web.

Firmware de comandos AT

Se puede utilizar tres de los *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-32.

Habilitación de aplicaciones

STATION y E_HTTP_CLIENT.

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 µF
- 1 pantalla LCD 16x2

Descripción

En esta práctica se utilizará la API de “Datos Abiertos Ciudad de México”, que es proporcionada por el Gobierno de la Ciudad de México. Esta API permite la consulta gratuita de los registros de la calidad del aire de los municipios y alcaldías de la Zona Metropolitana del Valle de México. La documentación de la API se encuentra en la página:

https://datos.cdmx.gob.mx/explore/dataset/prueba_datos_calidad_aire/information/

El módulo ESP realizará una petición HTTP para consultar el último registro de calidad del aire de alguna de los


```
{
    "nhits": 46,
    "parameters": {
        "dataset": "prueba_datos_calidad_aire",
        "timezone": "UTC",
        "rows": 1,
        "format": "json"
    },
    "records": [
        {
            "datasetid": "prueba_datos_calidad_aire",
            "recordid": "9a3524c352baa6974aacf771aa4874a6e99613cf",
            "fields": {
                "indice": "BUENA",
                "recomendacion_dos": "Puedes ejercitarte al aire
                libre",
                "recomendacion_tres": "Sin riesgo para grupos
                sensibles",
                "alcaldia_municipio": "ALVARO OBREGON",
                "riesgo": "Sin riesgo",
                "recomendacion_uno": "Puedes realizar actividades al
                aire libre",
                "fecha_actualizacion": "2020-05-20 15:00:16"
            },
            "record_timestamp": "2020-05-20T20:00:18.720000+00:00"
        }
    ]
}
```


Figura 19-A.1 *Contenido de la respuesta que solicita el último registro de calidad de aire.*

alcandías o municipios del ZMVM. Ya que la respuesta que proporciona la API es como la mostrada en la figura 19-A.1, se almacenará toda la respuesta en un arreglo de caracteres que se desplegará en una LCD. Para solicitar el último registro el módulo se conectará como cliente al servidor datos.cdmx.gob.mx en el puerto SSL 443 y ejecutará la petición que se muestra en la figura 19-A.3.

Después de desplegar el último registro en LCD, se enviará una petición errónea al servidor para obtener un código de error. La segunda petición que se envía es la mostrada en la figura 19-A.2.

Código

El código que deberá ejecutar para esta práctica se muestra en la figura 19-A.4, en que se debe ingresar nombre y contraseña de un AP con acceso a internet.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura 19-A.5.

Resultados

En la LCD se mostrará el código 200 que indica que la primera petición se ejecutó adecuadamente, posteriormente en la LCD se mostrará el último registro de calidad del aire de alguna de las alcaldías o municipios de la ZMVM.

Finalmente, en la LCD se mostrará el código 404 que indica que la segunda petición fue mal especificada, ya que se solicitó un recurso no disponible.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica.

https://bit.ly/2AV1LLg

```
GET /api/records/1.0/search/?dataset=prueba_datos_calidad_aire&rows=1 HTTP/1.1
Host: datos.cdmx.gob
```

Figura 19-A.3 *Petición HTTP que se envía a la API para obtener el último registro de la calidad del aire de alguna alcandía o municipio de la ZMVM.*

```
GET /api/records/1.0/search/?dataset=prueba_datos_calidad_aire&rows=1 HTTP/1.1
Host: datos.cdmx.gob
```

Figura 19-A.2 *Petición HTTP errónea que se envía a la API.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

#define wrong_uri "/api/records/1.0/search/?dataset=prueba&rows=1"
long codigo_estado_http=0;
char json[608];
long i=0;
long aux=0;

void main()
{
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /* En el canal de comunicación 0, se establece un cliente SSL*/
    create_ssl_pclient("datos.cdmx.gob.mx",443,0);
    /*Se envía petición HTTP para obtener el último registro de la calidad
    del aire de al alguna alcaldía o municipio de la ZMVM*/
    send_client(0,"GET /api/records/1.0/search/?dataset=prueba_datos_calidad_aire");
    send_client(0,"&rows=1 HTTP/1.1\r\nHost: datos.cdmx.gob");
    codigo_estado_http=get_complete_json(0,".mx\r\n\r\n", json, sizeof(json));
    delay_ms(1500);
    delete_client(0);
    lcd_gotoxy(5,1);
    lcd_putc("Codigo:");
    lcd_gotoxy(7,2);
    printf(lcd_putc,"%lu",codigo_estado_http);
    delay_ms(3000);
    lcd_putc("\f");
    lcd_gotoxy(1,1);
    while( json[i]!=0)
//Recorre el contenido de la respuesta almacenada y la muestra en la LCD
    {
        printf(lcd_putc,"%c",json[i]);
        i++;
        aux++;
        delay_ms(50);
        if(aux==16)
        {
            lcd_gotoxy(1,2);
        }
        else if(aux==32)
        {
            delay_ms(1000);
            lcd_putc("\f");
            lcd_gotoxy(1,1);
            aux=0;
        }
    }
    delay_ms(3000);
    lcd_putc("\f");
    //Ejecuta una petición HTTP incorrecta
    codigo_estado_http=request_http_client(0,1,"datos.cdmx.gob.mx",1,wrong_uri,443);
    lcd_gotoxy(5,1);
    lcd_putc("Codigo:");
    lcd_gotoxy(7,2);
    printf(lcd_putc,"%lu",codigo_estado_http);
    delay_ms(6000);
}
```

Figura 19-A.4 *Código de la práctica 19-A.*

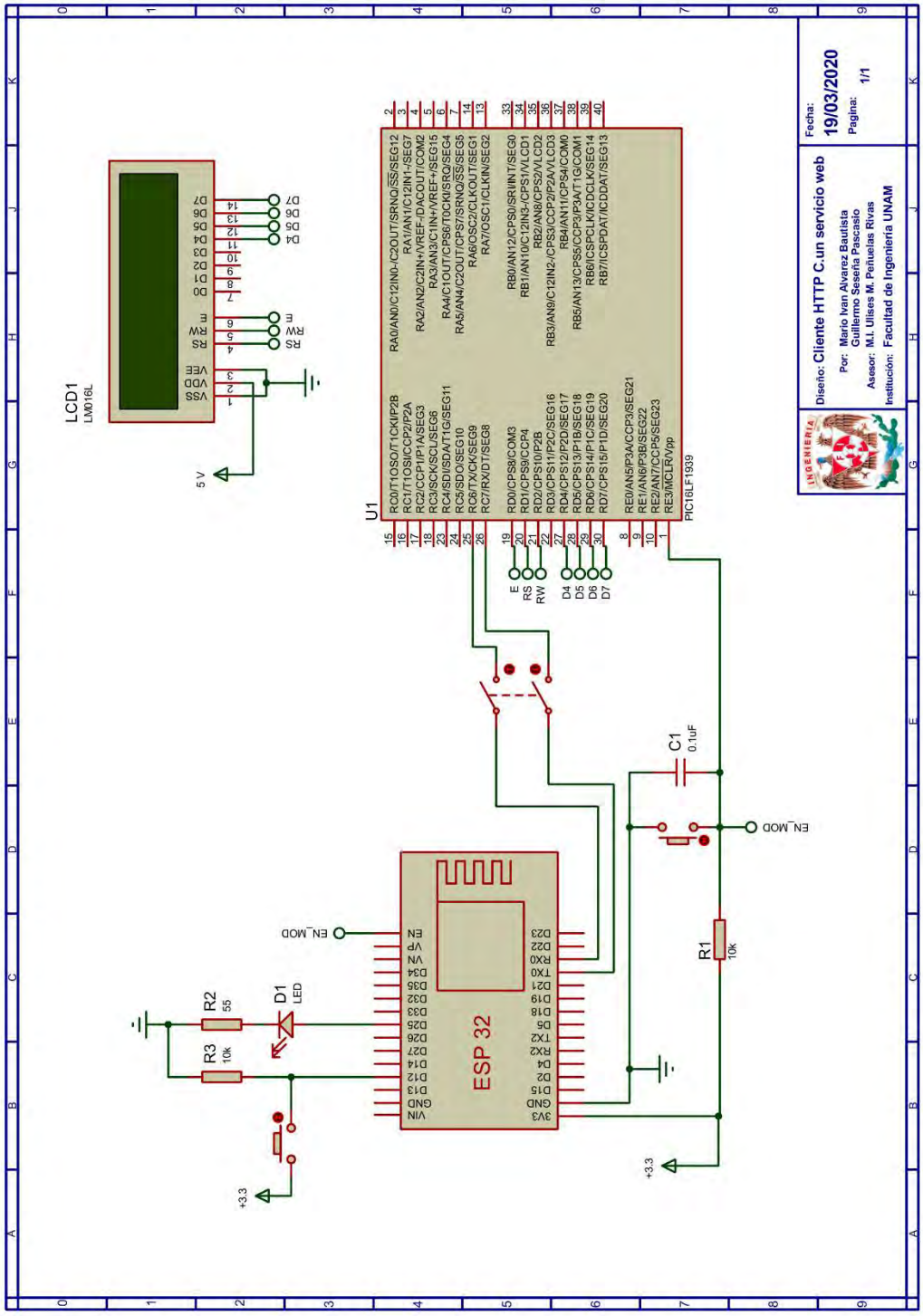


Figura 19-A.5 *Circuito de la práctica 19-A.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.
2. Rodriguez, A. *Servicio Web de RESTful: Los aspectos básicos*. 2015 [Citado 2020 Marzo]; Disponible en:https://www.ibm.com/developerworks/ssa/library/ws-restful/index.html.

Práctica 19-B Cliente HTTP: simulando un servicio web

Introducción

En prácticas anteriores se ha mostrado que el módulo ESP puede funcionar como un servidor o un cliente HTTP, por lo que es posible comunicar módulos entre si utilizando este protocolo. Utilizar HTTP implica que los módulos desempeñen alguno de los dos roles. Un PIC-ESP podría ser utilizado como un servidor para recabar información de otros módulos que funcionan como clientes, y a la vez otros clientes podrían consultar la información sin comunicarse con los que la originaron.

Utilizar HTTP, ha permitido interoperabilidad en las aplicaciones en línea, por lo que utilizar HTTP para comunicar PIC-ESP permitiría generar aplicaciones que van más allá de comunicar dos dispositivos[1].

Objetivo

Utilizar el protocolo HTTP para comunicar dos módulos ESP

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01 y un ESP32.

Habilitación de aplicaciones

- STATION, E_WEB_SERVER y E_JSON para el ESP-01
- STATION, E_HTTP_CLIENT para el ESP32

Materiales

- 1 computadora
- 2 microcontrolador PIC
- 2 módulo WiFi
- 2 interruptor DPST
- 6 resistores de 10 kΩ
- 2 botón pulsador n.o.
- 2 condensador de 0.1 µF
- 1 pantalla LCD 16x2
- 1 potenciómetro de 10 kΩ

Descripción

Aunque la biblioteca ESP no ofrece todas las funcionalidades para que un PIC-ESP ofrezca un servicio web basado en una arquitectura REST completa, para esta práctica se intentará simular uno, por lo que se utilizarán dos PIC-ESP.

El microcontrolador asociado al ESP-01 será establecido como un servidor web, mientras que el otro microcontrolador

será establecido como un cliente que envía peticiones HTTP al primero.

El PIC-ESP que funcionará como servidor ofrecerá dos recursos:

- _r0_ será accedido a través del método PUT para actualizar el último registro de las lecturas que un cliente envía.
- _r1_ será accedido mediante el método GET para obtener el último registro de las lecturas registradas.

El PIC-ESP que funciona como cliente tomará una lectura de una señal analógica producida por la variación de la posición del eje de un potenciómetro. El valor leído será enviado al servidor, a través de la petición HTTP, cuya estructura se muestra en la figura P19-B.1.

```
PUT /_r0_ HTTP/1.1
Host: {dirección IP del servidor web}
Connection: close
Content-Length: 12
Content-Type: application/json; charset=utf-8

{“pot”:0000}
```

Figura P19-B.1 *Estructura de la petición HTTP que se ejecuta para registrar el valor de lectura.*

La lectura que se registre podrá ser consultada desde el navegador de algún dispositivo que se encuentre conectado a la misma red que la de los módulos, o de manera remota si se ha realiza un mapeo de puertos. La respuesta que proporciona el servidor tendrá la estructura que se muestra en la figura P19-B.2, por lo que en el navegador sólo se verá la información en formato JSON.

```
HTTP/1.1 200 OK
Content-Length: 12
Content-Type: application/json; charset=utf-8

{“pot”:0000}
```

Figura P19-B.2 *Estructura de la respuesta HTTP de la consulta del valor de lectura.*

Código

El código que deberá ejecutar el microcontrolador PIC que se desempeña como servidor es el que se en la figura P19-B.4, en que se debe ingresar nombre y contraseña de un AP.

Mientras que, el código que deberá ejecutar el microcontrolador PIC que será cliente es el que se indica en la figura P19-B.5, en que se debe ingresar los siguientes parámetros:

- Nombre, contraseña de un AP.
- Dirección IP del módulo que funciona como servidor.

Circuito

El circuito para el PIC-ESP que funciona como servidor es el que se muestra en la figura P19-B.6, mientras que para el cliente el de la figura P19-B.7.

Resultados

Primero se ejecutará el código del uC PIC que está asociado al módulo que funcionará como servidor. Éste mostrará en la LCD la dirección IP que se debe especificar en el programa del PIC-ESP que funcionará como cliente.

Cuando se ejecute el código del cliente, en la LCD del servidor se observará el valor de la lectura que ha enviado el cliente.

El valor de la lectura se consultará desde un navegador, ingresando la dirección IP del servidor y solicitando el recurso _r1_, el valor se mostrará en formato JSON (ver figura P19-B.3)

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y los códigos de la práctica:

https://bit.ly/37H3mR5

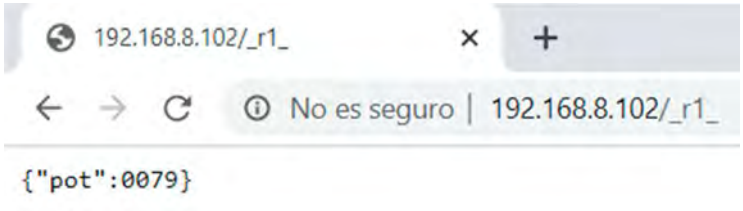


Figura P19-B.3 *Lectura accedida desde un navegador.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char IP[21]={0};
char json_buffer[21]={0};
int recurso=0;
char json[21]="{\"pot\":0000}";
char str[5]="\0";

void main()
{
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    get_ip_station(IP,sizeof(IP));
    lcd_gotoxy(2,1);
    lcd_putc("Direccion IP:");
    lcd_gotoxy(2,2);
    printf(lcd_putc,"%s",IP);
    //Establece buffer en que se alojan mensajes en formato JSON provenientes de los clientes
    set_http_sjsonbuffer(json_buffer,sizeof(json_buffer));
    //Crea el servidor web para el puerto 80 con un timeout de 120s
    create_http_server(80,4,120,&recurso);
    while(1)
    {
        //Actualiza el recurso solicitado para el primer elemento de la cola
        http_upgrade_resource();
        switch (recurso)
        {
            //Recurso que actualiza el valor de lectura
            case '0':
            get_value_http_sjson("pot",str,sizeof(str));
            json[7]=str[0];
            json[8]=str[1];
            json[9]=str[2];
            json[10]=str[3];
            lcd_putc("\f");
            lcd_gotoxy(6,1);
            lcd_putc("Valor:");
            lcd_gotoxy(7,2);
            printf(lcd_putc,"%s",str);
            http_response_put();
            break;

            //Recurso que recupera el valor de la lectura
            case '1':
            http_response_json(json);
            break;

            default:
            http_default_resource();
            break;
        }
        delay_ms(50);
    }
}
```

Figura P19-B.4 *Código de la práctica 19-A para el servidor.*

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define server_ip "dirección ip del servidor web"

long codigo_estado_http=0;
char json[21]="{\"pot\":0000}";
char str_valor[5]="\0";
long valor=0;
void main()
{
    setup_adc(adc_clock_div_64);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Establece buffer en que se alojan mensajes JSON provenientes del servidor web
    set_http_cjsonbuffer(json,sizeof(json));
    while (1)
    {
        //Obtención de valor
        set_adc_channel(10);
        valor=(read_adc()>>1);
        sprintf(str_valor,"%04lu",valor);
        json[7]=str_valor[0];
        json[8]=str_valor[1];
        json[9]=str_valor[2];
        json[10]=str_valor[3];
        /*Ejecuta petición HTTP en el canal 0 a través de una conexión TCP,
        el método utilizado es PUT(2)*/
        codigo_estado_http=request_http_client(0,0,server_ip,2,"/_r0_",80,json);
        delay_ms(1000);
    }
}
```

Figura P19-B.5 *Código de la práctica 19-B para el cliente.*

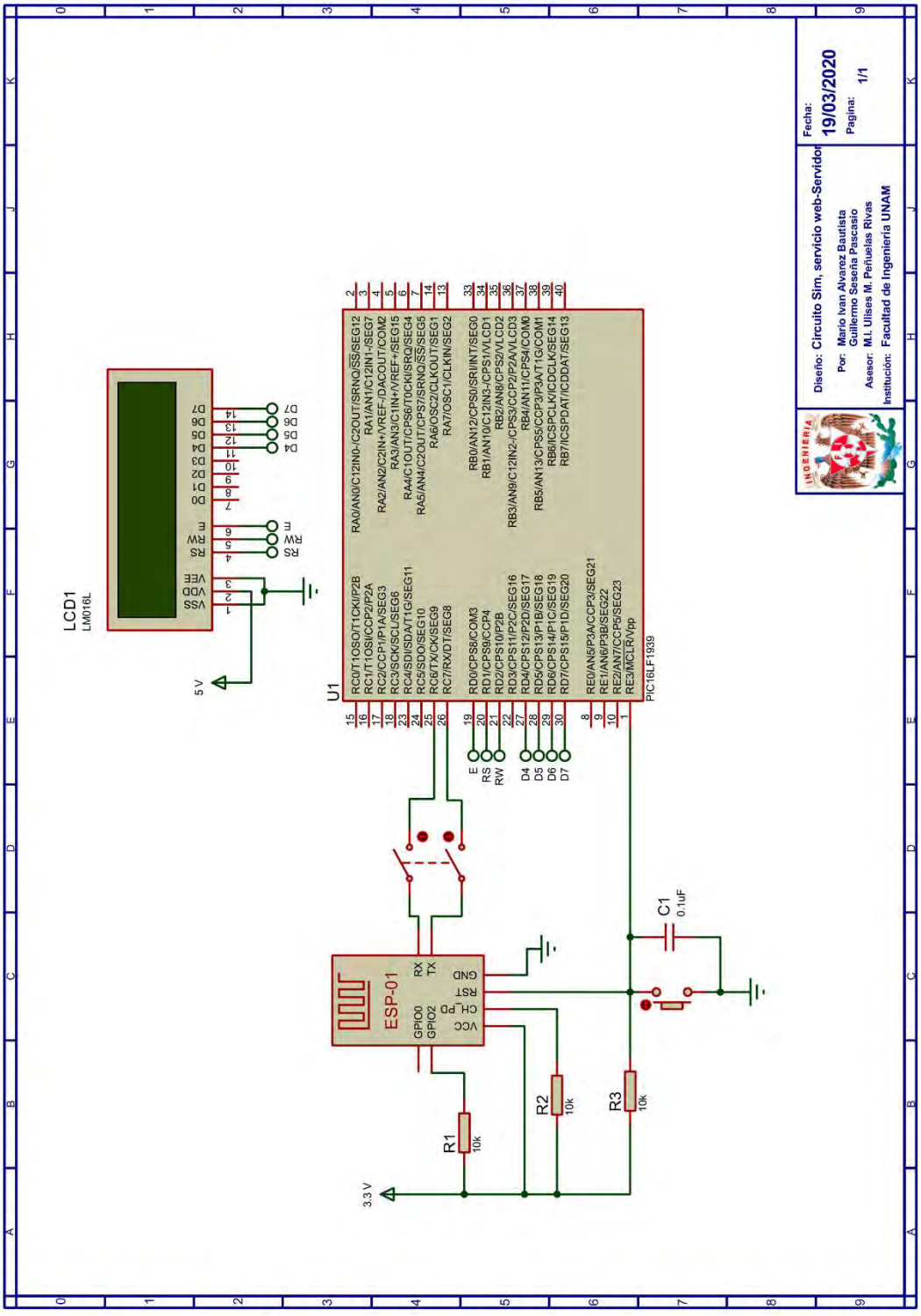


Figura P19-B.6 *Circuito de la práctica 19-A del servidor.*

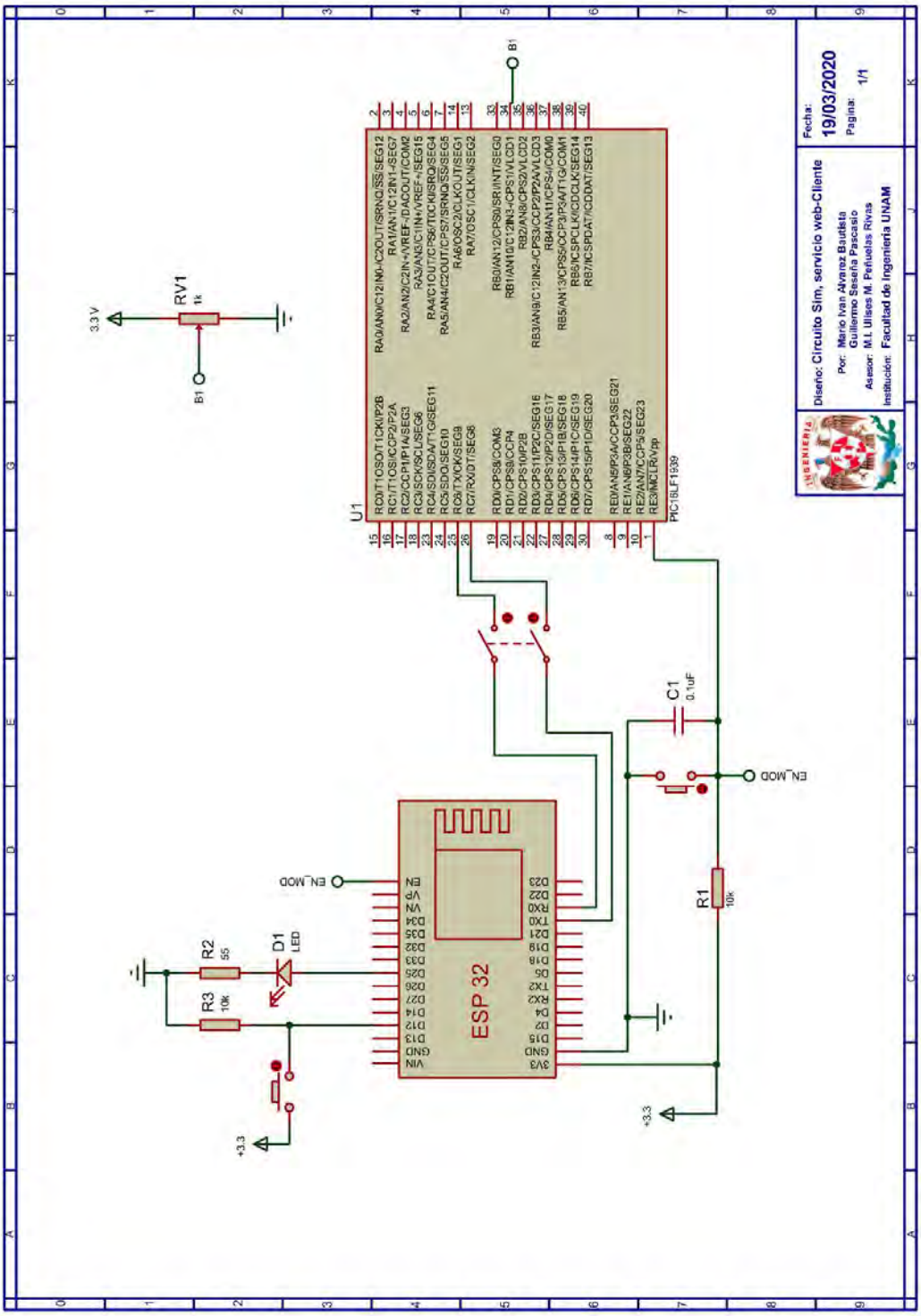


Figura P19-B.7 *Circuito de la práctica 19-A del cliente.*

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.

Práctica 20 Websocket

Introducción

El modelo de solicitud-respuesta usada en HTTP no es tan eficiente para aplicaciones que requieren de bajas latencias. En este modelo, el servidor no puede enviar información al cliente hasta que éste la solicite, por lo que el cliente es quien dirige la comunicación, ejecutando peticiones HTTP para este fin.

En aplicaciones que necesitan conocer la información más actual del servidor, el cliente tiene que preguntar por ella frecuentemente. Por cada solicitud que realice, se abre y se cierra una conexión TCP/SSL/TLS lo que puede generar un rendimiento pobre. En algunas aplicaciones se reutiliza la conexión realizando más de una petición, lo cual mejora sustancialmente el rendimiento de la aplicación, aun así, tienen el inconveniente que el intervalo de tiempo entre cada petición debe ser menor a la latencia permitida por la aplicación.

Existe otro método conocido como *long polling* en el que el cliente realiza una petición para obtener la información más actual, si el servidor tiene nueva información responderá a la petición de no ser así, retrasará su respuesta hasta obtener información más reciente o hasta que venza el *timeout* del servidor, sin embargo, esta técnica aún deja a cargo al cliente de solicitar la información.

Una alternativa para aplicaciones en tiempo real es *websocket*, un protocolo de aplicación creado para eliminar las limitaciones del modelo de petición/respuesta del protocolo HTTP, el cual otorga un medio libre de comunicación en el que la información fluye sin formato hasta que se implemente algún conjunto de reglas para que ambas partes puedan interpretar y distinguir la información. Este protocolo proporciona una conexión TCP o SSL/TLS simple destinada a aplicaciones web, la cual se mantiene abierta durante todo el intercambio de información entre el cliente y el servidor Dando como resultado, la posibilidad de que el servidor envíe mensajes al cliente, sin que este lo haya solicitado primero [1].

En la biblioteca ESP se ha desarrollado el *software* para establecer al PIC-ESP como un *websocket server*, bajo ciertas limitaciones. En ésta se permite establecer una sola conexión TCP, por lo que solo puede atender un cliente a la vez, si se intenta establecer conexiones paralelas: para la versión IDF, se rechaza la conexión hasta que haya concluido la anteriormente establecida; mientras que para las versiones SDK, cada vez que se solicite una conexión paralela, el PIC-ESP cierra la conexión anterior y establece una nueva conexión con el nuevo cliente.

```
GET /{cualquier URL} HTTP/1.1
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key: {clave aleatoria de 24 caracteres}
```

Figura P20.1 *Petición que espera la biblioteca ESP para establecer la comunicación con websocket.*

Aunque *websocket* es ajeno a HTTP, para establecer este tipo de comunicación el cliente y el servidor realizan un proceso de negociación, que se realiza mediante una petición HTTP (*Upgrade request*). Para biblioteca ESP, el cliente que desee conectarse al *websocket* debe especificar en la petición, al menos lo que se muestra en la figura P20.1. El recurso que solicite el cliente no importa, aunque es recomendable utilizar uno con un nombre breve, la clave aleatoria que sirve para verificar que realmente el cliente desea establecer esta comunicación, debe contener 24 caracteres ASCII.

Cuando la biblioteca identifica que un cliente ha enviado una petición válida, entonces otorga una respuesta HTTP que contiene lo que se muestra en la figura P20.2. El *Sec-WebSocket-Accept* otorga un clave hash de 28 bytes, que es

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: {clave hash}
```

Figura P20.2 *Respuesta que la biblioteca ESP otorga durante establecimiento de la comunicación del websocket.*

el resultado de convertir a base64 el cálculo en SHA-1 de la concatenación del *Sec-WebSocket-Key* y la *magic strin*g "258EAFA5-E914-47DA-95CA-C5AB0DC85B11".

El cálculo SHA-1 que se realiza durante la negociación y tiene dos alternativas de ejecución: una es a través de los métodos criptográficos que el módulo dispone, por lo que durante la negociación el PIC ejecuta un comando AT extra para obtener el cálculo; y la otra es a través de la función que se ha programado en la biblioteca ESP que ejecuta dicho cálculo y que se activa con E_INTERNAL_SHA1, sin embargo, éste demora más tiempo y consume más memoria del uC PIC que el de las funciones criptográficas del módulo.

Después de finalizada la negociación, el servidor y el cliente intercambian información a través de tramas de datos (*data frames*). La biblioteca ESP sólo puede reconocer tramas de texto, tramas binarias (bytes) y tramas de cierre de conexión. Las tramas binarias y de texto llegan con un cifrado XOR, que la biblioteca descifra y las almacena en un *buffer* para que el usuario puede acceder

a ellas. Para enviar información, la biblioteca ESP únicamente puede transmitir tramas binarias (bytes) y de texto con un contenido menor a 126 bytes por mensaje.

Ya que flujo de información entre el servidor y cliente carece de formato en comparación con HTTP, se ha incluido la opción de recibir información en formato JSON como en servidor web, está opción se habilita con la herramienta E_JSON que se encuentra en el archivo esp.h. Al activar dicha opción, el cliente debe enviar al módulo la representación textual de pares simples como una trama binaria insertando un byte inicial con valor igual a uno.

También para facilitar el desarrollo de aplicaciones que utilicen esta tecnología se han incluido funciones que envían tramas binarias en la que se puede insertar un byte inicial para que el cliente pueda distinguir distintos tipos de datos.

Objetivo

Proporcionar una página web dinámica que intercambie información con el módulo mediante *websocket.*

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION, E_WEBSOCKET_SERVER y E_JSON deben ser TRUE en esp.h

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 4 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 1 condensador de 0.1 μF
- 1 pantalla LCD 16x2
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 56 Ω

Descripción

Se proporcionará una página web dinámica, que contiene un *script* en javascript, el cual creará un *websocket client* que se conectará al *websocket server* del módulo con el fin de actualizar secciones de la página y enviar

información relacionada con la interacción del usuario.

La página permitirá al usuario controlar el estado de un LED; visualizar el valor de un contador que incrementa en el *loop* del programa principal del PIC, el cual tiene como fin mostrar la rapidez con la que se actualiza una página utilizando esta tecnología; y finalmente el envío de mensajes al PIC, los cuales se muestran en una LCD.

El módulo ESP proporcionará dos puertos: el primer puerto es el 80, en el cual se establece un servidor web que proporcionará a los clientes (navegadores) el documento HTML de la página (que incluye el *script* del *websocket client*); y el segundo puerto es el 5050, en el que se establece el *websocket server,* que se encargará de transferir información con el cliente que se creará durante la ejecución del *script* de la página que proporciona el primer puerto. El recurso web que otorgará el documento HTML de la página es definido en el recurso _r0_ del *switch* de atención de peticiones HTTP del servidor web.

Derivado de las características de *websocket*, el PIC-ESP deberá distinguir de la información entrante, el conjunto de solicitudes que enviará el *script* de la página para actualizar el contenido de la página y la información que modifica los estados del servidor (LED y *string* mostrado en la LCD). Mientras que, para la información saliente, el PIC-ESP deberá proporcionar algún formato para que el *script* pueda distinguir entre fragmentos de código HTML e información que indique los cambios de estados del servidor.

Para distinguir el tipo de información que recibe el módulo proveniente del *websocket client*, se establecerá que las peticiones de actualización de contenido deben ser enviadas como tramas de texto. Mientras que, para la información sobre cambios en los estados del servidor, se aprovechará la recepción de información en formato JSON de la biblioteca, en la que se recibe los mensajes de manera binaria (bytes), cuyo byte inicial tiene un valor igual a uno. Todos los posibles mensajes que el módulo puede reconocer del *websocket client* para esta práctica se muestran en la tabla P20.1.

Para la información que envía el *websocket server*, los fragmentos de código HTML se enviarán como tramas binarias, a los cuales se les debe insertar un byte inicial cuyo valor será dos. La información que indica a la página de cambios de estado del servidor se enviará en formato JSON como tramas binarias insertando un byte inicial con valor uno.

El documento HTML que envía el servidor web usa la estructura y estilo

Tabla P20.1 *Mensajes que el websocket server identifica para esta práctica.*

Mensaje	Tipo de trama	Descripción
L	Texto	Solicita el contenido de la página para la opción de control del LED
C	Texto	Solicita el contenido de la página para la opción que muestra el valor de contador
E	Texto	Solicita el contenido de la página para la opción de envío de mensajes
\1{LED:"on"}	Binaria	Enciende LED
\1{LED:"off"}	Binaria	Apaga LED
\1{Msj:"mensaje"}	Binaria	Sustituye el *string* que se muestra en la LCD por el valor de la clave Msj
M	Texto	Solicita al servidor que muestre en la LCD el mensaje que ha envido el usuario

utilizado en prácticas anteriores, pero incluye el *script* que se comunicará con el *websokcet server* del módulo, dicho *script* se encargará de establecer la conexión y manejará el envío/recepción de información. El código HTML utilizado en esta práctica se muestra en la figura P20.5, figura P20.6 y figura P20.7.

El *script* de esta práctica utiliza la API de WebSocket de javascript para manejar un cliente websocket. Sólo se explica el funcionamiento general del *script* utilizado, para mayor información acerca de la API puede consultar en [2]. Al cargar la página, el *script* comprobará que el navegador utilizado sea compatible con *websocket,* indicándole al usuario si es o no compatible. Si es compatible, el *script* intentará conectarse con el *websocket server* del módulo, que al establecerse correctamente le informará al usuario de su éxito.

En el *script* se definió la función que enviará texto al módulo ESP (`snd`). Esta función será utilizada para solicitar fragmentos de código HTML que actualizará el contenido de la página cuando el usuario interactúa con algunas de las tres opciones que ofrece la página.

Cuando el módulo envíe fragmentos de código HTML, se ejecutará el evento de recepción de mensajes (`ws.onmessage`), en el que se verificará que sea una trama binaria y que su primer byte tenga un valor igual a dos, entonces sustituirá el contenido del *div* ubicado en el *main* de la página por el resto del mensaje

recibido, el cual es tratado como una cadena de caracteres ASCII. También en este evento, se identificará los mensajes que actualizan la información que se muestra al usuario acerca del estado del LED o del valor del contador, para estos mensajes se verificará que sean tramas binarias y que el primer byte tenga un valor igual a uno, entonces el resto del mensaje es tomado como un JSON.

Si se recibe la clave “LED”, se utiliza su valor para modificar el texto que indica al usuario el estado del LED. Si se recibe la clave “C”, entonces se utiliza su valor para actualizar el valor del contador que se mostrará en la pantalla. También en el *script* se definió la función que envía al *websocket server* mensajes binarios en formato JSON (`snd_JSON`) para encender o apagar el LED, además para modificar el *string* que se muestra en la pantalla LCD.

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica es el mostrado en figura P20.8, figura P20.9 y figura P20.10, en que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P20.12.

Resultados

Al comenzar la ejecución del programa del PIC, se observará la dirección del módulo en la LCD. Esta dirección será utilizada para acceder al recurso _r0_ desde un navegador web. Al ingresar se cargará una página, aparecerán dos mensajes: el primero indicará que el navegador que se está usando es compatible con *websocket*, y el segundo indicará que se ha establecido conexión con el *websocket server* (ver figura P20.3).

En la página se mostrará tres opciones: una para controlar el estado de un LED, la segunda para visualizar el valor del contador del programa del PIC y finalmente la opción para enviar mensajes (ver figura P20.4). Al ingresar a cualquiera de éstas, se actualizará el contenido de la sección del *main* de la página, no se bloquea la interacción del usuario ni se redirecciona a otro recurso.

Al ingresar a la opción del control del LED, se muestrarán dos botones con los que se cambiará el estado de dicho elemento. Al realizarlo se actualiza el texto que indica el estado. En esta sección se podrá verificar la comunicación bidireccional, al presionar el botón que está conectado al PIC, el LED se encenderá o apagará, y se mostrará en la página el cambio de estado, sin necesidad de recargar la página.

Para la opción del contador se podrá observar como el contador incrementa su valor en una unidad, estos valores se muestran con mayor rapidez que en la práctica AJAX, ya que inmediatamente que el servidor genera un nuevo valor, lo envía a la página, la cual está predispuesta para mostrarlo en la pantalla. Finalmente, en la sección de envío de mensajes, cada mensaje envidado será mostrado en la pantalla LCD.

Si se abre otra pestaña en el navegador solicitando el recurso _r0_, entonces en la pestaña anterior se mostrará el mensaje de que se ha cerrado la comunicación, mientras que en la nueva pestaña se indicará que se ha establecido la conexión, por lo que la sesión se continuará en la nueva pestaña.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y los códigos de la práctica:

https://bit.ly/3hMVbXR

Nota

- La estructura del intercambio de información del *websocket* en esta práctica fue seleccionada arbitrariamente, con el fin de utilizar la mayoría de las características que la biblioteca ESP dispone para *websocket server*. Los usuarios pueden establecer el formato que deseen para el flujo de información en otras aplicaciones, siempre y cuando esté dentro de las capacidades que otorga la biblioteca ESP.

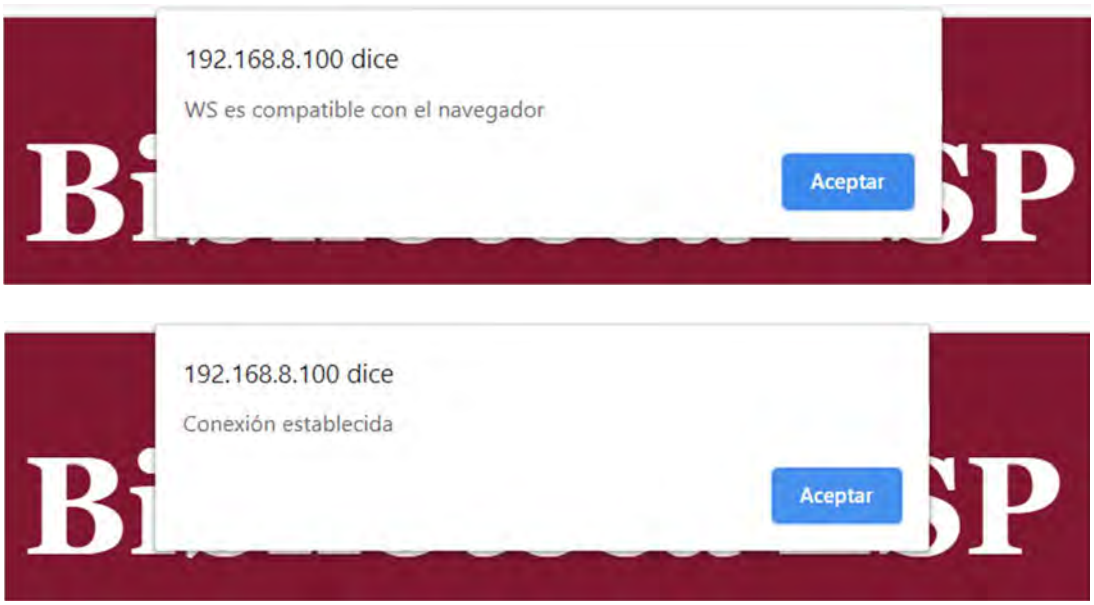


Figura P20.3 *Mensajes que indican que el navegador utilizado es compatible con Websocket y que la página ha establecido conexión en el Websocket.*

Figura P20.4 *Página de inicio de la práctica 20.*

```
<!DOCTYPE html><html><head><meta charset="utf-8"/><title>WebSocket</title>
<style type="text/css">
body{background-color:#EBEBEB;margin:0;height: 100vh;}
.c1{width:96%;margin-left:2%;overflow:hidden;}
header{background-color:#811832;text-align:center;}
nav{background-color:black;text-align: left;}
main{background-color:white;font-size:22pt;text-align: center;min-height: 50vh;}
footer{background-color:#3F3F3F;text-align: center;}
h1{font-family:Georgia;font-size:60pt;color:white;}
h2{font-size:26pt;color:white;}
.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:white;background-color:black;}
.b_1:hover{color:black;background-color:white;}
.b_2{padding:6px;font-family:Georgia;font-size:20pt;color:black;background-color:#fff57a;}
.b_2:hover{color:#fff57a;background-color:black;}
</style></head>
<body>
<header class="c1"><h1 onclick="location='_r0_'">Biblioteca ESP</h1></header>
<nav class="c1">
<table><tr>
<td><button class="b_1" onclick="snd('L');">LED</button></td>
<td><button class="b_1" onclick="snd('C');">Contador</button></td>
<td><button class="b_1" onclick="snd('E');">Envío</button></td>
</tr></table></nav>
<main class="c1">
<div id="d_cont"><br><br><p
>Diríjase al control del LED, al Contador o al envío de mensajes.</p>
</div></main>
<footer class="c1"><h2>Febrero 2020</h2></footer>
</body></html>

<script type = "text/javascript">
//Si websocket es compatible con el navegador
if ("WebSocket" in window) {
    alert("WS es compatible con el navegador");
    //Abre conexión en el puerto del PIC-ESP destinado al websocket
    var ws=new WebSocket("ws://IP DEL MÓDULO:5050/echo",null);
    ws.binaryType="arraybuffer";
    ws.onopen=function() {
        alert("Conexión establecida");
    };
    //Al recibir información del websocket
    ws.onmessage=function (evt) {
        //Obtiene el payload
        var bu=evt.data;
        //Si el mensaje es binario (bytes)
        if(bu instanceof ArrayBuffer){
            var B_bu=new Uint8Array(bu);
            /*Obtiene el primer byte del arreglo para saber que tipo de
            dato es de acuerdo lo que se estableció para la práctica*/
            var typ=B_bu[0];
```

Figura P20.5 *Primera parte del documento HTML utilizado en la práctica 20.*

```
        //Obtiene el resto de la carga útil del mensaje
        var dt=new TextDecoder("utf-8").decode(B_bu.slice(1, B_bu.length));
        var cont;
        //Si es informacion en formato JSON
        if(typ==1)
        {
            //Convierte el texto en un dato JSON
            var jn=JSON.parse(dt);
            //Busca clave "L" que contiene el estado del LED
            if(jn.hasOwnProperty('L'))
            {
                cont=document.getElementById('LED');
                if (cont!==null)
                {
                    //Actualiza texto del estado LED
                    if(jn.L=='on')
                        cont.innerHTML='LED Encendido';
                    else if(jn.L=='off')
                        cont.innerHTML='LED Apagado';
                }
            }
            //Busca clave C que contiene el valor del contador
            else if(jn.hasOwnProperty('C'))
            {
                //Actualiza texto del contador
                cont=document.getElementById('cntr');
                if (cont!==null)
                    cont.innerHTML=jn.C;
            }
        }
        //Si es HTML
        else if(typ==2)
        {
            //Actualiza la sección del main
            cont=document.getElementById('d_cont');
            if(cont!==null)
                cont.innerHTML=dt;
        }
    }
};
//Envia texto al websocket
function snd(str){
    ws.send(str);
}
```

Figura P20.6 *Segunda parte del documento HTML utilizado en la práctica 20.*

```
    /*Envía datos binarios al websocket insertando un primer byte con valor 1
    para que la biblioteca lo interprete como mensaje en formato JSON*/
    function snd_JSON(obj)
    {
        var js=JSON.stringify(obj);
        var dt=new TextEncoder("utf-8").encode(js);
        ar=new Uint8Array(dt.length+1);
        //Inserta un primer byte
        ar[0]=1;
        //Inserta el mensaje
        ar.set(dt,1);
        //Envia arreglo de bytes
        ws.send(ar);
    }
    //Envía mensaje para encender o apagar LED
    function snd_LED(str){
        var obj=new Object();
        obj.LED=str;
        snd_JSON(obj);
    }
    //Envía mensaje que se muestra en la LCD
    function snd_Msj(str){
        var obj=new Object();
        obj.Msj=str;
        snd_JSON(obj)
    }
    ws.onclose=function() {
        alert("Conexión cerrada");
    };
}
else{
    alert("WS no es compatible con el navegador");
}
</script></html>

<!--Se muestra el contenido del main (div "d_cont") que el PIC-ESP envia para las tres opcion
es-->
<!--Para la opción del control del LED-->
<!--
    <br><br><br><button class="b_2" onclick="snd_LED('on');">Encender</button>
    <button class="b_2" onclick="snd_LED('off');">Apagar</button><br>
    <p id="LED"></p>
-->

<!--Para la opción de visualizar el valor del contador-->
<!--
    <br><p>Escriba lo que desea enviar:</p>
    <input type="text" ID="str">
    <br><br><button class="b_2" onClick="snd_Msj(str.value);snd('M');">Enviar</button>
-->

<!--Para la opción de envío de mensajes al PIC-->
<!--
    <br><br><p>Contador:</p><p id="cntr"></p>
-->
```

Figura P20.7 *Tercera parte del documento HTML utilizado en la práctica 20.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char json_buffer[15]={0};
char server2_buffer[21]={0};
char IP[21]={0};
int recurso=0;
char json_contador[]="{\"C\":\"000\"}";
const uint8 json_onLED[]="{\"L\":\"on\"}";
const uint8 json_offLED[]="{\"L\":\"off\"}";
char mensaje[10]={0};
int contador=0;
char str_contador[4];
short print_message=0;
short send_div_LED=0;
short send_div_CONTADOR=0;
short send_div_ENVIO=0;
short send_valor_contador=0;


const int base_1[]=
"<!DOCTYPE html><html><head><meta charset=\"utf-8\"/><title>WebSocket</title><style type=\"te
xt/css\">body{background-color:#EBEBEB;margin: 0;height: 100vh;}header{background-color:#8118
32; text-align: center; min-height: 25vh;}nav{background-color:black;text-align: left;}main{b
ackground-color:white;font-size:22pt;text-align: center;min-height: 50vh;}footer{background-c
olor:#3F3F3F;text-align: center;}h1{font-family:Georgia;font-size:60pt;color:white;}h2{font-s
ize:26pt;color:white;}.c1{width:96
%
;margin-left:2%;overflow:hidden;}.b_1{padding:6px;font-family:Georgia;font-size:26pt;color:wh
ite;background-color:black;}.b_1:hover{color:black;background-color:white;}.b_2{padding:6px;f
ont-family:Georgia;font-size:20pt;color:black;background-color:#fff57a;}.b_2:hover{color:#fff
57a;background-color:black;}.t1 {table-layout:fixed;width:100
%
;border:0;}</style></head><body><header class=\"c1\"><h1 onclick=\"location='_r0_'\">Bibliote
ca ESP</h1></header><nav class=\"c1\"><table><tr><td><button class=\"b_1\" onclick=\"snd
('L');\">LED</button></td><td><button class=\"b_1\" onclick=\"snd('T');\">Contador</button></
td><td><button class=\"b_1\" onclick=\"snd('E');\">Envío</button></td></tr></table></nav><mai
n class=\"c1\"><div id=\"d_cont\"><br><br><p>Diríjase al control del LED, al Contador o al en
vío de mensajes.</p></div></main><footer class=\"c1\"><h2>Febrero 2020</h2></footer></body></
html>"
;
const int base_2[]=
"<script type = \"text/javascript\">if (\"WebSocket\" in window) {alert(\"WS es compatible co
n el navegador\");var ws=new WebSocket(\"ws://"
;
```

Figura P20.8 *Primera parte del código del microcontrolador PIC para la práctica 20.*

```
const int base_3[]=
":5050/echo\",null);ws.binaryType=\"arraybuffer\";ws.onopen=function() {alert(\"Conexión esta
blecida\");ws.send(\"Hola\");
};ws.onmessage=function (evt) {var bu=evt.data;if(bu instanceof ArrayBuffer){var B_bu=new Uin
t8Array(bu);var typ=B_bu[0];var dt=new TextDecoder(\"utf-8\").decode(B_bu.slice(1, B_bu.lengt
h));var cont;if(typ==1){var jn=JSON.parse(dt);if(jn.hasOwnProperty('L')){cont=document.getEle
mentById('LED');if (cont!==null){if(jn.L=='on')cont.innerHTML='LED Encendido';else if(jn.L=
='off')cont.innerHTML='LED Apagado';
}
}else if(jn.hasOwnProperty('C')){cont=document.getElementById('cntr');if (cont!==null)cont.in
nerHTML=jn.C;
}
}else if(typ==2){cont=document.getElementById('d_cont');if(cont!==null)cont.innerHTML=dt;
}
}
};function snd(str){ws.send(str);}function snd_JSON(obj){var js=JSON.stringify(obj);var dt=ne
w TextEncoder(\"utf-8\").encode(js);ar=new Uint8Array(dt.length+1);ar[0]=1;ar.set(dt,1);ws.se
nd(ar);
}function snd_LED(str){var obj=new Object();obj.LED=str;snd_JSON(obj);
}function snd_Msj(str){var obj=new Object();obj.Msj=str;snd_JSON(obj)
}ws.onclose=function() {alert(\"Conexión cerrada\");
};
}else{alert(\"WS no es compatible con el navegador\");
}
</script></html>";
const char cont_led[]=
"<br><br><br><button class=\"b_2\" onclick=\"snd_LED('on');\">Encender</button><button class=
\"b_2\" onclick=\"snd_LED('off');\">Apagar</button><br><p id=\"LED\"></p>"
;
const char cont_envio[]=
"<br><p>Escriba lo que desea enviar:</p><input type=\"text\" ID=\"str\"><br><br><button class
=\"b_2\" onClick=\"snd_Msj(str.value);snd('M');\">Enviar</button>"
;
const char cont_contado[]="<br><br><p>Contador:</p><p id=\"cntr\"></p>";

void main()
{
    set_tris_b(0b01111111);
    output_b(0);
    lcd_init();
    lcd_gotoxy(2,1);
    lcd_putc("Direccion IP:");
    begin_esp();
    begin_station("Estacion",SSID,password);
    get_ip_station(IP,sizeof(IP));
    lcd_gotoxy(2,2);
    printf(lcd_putc,"%s",IP);
    //Establece buffer en que se alojan la información que recibe el websocket
    set_buffer_websocket(server2_buffer,21);
    //Establece buffer en que se alojan mensajes en formato JSON provenientes de los clientes
    set_ws_jsonbuffer(json_buffer,15);
    /*Crea el servidor web para el puerto 80, permite cuatro conexiones simultáneas con un
    timeout  de 60s */
    create_http_server(80,2,60,&recurso);
    /*Crea el servidor (habilita puerto) para comunicación mediante websocket*/
    create_websocket_server(5050);
```

Figura P20.9 *Segunda parte del código del microcontrolador PIC para la práctica 20.*

```
while(1)
{
    //Se encarga de otorgar el documento HTML que contiene el script
    request_connection_websocket();
    http_upgrade_resource();
    switch (recurso)
    {
        case '0':
        http_send_const(base_1);
        http_send_const(base_2);
        http_send(IP);
        http_send_const(base_3);
        http_finish(1);
        break;

        default:
        http_default_resource();
        break;
    }

    //Se encarga de manejar el envio de informacion del websocket
    //Compara que contenido de página ha solicitado el usuario
    send_div_LED=compare_WSbuffer("L",1);
    send_div_CONTADOR=compare_WSbuffer("C",1);
    send_div_ENVIO=compare_WSbuffer("E",1);
    print_message=compare_WSbuffer("M",1);
    //Envia contenido de la página para actualizar la sección del main
    if(send_div_LED==1)//Envia el contenido de control del LED
    {
        send_binary_insert_websocket_const(cont_led,2);
        //Revisa estado del LED
        if(input_state(PIN_B7))
        {
            send_binary_insert_websocket_const(json_onLED,1);
        }
        else
            send_binary_insert_websocket_const(json_offLED,1);
    }
    else if(send_div_CONTADOR==1)//Envía contenido del visualizador del Contador
    {
        send_binary_insert_websocket_const(cont_contado,2);
    }
    else if(send_div_ENVIO==1)//Envía contenido del para enviar  mensajes
    {
        send_binary_insert_websocket_const(cont_envio,2);
    }
    //Si el usuario ha enviado un mensaje lo imprime en la LCD
    else if(print_message==1)//Obtiene el valor de la clave "Msj" en el buffer JSON
    {
        get_value_wsjson("Msj",mensaje,sizeof(mensaje));
        printf(lcd_putc,"\f");
        lcd_gotoxy(5,1);
        printf(lcd_putc,"Mensaje:");
        lcd_gotoxy(1,2);
        printf(lcd_putc,"%s",mensaje);
    }
```

Figura P20.10 *Tercera parte del código del microcontrolador PIC para la práctica 20.*

```
        //Desactiva el envio del valor del contador
        if(send_div_LED==1 || send_div_ENVIO==1)
            send_valor_contador=0;
        else if(send_valor_contador==0 && send_div_CONTADOR==1)
            send_valor_contador=1;

        //Envia el valor del contador
        if(send_valor_contador==1)
        {
            //json_contador="{\"C\":\"\0";
            sprintf(str_contador,"%u",contador);
            for(int i=0; i<4; i++)
            {
                if(str_contador[i]=='\0')
                {
                    json_contador[6+i]='"';
                    json_contador[7+i]='}';
                    json_contador[8+i]='\0';
                    break;
                }
                else
                    json_contador[6+i]=str_contador[i];
            }
            send_binary_websocket(json_contador,get_str_length(json_contador),1);
        }

        //Compara si existe un nuevo mensaje la clave "LED" con un valor "on"
        if(compare_wsjson("LED","on",1))
        {
            output_bit(PIN_B7,1);//Enciende LED
            send_binary_insert_websocket_const(json_onLED,1);
        }
        //Compara si existe un nuevo mensaje la clave "LED" con un valor "off"
        if(compare_wsjson("LED","off",1))
        {
            output_bit(PIN_B7,0);//Apaga LED
            send_binary_insert_websocket_const(json_offLED,1);
            //WSServer_SendText_Const(aLed);
        }

        //Control del LED desde el botón
        if(1==input_state(PIN_B6))
        {
            output_toggle(PIN_B7);
            if(1==input_state(PIN_B7))
            {
                send_binary_insert_websocket_const(json_onLED,1);
            }
            else
                send_binary_insert_websocket_const(json_offLED,1);
            while(1==input_state(PIN_B6))
            {

            }
          }
        contador++;
        delay_ms(20);
    }
}
```

Figura P20.11 *Cuarta parte del código del microcontrolador PIC para la práctica 20.*

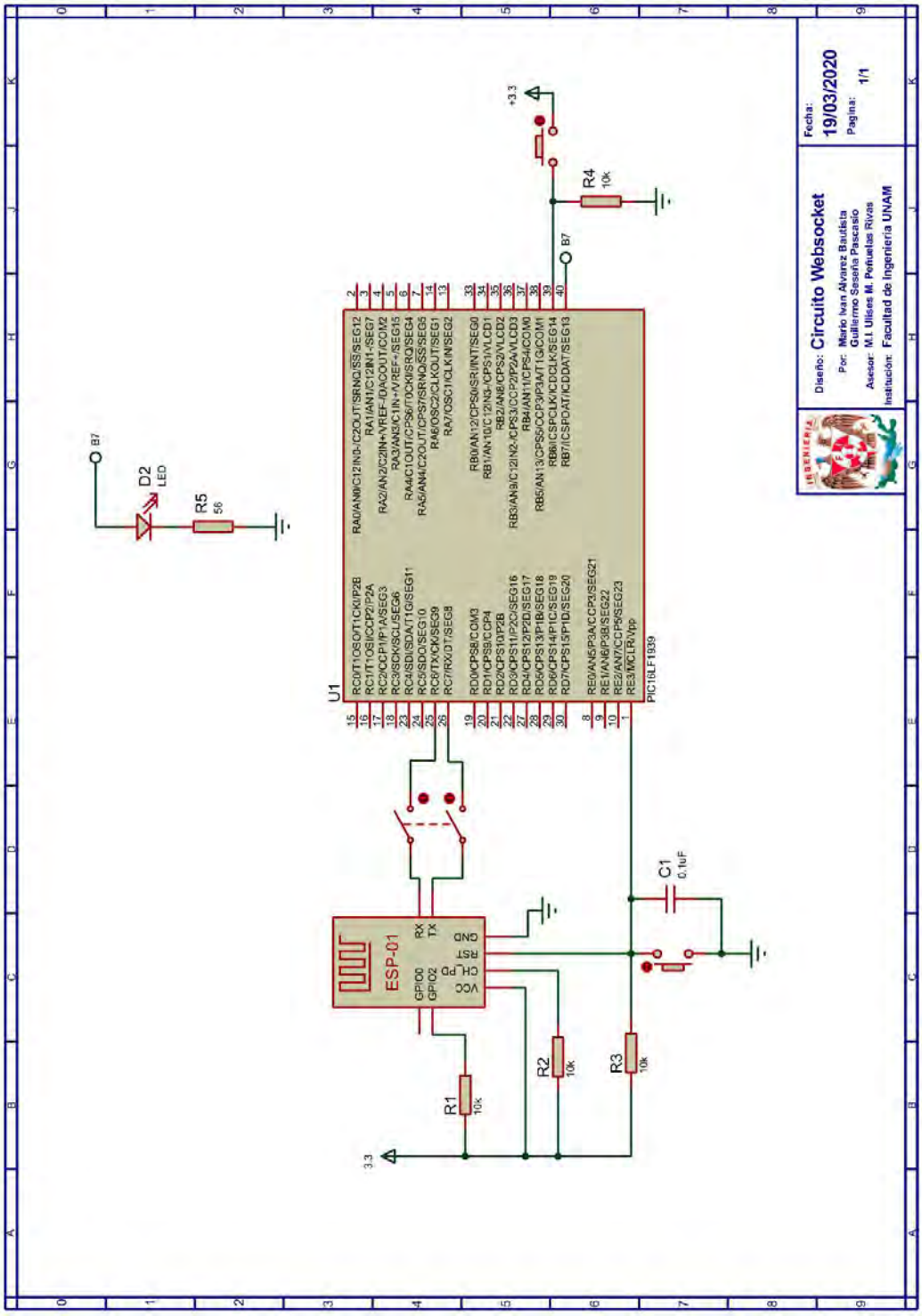


Figura P20.12 *Circuito de la práctica 20.*

Referencias

1. Internet Engineering Task Force. *The WebSocket Protocol*. 2011 [Citado 2020 Marzo]; Disponible en:https://tools.ietf.org/html/rfc6455.
2. Colaboradores de MDN. *WebSocket*. [Citado 2020 Marzo]; Disponible en:https://developer.mozilla.org/es/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications.

Práctica 21-A MDNS: reclamar un hostname

Introducción

DNS de multidifusión *(Multicast* DNS, MDNS) es un protocolo creado para traducir nombres únicos de dispositivos (*hostnames)* que terminen con *.local* en direcciones IP en redes locales que no cuentan con un servidor DNS, para esto cuando un dispositivo requiere resolver un *hostname* envía un mensaje de consulta a una dirección IP *multicast* (224.0.0.251) y puerto (5353) previamente acordado por los participantes. Este mensaje de consulta llegará a todos los dispositivos de la red local y cuando uno de ellos observa su propio nombre en el mensaje de consulta, éste responderá a la misma dirección con un mensaje que incluye su dirección IP y un Tiempo de vida (*Time to Live,* TTL), que indica cuantos segundos debe mantener la información el dispositivo que realizó la consulta, en su memoria cache. Esta información también es utilizada por los demás dispositivos para actualizar su información.

El proceso para reclamar un *hostname* se da en dos etapas, la primera, conocida como sondeo (*probing),* envía mensajes de tipo consulta con el *hostname* deseado para verificar que este nombre es único, lo cual se comprueba al no recibir respuestas. La segunda etapa conocida como anuncio (*announcing),* se envían mensajes de tipo respuesta a todos los dispositivos participantes para que almacenen el nombre y quien lo reclama. Cuando el dispositivo desea abandonar el grupo MDNS, envía un mensaje tipo respuesta con su información y un TTL con valor de cero, esto hace que los dispositivos que reciben el mensaje eliminen la información del dispositivo de su memoria cache[1].

Generalmente, MDNS no viene incorporado de manera nativa en todos los dispositivos, por lo que se debe instalar en las computadoras:

- Bonjour de apple en computadoras con sistema operativo Windows.
- Bonjour de apple en computadoras con sistema operativo Mac OSX.
- Avahi en computadoras con sistema operativo Linux.

El *firmware* SDK NONOS de los módulos que incorporan el SoC ESP8266 fue modificado el para principalmente llevar a cabo las etapas que MDNS implementa para reclamar un *hostname* con la finalidad de evitar un conflicto en la red, leer las diferentes respuestas contenidas en un solo paquete, establecer un TTL personalizado y se añadió la posibilidad de realizar una consulta MDNS para obtener la dirección IP de un dispositivo,

únicamente cuando ésta es versión IPv4. En el caso de del *firmware* ESP-IDF algunas de estas características ya están implementadas.

La biblioteca ESP permite realizar las siguientes acciones:

- Reclamar un *hostname.*
- Anunciar hasta dos servicios con dos registros TXT.
- Obtener la dirección IP de un *hostname.*

Objetivo

Mostrar como reclamar un *hostname* y obtener la dirección IP de uno.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_MDNS.

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 adaptador USB-TTL
- 1 módulo WiFi
- 1 interruptor DPST
- 4 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 1 condensador de 0.1 µF

Descripción

En esta práctica se realizarán dos actividades:

- Reclamar un *hostname.*
- Obtener dirección IP de un *hostname.*

I Reclamar un *hostname*

Se deberá conectar una computadora y el módulo WiFi a un mismo punto de acceso. En el módulo se deberá reclamar el *hostname* esp.local con un TTL de 500 (sólo en SDK NONOS es posible establecer el TTL personalizado). En la computadora, se deberá abrir el símbolo de sistema de Windows o la terminal en Linux y hacer un ping a esp.local para comprobar que fue reclamado. Posteriormente si el ping fue realizado, se presionará el botón asociado al microcontrolador para que el módulo deje el grupo MDNS notificando a los participantes. Después, se debe hacer nuevamente un ping a esp.local y esta vez, la terminal del símbolo de sistema

de windows debe notificar que no se ha encontrado el *host* destino.

II Obtener la dirección IP de un *hostname*

Para esta actividad se deberá reclamar un *hostname* para la computadora. Una vez hecho esto, se debe modificar el programa del uC PIC para obtener la dirección IP del *hostname* de la computadora al presionar un botón y la deberá imprimir en una terminal. Finalmente se debe comprobar que la dirección mostrada en la terminal corresponda con la dirección IP asignada por el punto de acceso a la computadora.

Código

El código que deberá ejecutar el microcontrolador PIC para la primera actividad se muestra en la figura P21-A.4, en el que se debe ingresar nombre y contraseña de un AP

Mientras que el código para la segunda actividad se muestra en la figura P21-A.5, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP.
- *Hostname* de la computadora a obtener su dirección IP.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P21-A.6.

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC para la primera parte. Se debe abrir el Simbolo de sistema de windows para hacer un ping a esp.local, si el módulo ha reclamado dicho *hostname* se obtendrá el resultado mostrado en la figura P21-A.1.

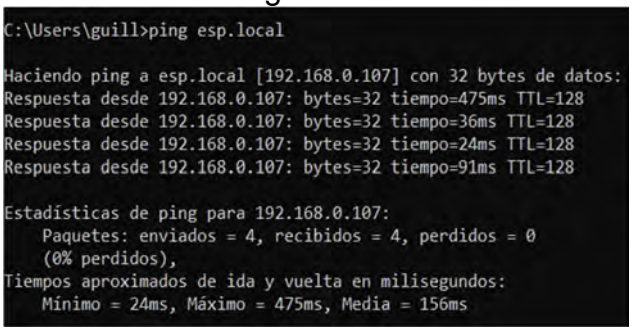


Figura P21-A.1 *Ping exitoso a esp.local.*

Posteriormente se debe presionar el botón asociado al microcontrolador para que el módulo abandone el grupo MDNS. Se realizará nuevamente un ping y se obtendrá el resultado mostrado en la figura P21-A.2, lo que lo que comprueba que el módulo ha abandonado el grupo MDNS.

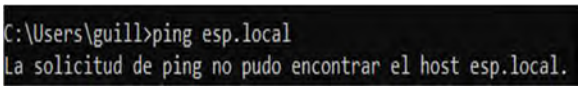


Figura P21-A.2 *Ping no exitoso de esp.local.*

Se deberá reclamar un *hostname* para la computadora para la segunda parte de esta práctica. Generalmente, cuando se instala MDNS, éste toma como el *hostname* por defecto el nombre del equipo, el cual se puede consultar en

"*Acerca de tu PC"* en la sección especificaciones de tu dispositivo (ver figura P21-A.3). Para obtener la IP que le fue asignada a la computadora se debe abrir el *Command prompt* y ejecutar *ipconfig,* la dirección IP corresponderá a la especificada en IPV4.

Figura P21-A.3 *Hostname de la PC.*

Se deberá cambiar el código del microcontrolador PIC para la segunda parte de la práctica especificando el *hostname* de la computadora. Una vez que en la terminal se observe el mensaje *MDNS ready,* se presionará el botón asociado a éste para obtener la dirección IP de la computadora mediante una consulta MDNS, dicha dirección será impresa en la terminal y se deberá corroborar que coincide con la obtenida previamente.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3dkWpGq

Notas

- Se recomienda usar el programa Hterm para simular la terminal.
- En el circuito de esta práctica, los pines utilizados para la conexión del adaptador USB-TLL son los establecidos por defecto para el segundo serial por *software* del uC PIC.
- MDNS funciona únicamente para direcciones IPv4.
- Para la versión IDF pueden cambiar algunos parámetros de las funciones, pero no permite notificar a los participantes de sus salida del grupo.

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","SSID","password");
    /*Se reclama un hostaname con un TTL de 500
    y se une al grupo MDNS*/
    start_mdns(500,"esp");
    while(1)
    {
        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta dejar de presionar el botón
            }
            // El módulo deja el grupo MDNS
            stop_mdns(1);
        }
    }
}
```

Figura P21-A.4 Código de la práctica 21-A, actividad 1.

```
#include <16lf1939.h>
#include "esp.h"

char IP[16]={0}; // dirección IP
long long TTL=0; // valor del TTL
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","SSID","password");
    /*Se reclama un hostaname con un TTL de 500
    y se une al grupo MDNS*/
    start_mdns(500,"esp");
    fprintf(PIC,"MDNS ready\r\n");
    while(1)
    {
        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                //Se mantiene hasta dejar de presionar el botón
            }
            /*realiza una consulta MDNS para obtener la dirección IP
             del hostname  especificado  (hostname.local)*/
            query_mdns("LAPTOP-BQTUIMVM",IP, sizeof(IP), &TTL);
            delay_ms(100);
            fprintf(PIC,"Diección IP:%s\r\n"IP);
        }
    }
}
```

Figura P21-A.5 Código de la práctica 21-A, actividad 2.

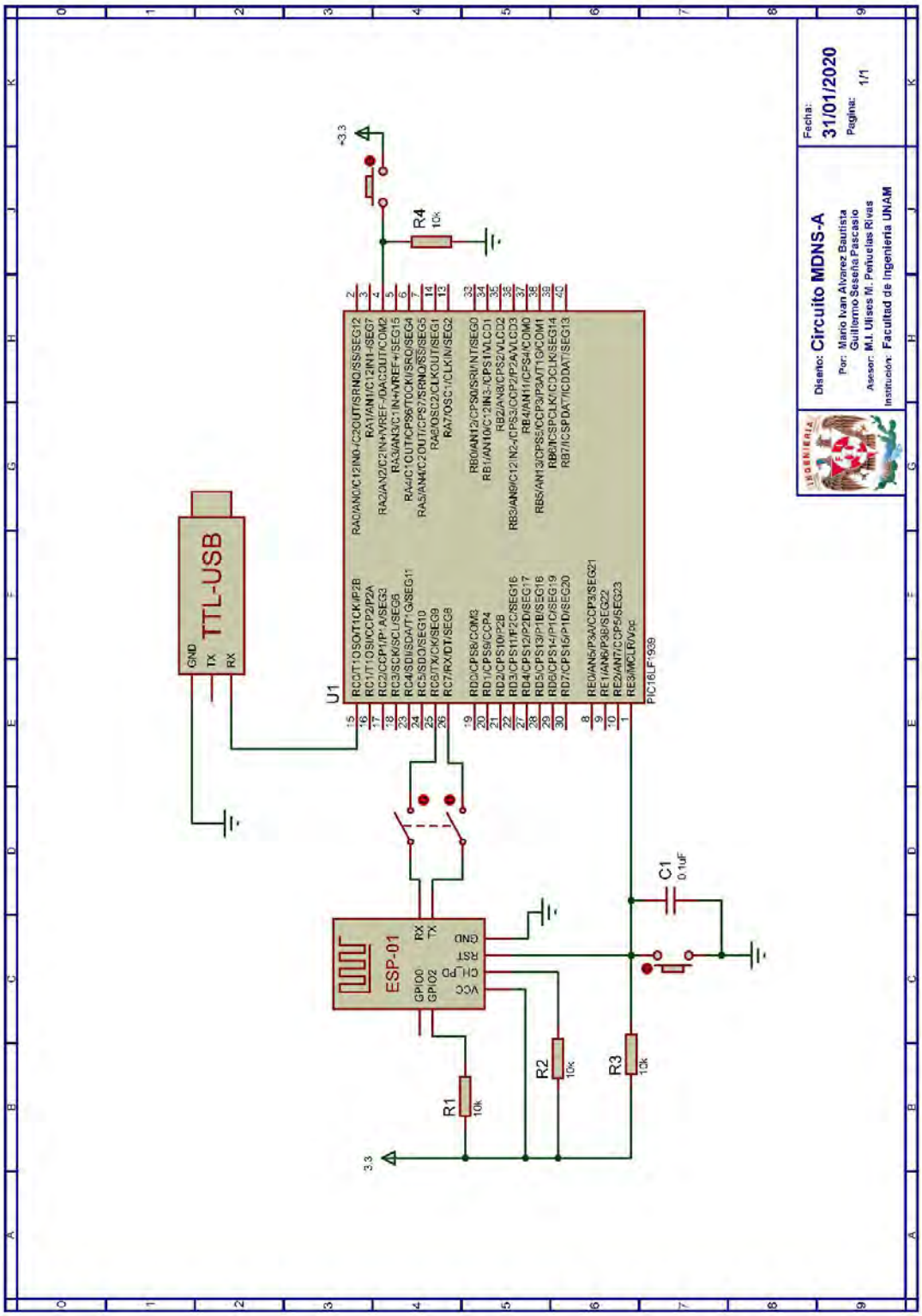


Figura P21-A.6 *Circuito práctica 21-A.*

Referencias

1. Cheshire, S. and D. Steinberg, *Zero Configuration Networking: The Definitive Guide*, ed. D.G. Series. 2006: O'Reilly Media.

Práctica 21-B MDNS: anunciar un servicio MDNS

Introducción

DNS de multidifusión *(Multicast* DNS, MDNS) es un protocolo creado para traducir nombres únicos de dispositivos (*hostnames)* que terminen con *.local* en direcciones IP en redes locales que no cuentan con un servidor DNS, para esto cuando un dispositivo requiere resolver un *hostname* envía un mensaje de consulta a una dirección IP *multicast* (224.0.0.251) y puerto (5353) previamente acordado por los participantes. Este mensaje de consulta llegará a todos los dispositivos de la red local y cuando uno de ellos observa su propio nombre en el mensaje de consulta, éste responderá a la misma dirección con un mensaje que incluye su dirección IP y un Tiempo de vida (*Time to Live,* TTL), que indica cuantos segundos debe mantener la información el dispositivo que realizó la consulta, en su memoria cache. Esta información también es utilizada por los demás dispositivos para actualizar su información.

El proceso para reclamar un *hostname* se da en dos etapas, la primera, conocida como sondeo (*probing),* envía mensajes de tipo consulta con el *hostname* deseado para verificar que este nombre es único, lo cual se comprueba al no recibir respuestas. La segunda etapa conocida como anuncio (*announcing),* se envían mensajes de tipo respuesta a todos los dispositivos participantes para que almacenen el nombre y quien lo reclama.[1]. Cuando el dispositivo desea abandonar el grupo MDNS, envía un mensaje tipo respuesta con su información y un TTL con valor de cero, esto hace que los dispositivos que reciben el mensaje eliminen la información del dispositivo de su memoria cache[1].

Generalmente, MDNS no viene incorporado de manera nativa en todos los dispositivos, por lo que se debe instalar en las computadoras:

- Bonjour de apple en computadoras con sistema operativo Windows.
- Bonjour de apple en computadoras con sistema operativo Mac OSX.
- Avahi en computadoras con sistema operativo Linux.

Como la dirección IP puede cambiar a lo largo del tiempo, pero el *hostname* MDNS generalmente no lo hará, este último puede ser utilizado para acceder u ofrecer información sobre a servicios disponibles en la red local que estén siendo ofrecidos por los dispositivos que reclaman un *hostname,* sin tener que saber la dirección IP de éste para acceder. El mecanismo que permite saber qué servicios están disponibles en

la red local se llama Descubrimiento de servicio DNS (*DNS Service Discovery,* DNS-SD), el cual enumera los servicios disponibles en la red, su dirección IP y otra información que sea necesaria para conectarse y usarlos.

En muchos casos, para acceder a un servicio sólo se necesita el *hostname* o la dirección IP de donde reside el servicio y el número del puerto, pero también es posible añadir más información como parámetros o atributos que sean de interés para el usuario mediante los denominados registros TXT (TXT r*ecord)* en una estructura *clave=valor.* El nombre del servicio anunciado generalmente viene acompañado con *_tcp* que sugiere que el servicio se ejecuta en una conexión TCP [1], que es el caso que se da en los módulos que implementan el SoC ESP8266 y los ESP32. Así mismo, estos módulos pueden anunciar hasta dos servicios, cada uno con hasta dos registros TXT, sin embargo, estos servicios deben ser anunciados en puertos diferentes.

La biblioteca ESP permite realizar las siguientes acciones:

- Reclamar un *hostname.*
- Anunciar hasta dos servicios con dos registros TXT.
- Obtener la dirección IP de un *hostname.*

Objetivo

Mostrar como anunciar un servicio y reconocer su utilidad.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION, E_WEB_SERVER y E_MDNS deben ser TRUE en esp.h

Materiales

- 1 teléfono inteligente.
- 1 computadora.
- 1 microcontrolador PIC.
- 1 módulo WiFi.
- 1 interruptor DPST.
- 3 resistores de 10 kΩ.
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF.
- 1 resistor de 56 Ω.
- 1 LED de 2.2V y 20 *mA.*

Descripción

Se deberá conectar una computadora y el módulo WiFi a un mismo punto de acceso. En el módulo se deberá crear un servidor WEB en el puerto 80, el cual al recibir el mensaje ON encenderá un LED y al recibir el mensaje OFF lo apagará. Este LED comenzará estando encendido. Así mismo, se deberá reclamar el *hostname* con un TTL de 800 (sólo en SDK NONOS es posible personalizar el TTL) y anunciar un servicio llamado luces en el puerto 80 con Encender=/ON y Apagar=/OFF como registros TXT.

Para hacer uso de MDNS, se deberá abrir un navegador web, se recomienda utilizar Google Chrome, y en la barra de direcciones se escribir:

- esp.local:80/ON
- esp.local:80/OFF

Finalmente, se debe observar el comportamiento del LED.

Código

El código que deberá ejecutar el microcontrolador PIC se muestra en la figura P21-B.2, en el que se debe ingresar nombre y contraseña de un AP.

Circuito

El circuito para llevar a cabo esta práctica se muestra figura P21-B.3.

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC y observar que el LED está encendido, se deberá abrir un navegador web y en la barra de direcciones escribir:

- esp.local:80/ON
- esp.local:80/OFF

para controlar el estado del LED.

Para descubrir que el servicio se está anunciando, se utilizará un teléfono inteligente que deberá tener instalado una aplicación que cuente con DNS-SD. En este caso se recomienda utilizar la aplicación *Service Browse*r desarrollada por Andriy Druk. En esta aplicación se debe observar el servicio que se ha creado (ver figura P21-B.1) y al hacer clic sobre éste se observará toda su información (ver figura P21-B.4).

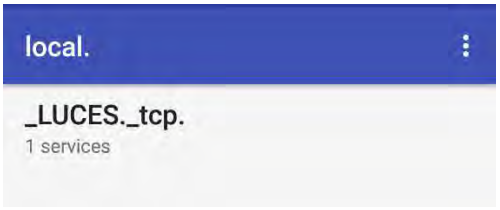


Figura P21-B.1 *Servicio anunciado.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/30XNFUf

```
#include <16lf1939.h>
#include "esp.h"

int resource=0;
char server_buffer[6]={0};
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","SSID","123456789");
    //Se establece el buffer del servidor para recibir mensajes
    set_buffer_server1(server_buffer,sizeof(server_buffer));
    // Se crea un servidor WEB en el puerto 80
    create_http_server(80,4,4000,&recurso);
    /*Se configura los registros TXT del primer servicio a ser anunciado
    el cual será accesible en el puerto del primer servidor TCP*/
    setup_service1_mdns("LUCES","Encender=/ON","Apagar=/OFF");
    /*Se reclama un hostaname con un TTL de 800
    y se une al grupo MDNS*/
    start_mdns(800,"esp");
    while(1)
    {
        http_upgrade_resource();
        switch (resource)
        {
            default:
            http_default_resource();
            break;
        }
        /*Conforme al mensaje recibido se controla el estado del LED*/
        if(1==compare_S1buffer("ON",1))
        {
            output_bit(PIN_A2,TRUE);
        }
        if(1==compare_S1buffer("OFF",1))
        {
            output_bit(PIN_A2,FALSE);
        }
        delay_ms(100);
    }
}
```

Figura P21-B.2 *Código de la práctica 21-B.*

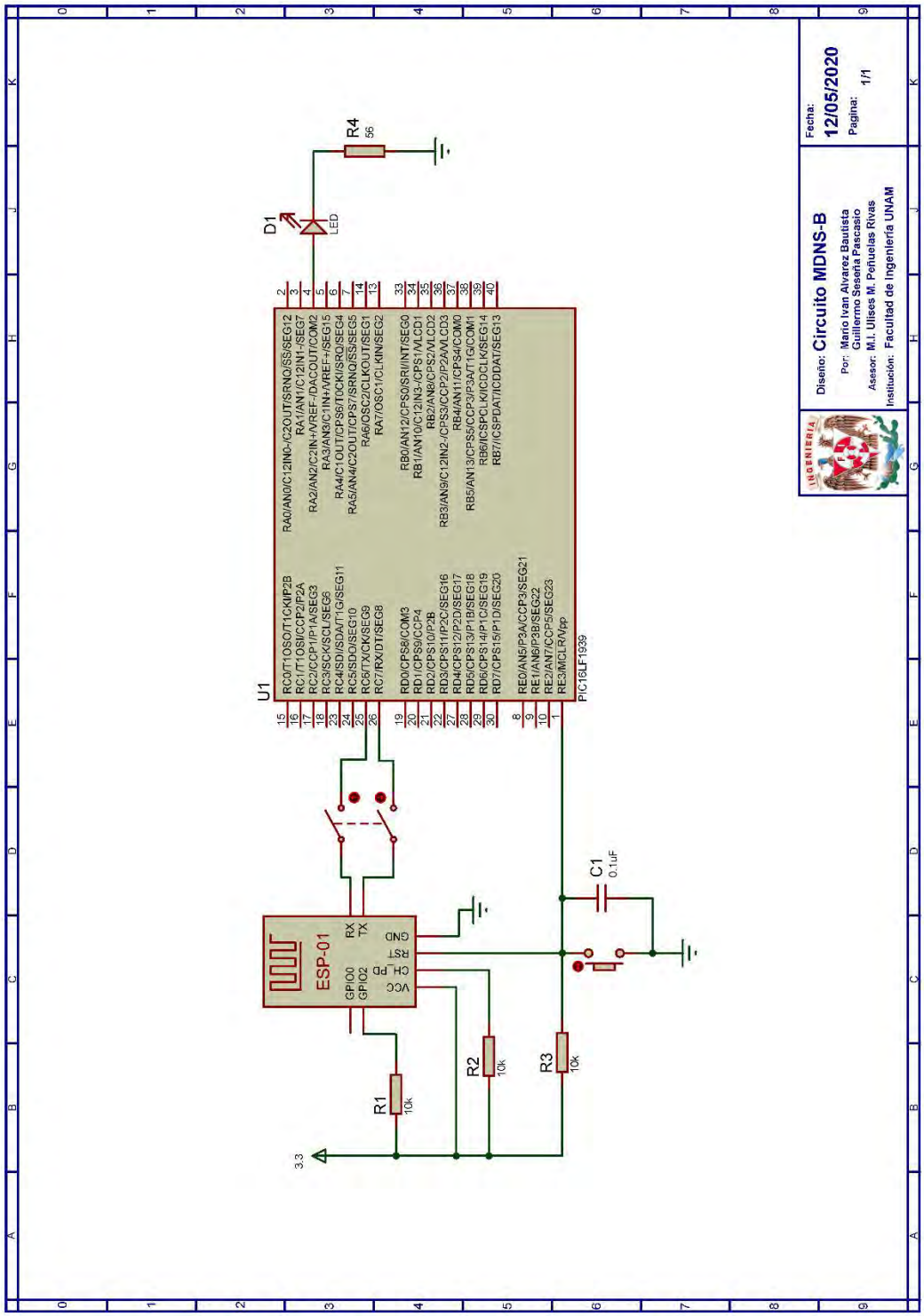


Figura P21-B.3 *Circuito de práctica 21-B.*

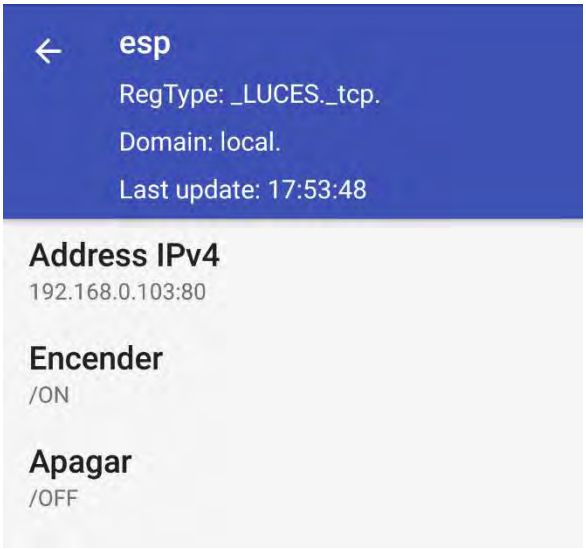


Figura P21-B.4 *Características del servicio.*

Notas

- Para la versión IDF algunos parámetros de las funciones pueden cambiar

Práctica propuesta

Utilizar MDNS para anunciar dos servicios diferentes, para lo cual se deberá hacer uso del segundo servidor.

Referencias

1. Cheshire, S. and D. Steinberg, *Zero Configuration Networking: The Definitive Guide*, ed. D.G. Series. 2006: O'Reilly Media.

Práctica 22-A APIs de Google con OAuth 2.0

Introducción

Oauth 2.0 es un estándar de autorización que proporciona flujos simples de autorización para que aplicaciones accedan a recursos web protegidos. Este estándar es ampliamente utilizado en internet por empresas como Google, Facebook, Microsoft, GitHub, Twitter y muchas más.

Oauth 2.0 evita que un usuario tenga que compartir sus credenciales (nombre de usuario y contraseña) de algún servicio en internet con una aplicación de terceros, la cual ejecuta en dicho servicio ciertas acciones en representación del usuario[1].

Para lograr lo anterior, la aplicación debe poseer un *access token* (token de acceso), que es un tipo de credencial que otorga acceso temporal y restringido a los recursos de un usuario. Un *access token* es emitido por un servidor de autorización cuando el usuario otorgar el consentimiento a la aplicación de ejecutar un conjunto de acciones (*scope*) en representación de él.

OAuth 2.0 otorga *access tokens* de acuerdo con diversos tipos de concesiones (*grant types*), que a partir de éstos proporciona distintos flujos de

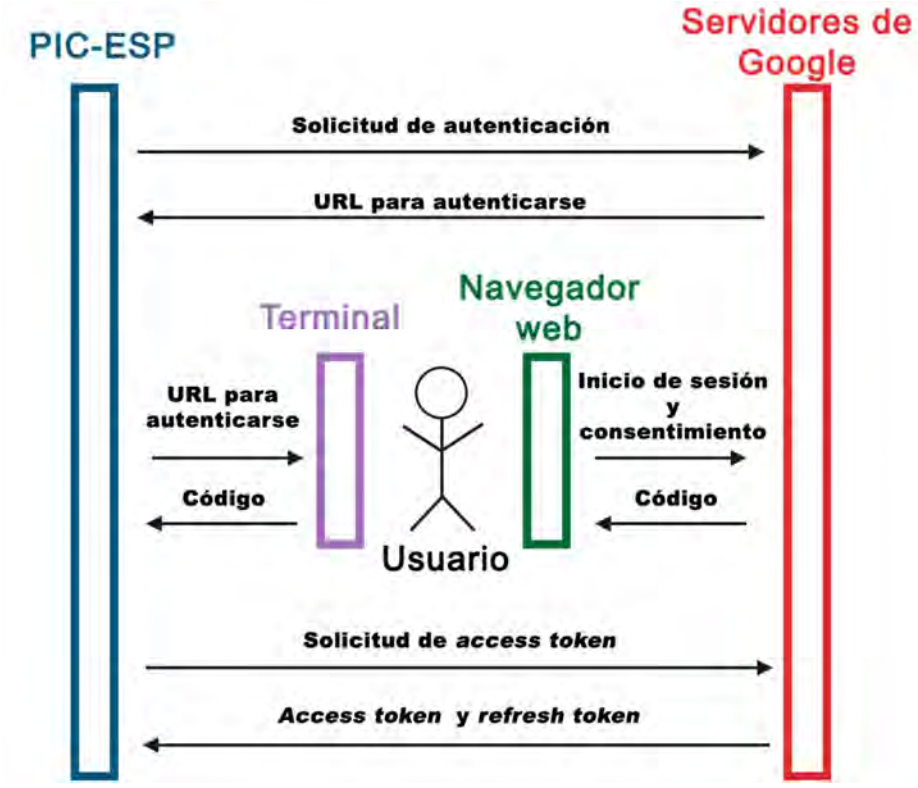


Figura P22-A.1 *Flujo de autorización utilizado en la Biblioteca ESP.*

autorización para atender a la diversidad de escenarios que se presentan al desarrollar aplicaciones móviles, de escritorio, de navegación web e incluso de dispositivos IoT.

Como se ha ilustrado en prácticas anteriores, la biblioteca ESP permite que un módulo WiFi sea establecido como un cliente HTTP para consumir diversos servicios web. Cuando estos servicios requieren de autorización mediante OAuth 2.0, basta con obtener un *access token* de manera externa y colocarlo en el encabezado de autorización de la petición HTTP. Sin embargo, los *access token* sólo son válidos durante un corto periodo de tiempo, por lo que este método resulta impráctico cuando se solicitan recursos durante largos periodos de tiempo, ya que el *access token* debe ser reemplazado por uno nuevo, lo que implica modificar el programa que ejecuta el microcontrolador PIC.

Teniendo en cuenta lo anterior, en la biblioteca ESP se han desarrollado dos funciones para obtener y refrescar *access tokens*, sin embargo, éstas están acotadas para las APIs de Google. Esta elección se debió a que las APIs de Google cuentan con un gran catálogo de opciones con las que se pueden realizar distintas aplicaciones aunado a que Google implementa algunos flujos de autorización destinados a dispositivos con reducidas capacidades computacionales e interactivas, como es el caso del conjunto que se propone en este trabajo.

Tabla P22-A.1 *Ejecución de peticiones HTTP involucradas en el flujo de autorización.*

Petición HTTP	**Respuesta HTTP**
Solicitud de autentificación	
GET /o/oauth2/v2/auth?client_id={ID de cliente}&response_type=code&scope={scope}/auth/spreadsheets&redirect_uri=urn:ietf:wg:oauth:2.0:oob&access_type=offline HTTP/1.1\r\n **Host: accounts.google.com:443\r\n** \r\n	HTTP/1.1 302 Found Content-Type: application/binary\r\n ... Location: {Enlace de autentificación}\r\n ...
Solicitud de *access token*	
POST /oauth2/v4/token HTTP/1.1\r\n **Host: www.googleapis.com:443\r\n** Content-Length: {longitud del contenido}\r\n **Content-Type: application/x-www-form-urlencoded\r\n** **\r\n** **code={código}&client_id={ID de cliente}&client_secret={secreto de cliente}&redirect_uri=urn:ietf:wg:oauth:2.0:oob&grant_type=authorization_code**	HTTP/1.1 200 OK\r\n Content-Type: application/json; charset=utf-8\r\n ... \r\n {"access_token": "{access token}ss tokens", "expires_in":3599, "refresh_token":"{refresh token}", "scope": "{scope}", "token_type": "Bearer"}

Figura P22-A.2 *Obtención de un access token desde OAuth 2.0 Playground.*

La primera función llamada `get_token_oauth20` ejecuta todo un flujo de autorización para obtener un *acccess token*. En este flujo, el módulo envía peticiones HTTP al servidor accounts.google.com, utilizando tanto un ID como un secreto de cliente, que son las credenciales que otorga Google a cada aplicación que se desea obtener acceso y que son generadas desde la consola de Google (el flujo implementado se ilustra en la figura P22-A.1, mientras que, la peticiones y respuestas involucradas se describe en la tabla P22-A.1). Como se requiere de la interacción del usuario, se involucra el uso de una terminal y de un navegador web para este fin.

El flujo comienza con el módulo solicitando la autentificación del usuario, en la petición incluye el ID del cliente y el *scope*. Si se ejecuta correctamente, la respuesta contendrá el encabezado *Location (*ubicación*)*, en el cual se haya el enlace que posteriormente es mostrado en la terminal. El usuario copia y pega el enlace a un navegador web de algún dispositivo, al hacerlo se direcciona al inicio de sesión, en donde coloca las credenciales de su cuenta de Google (dirección de correo y contraseña). Posteriormente la página solicita el consentimiento para permitirle al módulo realizar las acciones que especifica el *scope* (alcance). Al aceptar el servidor se muestra el código de autorización, el cual es ingresado al PIC mediante la terminal. Cuando el PIC obtenga el código, realiza la petición del *access token*, en la que incluye el código, el ID y secreto del cliente. El servidor responde con el *access token* y un *refresh token*, los cuales se almacenan en la RAM del microcontrolador PIC. Los *access tokens* de las APIs de Google expiran 60 minutos después de su obtención.

Tabla P22-A.2 *Ejecución de petición y respuesta HTTP involucradas en la obtención de un nuevo access token a partir del refresh token.*

Petición HTTP	**Respuesta HTTP**
Solicitud de nuevo *Access token*	
POST /oauth2/v4/token HTTP/1.1\r\n **Host: www.googleapis.com:443\r\n** Content-Length: {longitud del contenido} **Content-Type: application/x-www-form-urlencoded\r\n** **\r\n** **refresh_token={refresh token}&client_id={ID de cliente}&client_secret={secreto del cliente}&grant_type=refresh_token**	HTTP/1.1 200 OK\r\n Content-Type: application/json; charset=utf-8\r\n ... \r\n {"access_token": "{access token}", "expires_in": 3599, "scope": "{scope}", "token_type": "Bearer"}

La ejecución de la función descrita anteriormente puede ser prescindida si se desea obtener un a*ccess token* de manera externa, por ejemplo, accediendo a la página https://developers.google.com/oauthplayground/ se puede obtener *access tokens* para cualquiera de las APIs de Google (ver figura P22-A.2). Sin embargo, no hay forma de obtener uno nuevo durante la ejecución del código del PIC, por lo que se tiene que generar uno nuevo cada hora e ingresarlo al programa de PIC.

Aún con la función que obtiene *Access tokens* desde el PIC-ESP, sigue estando incompleta si no hay manera de obtener un nuevo *access token* durante la ejecución de programa, por eso la biblioteca ha incluido una función llamada `refresh_token_oauth20` para obtener un nuevo *access token* a partir del *refresh token*.

El proceso para obtener un nuevo *access token* ya no requiere que el usuario otorgue de nuevo los permisos, ya que se especificó durante el flujo de autorización que el acceso es fuera de línea (*offline*), indicando que no requiere que el usuario esté presente para actualizar el *access token*. Por esto, la función sólo ejecuta una petición HTTP al servidor accounts.google.com, la cual incluye al *refresh token*, el ID y secreto del cliente. La petición y respuesta utilizada se muestra en la tabla P22-A.2. Es importante mencionar que un *refresh token* puede actualizar hasta un máximo de 50 *access tokens* y es válido por solo 6 meses.

La desventaja que supone obtener un *access token* con la función que ejecuta el flujo descrito, es que siempre se necesita que el usuario otorgue el consentimiento, que quizá no sea practico para todas las aplicaciones. Lo que se puede realizar ante esta situación, es ejecutar primero un programa en el que sólo se obtenga el *access token* y el *refresh token*, posteriormente éstos se incluyen en código de la aplicación y se indica que la función para obtener un nuevo *access token* se ejecute cuando el servidor comience a responder que la petición no está autorizada (Código de estado HTTP igual a 401) o que se ejecute cada cierto tiempo para evitar usar un *access token* expirado.

Objetivo

Obtener un *access token* para ingresar a una de las API de Google.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h

- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_GOOGLE_OAUTH20

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 4 resistores de 10 kΩ
- 2 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

En esta práctica, el módulo ESP obtendrá un *access token* para la API de hojas de cálculo de Google (Google *Sheets API*) mediante la ejecución del flujo de autorización implementando en la biblioteca ESP. Posteriormente con un botón actualizará el *access token* a través del *refresh token.*

Para esta práctica se requiere de una cuenta en Google. Lo primero que se debe realizar es la obtención de las credenciales de la aplicación. Para crear dichas credenciales, se accede a la página de https://console.developers.google.com/apis que pedirá ingresar a una cuenta de Google. Posteriormente aparecerá un mensaje de bienvenida a *Google Cloud*

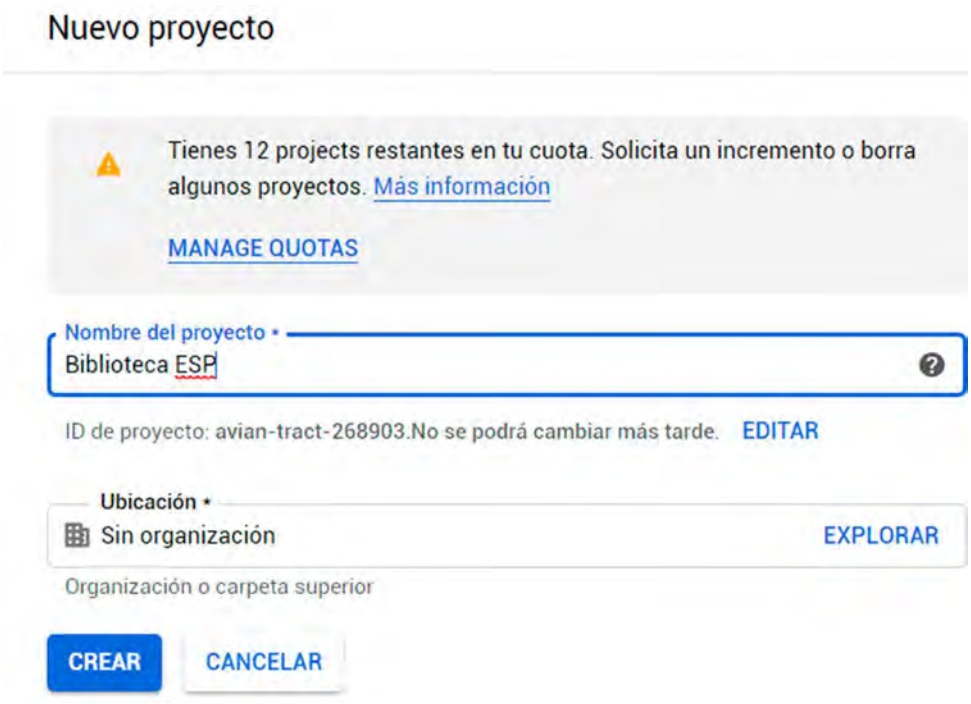


Figura P22-A.3 *Creación de nuevo proyecto en Google APIs.*

Platform en el que se pide leer y aceptar las condiciones de uso. Una vez aceptadas, se debe crear el primer proyecto agregando un nombre y ubicación a éste, aunque basta con sólo especificar el nombre (ver figura P22-A.3).

Una vez creado el proyecto se deberá configurar la pantalla de consentimiento, la cual aparecerá cuando se le pida al usuario aceptar los permisos que solicitará el módulo ESP, para esta práctica sólo será necesario configurar el nombre de la aplicación (ver figura P22-A.4). Después de configurar la pantalla de consentimiento, en el buscador de APIs y servicios, se busca Google Sheets API y se habilita (ver figura P22-A.5).

Luego se accede al apartado de credenciales, se presiona Crear credenciales e ID de cliente OAuth, la página solicitará el nombre y tipo de la aplicación. Como tipo se selecciona "*Otro*" (ver figura P22-A.6), es posible

Figura P22-A.4 *Configuración de pantalla de consentimiento Oauth.*

que no aparezca esta opción, en ese caso se selecciona "*Televisiones y dispositivos con posibilidades de interacción limitada*". Al presionar "*Crear*", aparece una ventana que proporciona tanto el ID como el secreto del cliente (ver figura P22-A.7).

El módulo obtendrá el *access token* utilizando las credenciales que se acaban de obtener y como *scope* se utilizará:

https://www.googleapis.com/auth/spreadsheets .

El tamaño del *access token* y del *refresh token* es variable, pueden alcanzar hasta 2048 y 512 bytes respectivamente. Antes de especificar los tamaños en el código del microcontrolador PIC, se recomienda acceder a OAuth 2.0 Playground para comprobar el tamaño que se esté implementando en ese momento. Para más información de AOuth 2.0 y Google se recomienda visitar la página

https://developers.google.com/identity/protocols/oauth2

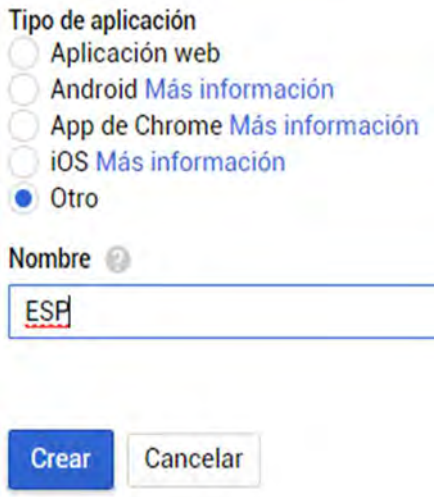


Figura P22-A.6 *Selección de tipo de aplicación.*

Figura P22-A.5 *Habilitación de Google Sheets API.*

Figura P22-A.7 Ejemplo de ID y secreto de cliente.

Código

En el archivo esp.h, en la zona de OAuth20 se colocan los siguientes parámetros (ver figura P22-A.16):

- ID de cliente
- Secreto del cliente
- Scope

El código que deberá ejecutar el microcontrolador PIC para esta práctica se muestra en la figura P22-A.18, en el que se ingresa nombre y contraseña de un AP con acceso a internet.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P22-A.19.

Resultados

Al iniciar, en la terminal se observará el encabezado "*Location*", el cual contiene el enlace para autentificar la cuenta de Google (ver figura P22-A.8). Dicho enlace deberá ser copiado y accedido en un navegador web. La página redireccionará a un inicio de sesión, si hay sesión abierta en ese momento, sólo bastará con seleccionar la cuenta, en caso contrario, se ingresará las credenciales de la cuenta, que son la dirección de correo electrónico y su contraseña (ver figura P22-A.9).

En posteriores solicitudes es posible que la página muestre un mensaje de advertencia, indicando que la aplicación no ha sido verificada por Google, por lo que se tiene que presionar la Configuración avanzada e ir a la aplicación (ver figura P22-A.10). Para la primera interacción no se mostrará dicha advertencia.

```
Moodulo esp listo para su uso\r\n
Location: https://accounts.google.com/signin/oauth?client_id=6
96765077946-kl3svccaunob336rb4k2hi5j1kkdp62o.apps.googleuserco
ntent.com&response_type=code&scope=https://www.googleapis.com/
auth/spreadsheets&redirect_uri=urn:ietf:wg:oauth:2.0:oob&acces
s_type=offline&o2v=2&as=wGlNxuAxAN3B55pJstwtXg\r\n
Ingrese coodigo\r\n
```

Figura P22-A.8 Enlace para autentificarse.

Posteriormente la página solicita otorgar los permisos para que la aplicación, en este caso el módulo, pueda realizar ciertas acciones en nombre del propietario de la cuenta. Para esta demostración los permisos que se otorgan son: ver, editar, crear y borrar hojas de cálculo de Google Drive (ver figura P22-A.11).

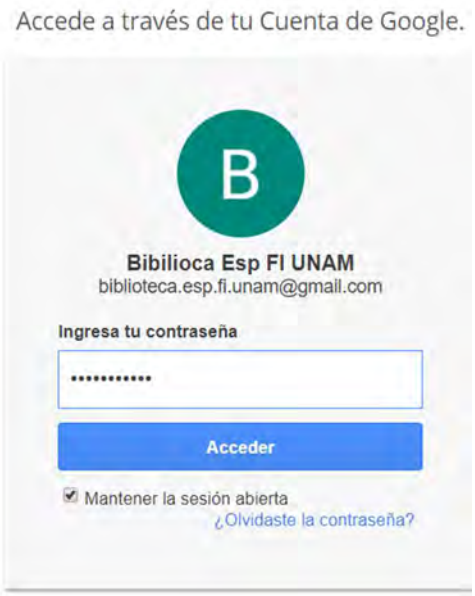


Figura P22-A.9 *Autentificación.*

Al otorgar los permisos, la página mostrará un código de aplicación (ver figura P22-A.12) el cual es copiado e introducido a la terminal (ver figura P22-A.14). Después de ingresar dicho código se mostrará en pantalla un *access token* que tiene validez de 60 minutos y un *refresh token* para generar nuevos *tokens* válidos (ver figura P22-A.15).

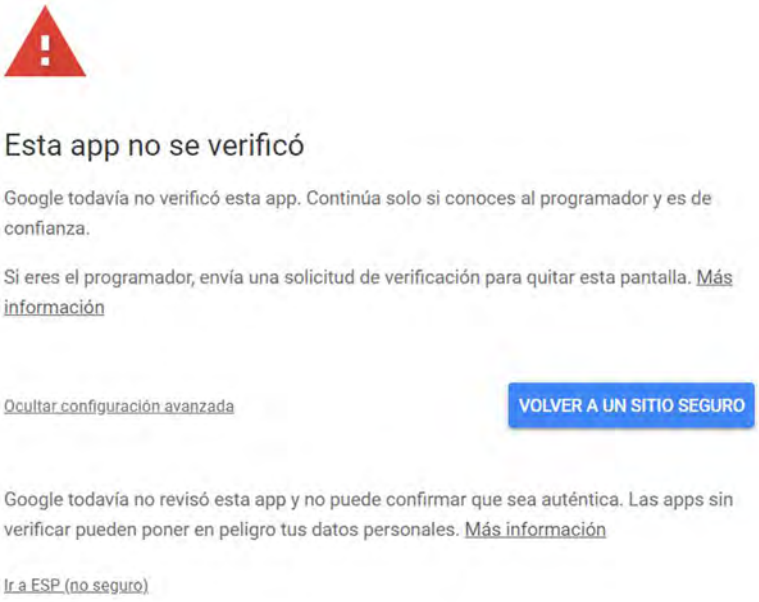


Figura P22-A.10 *Advertencia que indica que Google no ha verificado la aplicación.*

Figura P22-A.11 *Otorgamiento de permisos para utilizar la API de hojas de cálculo.*

Finalmente se presiona el botón destinado a obtener un nuevo *access token* a partir del *refreh token*, cada vez que se solicite uno nuevo se mostrará en la terminal (ver figura P22-A.13).

Por seguridad Google puede enviar un correo electrónico de alerta de seguridad, en el que pregunta, si quien accedió a la cuenta efectivamente era el propietario o es alguien ajeno que está tomando su identidad, en este caso se afirma que quien accedió fue el propietario (ver figura P22-A.17).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2zLbgfy

Figura P22-A.12 *Código de aplicación.*

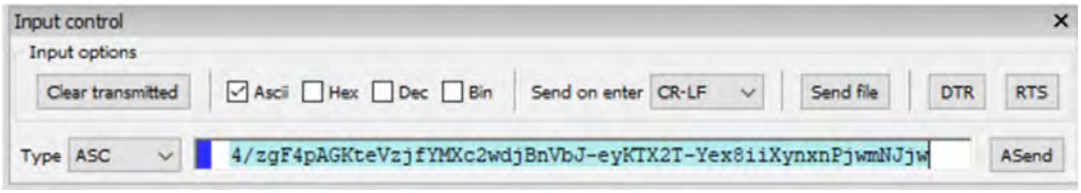


Figura P22-A.14 *Ingreso del código a la terminal.*

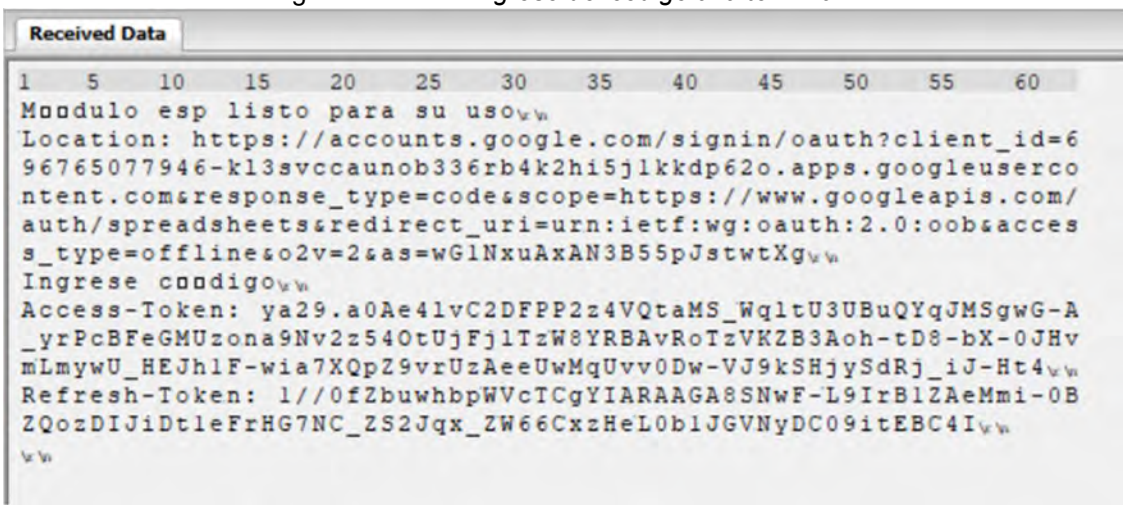


Figura P22-A.15 *Obtención de access token y refresh token.*

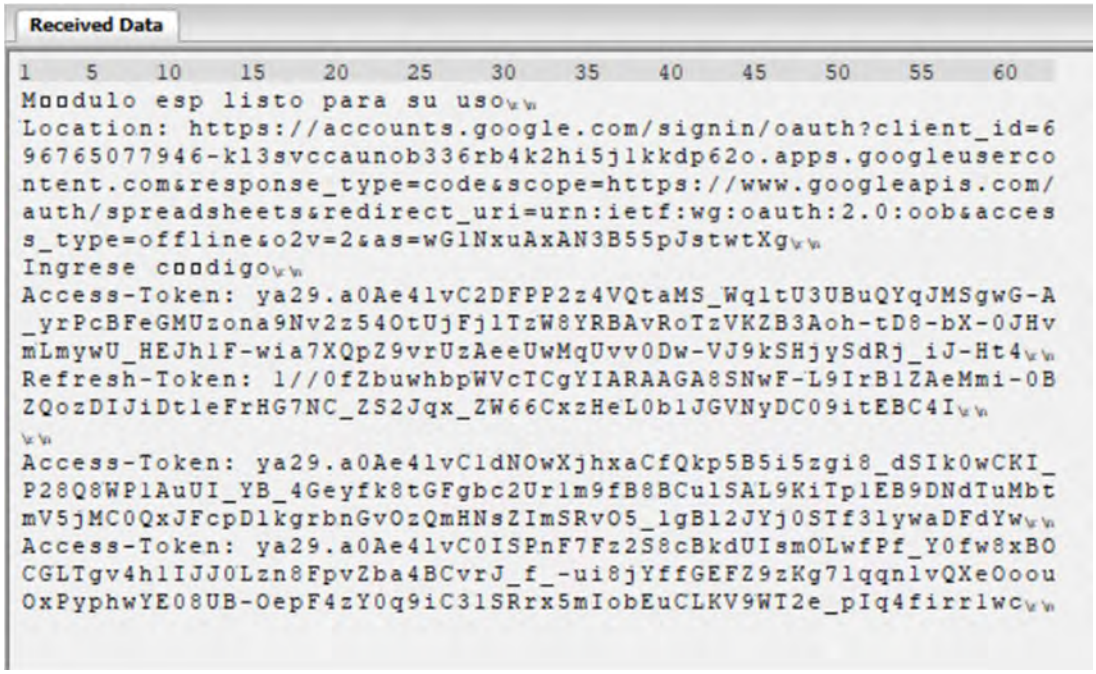


Figura P22-A.13 *Obtención de nuevo access token a partir del refresh token.*

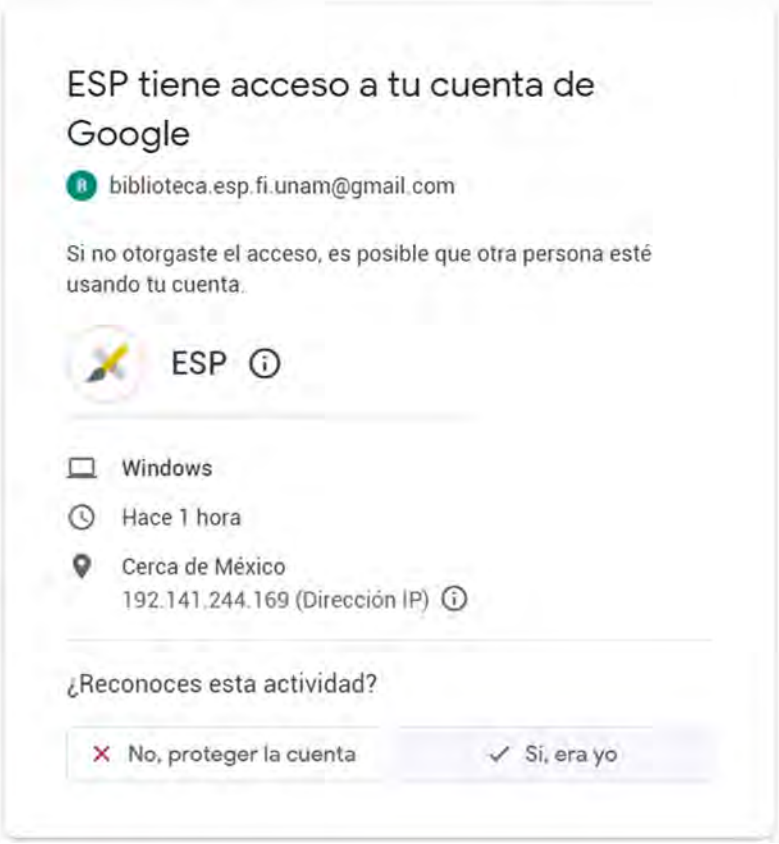


Figura P22-A.17 *Correo de Alerta de seguridad.*

```
/*************************************************************************************************
 *                                    OAUTH2.0
*************************************************************************************************
*/
#define CLIENT_ID_OAUTH20 "ID de usuario"
#define CLIEND_SECRET_OAUTH20 "Secreto de usuario"
#define SCOPE_OAUTH20 "https:\/\/www.googleapis.com/auth/spreadsheets"

#define HOST_OAUTH20 "accounts.google.com"
#define HOST_OAUTH20_ "www.googleapis.com"
#define PORT_OAUTH20 443
#define URL_OAUTH20 "/o/oauth2/v2/auth?"
#define URL_OAUTH20_ "/oauth2/v4/token"
```

Figura P22-A.16 *Zona de ingreso de parámetros para la práctica 22-A en esp.h.*

```c
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"

char access_token[172]={0};
char refresh_token[105]={0};

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Obtiene el access token y el refresh token para el id de usuario,
    secreto de usuario y scope definido en esp.h*/
    get_token_oauth20(access_token,sizeof(access_token),refresh_token,sizeof(refresh_token));
    fprintf(PIC,"Access-Token: %s\r\n",access_token);
    fprintf(PIC,"Refresh-Token: %s\r\n\r\n",refresh_token);
    //Al presionar botón solicita un nuevo access token
    while(1)
    {
        if(1==input_state(PIN_B7))
        {
            //Obtiene un nuevo access token válido
            refresh_token_oauth20(0,access_token,refresh_token);
            fprintf(PIC,"Access-Token: %s\r\n",access_token);
            while(1==input_state(PIN_B7))
            {
                //Se mantiene hasta que se suelte
            }
        }
    }
}
```

Figura P22-A.18 *Código de la práctica 22-A.*

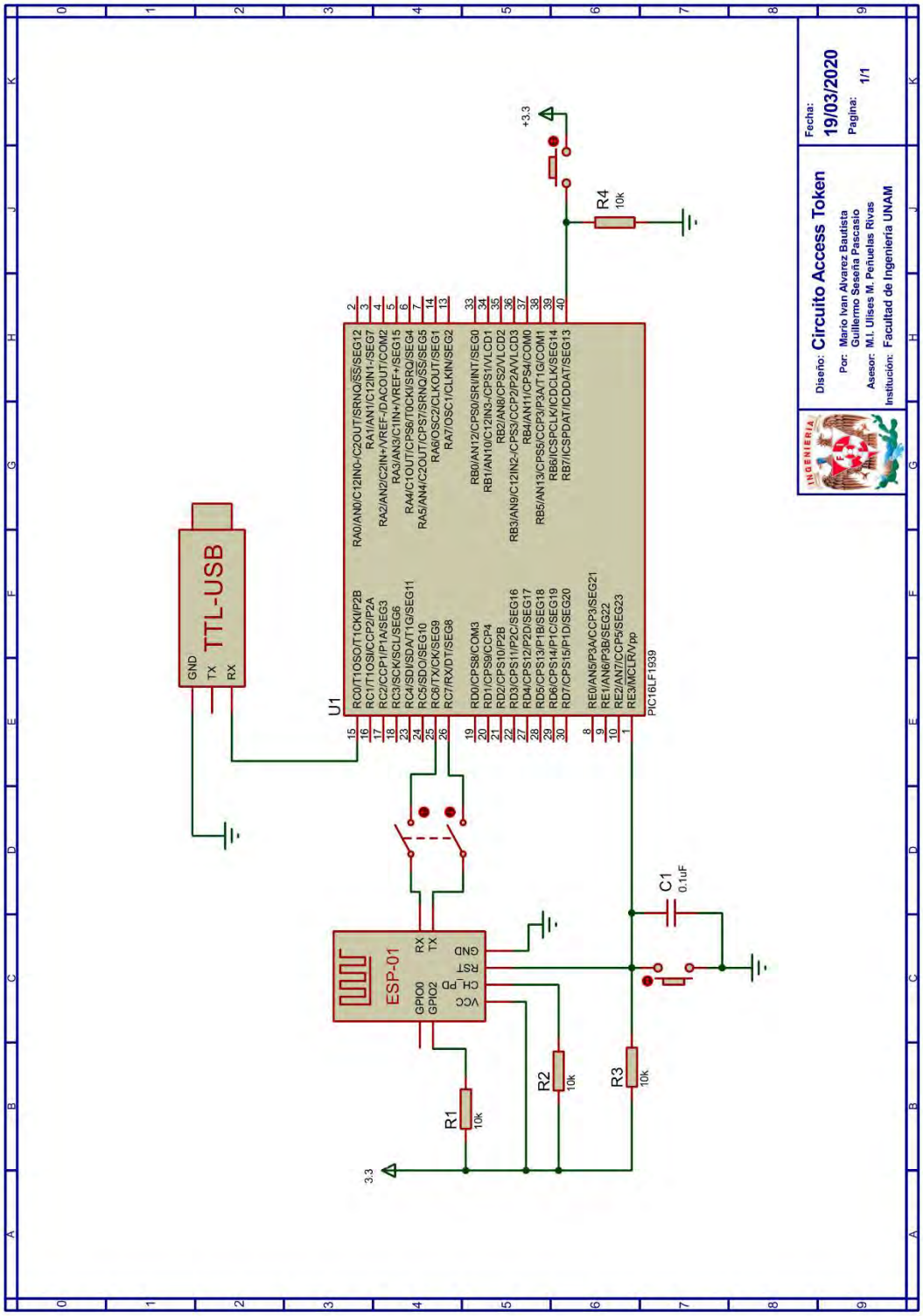


Figura P22-A.19 *Circuito de la práctica 22-A.*

Referencias

1. *OAuth 2.0* Disponible en:https://oauth.net/2/.

Práctica 22-B APIs de Google con Oauth 2.0: registro y consulta de datos de una hoja de cálculo.

Introducción

La API de hojas de cálculo de Google (Google Sheets API) permite crear, leer, modificar y eliminar hojas de cálculo. Esta API está basada en un servicio REST, por lo que los clientes la utilizan mediante peticiones HTTP. Al igual que todas las demás APIs de Google, ésta sólo puede ser accedida mediante una conexión segura.

Las peticiones que se envían a la API pueden requerir o no de autorización. Aquellas que no, están destinadas para acceder a recursos no protegidos, por ejemplo, obtener información de una hoja de cálculo pública. Para este tipo de uso, solo basta incluir en la petición una *API Key* (clave API).

Las peticiones que soliciten recursos protegidos, como lo son documentos privados, necesitan obtener un *access token*, a través de algún flujo de autorización de OAuth2.0. Para este tipo de uso, la API ha definido seis tipos de alcances (*scope*), los cuales se muestra en la tabla P22-B.1.

Como la biblioteca ESP permite al conjunto PIC-ESP establecer conexiones seguras y ejecutar libremente peticiones HTTP, éste puede funcionar como una aplicación que consuma el servicio de la API de hojas de cálculo, sin embargo, como ya se ha expresado en prácticas anteriores la biblioteca ESP no proporciona una solución general para manejar la información proveniente de las respuestas de las peticiones, por lo que su utilidad puede ser reducida a aplicaciones que no requieran solicitar grandes cantidades de información a la API.

Tabla P22-B.1 *Scopes disponibles para la Google Sheets API [1].*

Scope	Descripción
https://www.googleapis.com/auth/spreadsheets.readonly	Permite la lectura del contenido de hojas de cálculo del usuario junto con las propiedades de estos documentos.
https://www.googleapis.com/auth/spreadsheets	Permite lectura/escritura de las hojas de cálculo del usuario junto con las propiedades de estos documentos.
https://www.googleapis.com/auth/drive.readonly	Permite únicamente la lectura metadatos y el contenido de los documentos del usuario
https://www.googleapis.com/auth/drive.file	Acceso a un archivo creado o abierto por la aplicación.
https://www.googleapis.com/auth/drive	Autorización completa para acceder a todos los archivos de un usuario. Se recomienda solo cuando sea estrictamente necesario.

Para esta API en particular, la biblioteca ESP ha proporcionado una función con la que guarda la información del contenido de las celdas de alguna hoja de cálculo, está información es tomada del contenido de la respuesta, la cual se encuentra en formato JSON y es almacenada en un *string* que contiene una matriz de datos.

Objetivo

Registrar y consultar datos de una de hoja de cálculo a través de la API de hojas de cálculo de Google.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION, E_RSSI, E_HTTP_CLIENT y E_GOOGLE_OAUTH20

Materiales

- 1 computadora
  2 microcontrolador PIC
- 2 módulos WiFi
- 2 interruptor DPST
- 6 resistores de 10 kΩ
- 2 botón pulsador n.o.
- 2 condensador de 0.1 µF
- 1 pantalla LCD 16x2

Descripción

Se utilizarán dos módulos WiFi, uno configurará una hoja de cálculo de Google que obtendrá valores RSSI y los registrará en la hoja de cálculo. El otro módulo consultará la hoja de cálculo para obtener el promedio de las lecturas RSSI, esta información será mostrada en una LCD.

Esta práctica toma como punto de partida la anterior práctica (Práctica 22-A), en la cual se obtuvo un *access token* y *refresh token* para esta API. Por lo que los módulos sólo refrescarán el *access token* cuando éste expire. Lo anterior implica que para esta práctica se debe disponer de un *refresh token* válido junto con las credenciales de la aplicación (ID y secreto de cliente) también obtenido en la práctica anterior.

Además, será necesario que el usuario cree una hoja de cálculo ya que a los módulos sólo se les delegó la escritura y lectura de ésta. Al crear la hoja de cálculo, se debe acceder a ella y obtener de su enlace el ID de la hoja

(*spreadsheetId*), este ID se utilizará en las peticiones HTTP. Por ejemplo, en la figura P22-B.1 el enlace es:

https://docs.google.com/spreadsheets/d/1DLS86KxCNtk6x7fs6_xbfZddsJ-J6bmE7GQqbKae9CE/edit#gid=0

Por tanto, el ID de esta hoja de cálculo es: 1DLS86KxCNtk6x7fs6_xbfZddsJ-J6bmE7GQqbKae9CE.

De la hoja de cálculo se usará las columnas A, B y C. En la columna A, se registrarán las lecturas de los valores RSSI, cada nueva lectura se guardará en la celda siguiente de esta columna (A# número de registro). En la columna B solo se utilizará una celda (B2) en la que se guardará el nombre de la última celda que ha almacenado una lectura. Y la columna C utilizará la celda (C2), la cual calculará el promedio de las lecturas mediante la función "`=PROMEDIO(A2:INDIRECTO(B2))`", esta función obtiene el promedio del rango de valores que inicia en A2 y termina en la celda que se indica en B2.

El módulo que se encargará de establecer la configuración y de registrar las lecturas, ejecutará peticiones HTTP mediante el método PUT para actualizar el contenido de las celdas. En la primera petición que realizará dicho módulo, colocará el nombre de las columnas que se utilizan, insertará valores iniciales e introducirá la función del cálculo del promedio.

Una vez que se configure la hoja, se comenzará a registrar valores RSSI y actualizará el valor de la celda B2, para cada una de éstas se ejecuta una petición reciclando la conexión. Mientras que, el otro módulo ejecutará peticiones GET para obtener el valor de la celda C2, que contiene el promedio de las lecturas.

Las peticiones HTTP que ejecutan los módulos se muestran en la tabla P22-B.2.

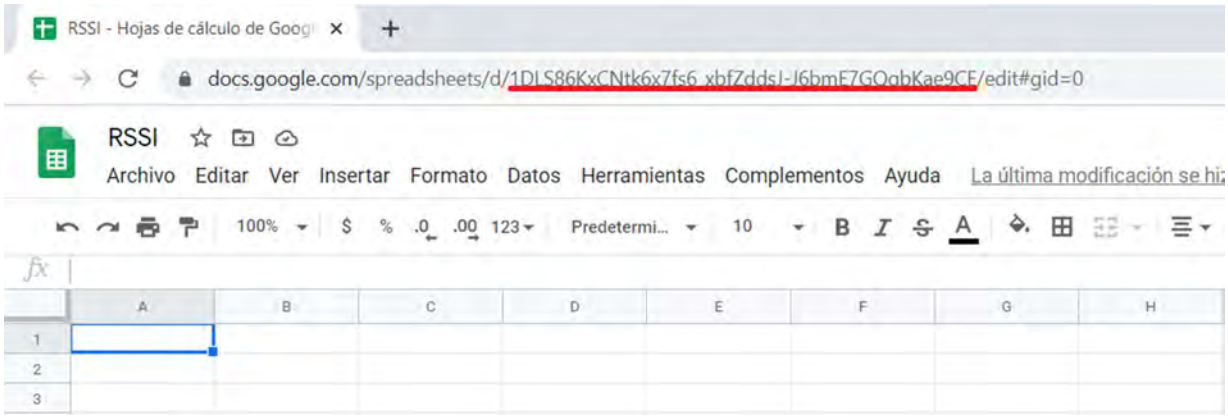


Figura P22-B.1 *SpreadsheetId contenido en el enlace de la hoja de cálculo.*

Tabla P22-B.2 *Peticiones HTTP utilizados en la práctica 22-B para registro y consulta de datos de una hoja de cálculo.*

Colocar nombre a las columnas e incluir la función del cálculo de promedio
PUT /v4/spreadsheets/{spreadsheetId}/values/Hoja%201!A1:C2?valueInputOption=USER_ENTERED HTTP/1.1\r\n Authorization: Bearer {access_token}\r\n Content-Length: 139\r\n Host: sheets.googleapis.com\r\n Content-Type: application/json\r\n \r\n {"range":"Hoja 1!A1:C2","majorDimension":"ROWS","values":[["LECTURAS","ULTIMA CELDA","PROMEDIO"],["0","A2","=PROMEDIO(A2:INDIRECTO(B2))"]]}
Registro de lecturas
PUT /v4/spreadsheets/{spreadsheetId}/values/Hoja%201!A{numero de celda}?valueInputOption=USER_ENTERED HTTP/1.1\r\n Authorization: Bearer {access_token}\r\n Connection: keep-alive\r\n **Content-Length: {longitud del contenido}\r\n** Host: sheets.googleapis.com\r\n Content-Type: application/json\r\n \r\n {"range":"Hoja 1!A{número de celda}","majorDimension":"ROWS","values":[[{valor rssi}]]}
Actualización del nombre de la última celda
PUT /v4/spreadsheets/{spreadsheetId}/values/B2?valueInputOption=USER_ENTERED HTTP/1.1\r\n Authorization: Bearer {access_token}\r\n Content-Length: {longitud del contenido}\r\n Host: sheets.googleapis.com\r\n Content-Type: application/json\r\n \r\n {"range":"B2","majorDimension":"COLUMNS","values":[["A{número de celda}"]]}
Solicitud del contenido de la celda que contiene el promedio
GET /v4/spreadsheets/{spreadsheetId}/values/C2 HTTP/1.1\r\n Authorization: Bearer {access_token}\r\n Host: sheets.googleapis.com\r\n \r\n

- Los parámetros y variables en las peticiones HTTP están resaltados en color amarillo.

Ambos módulos se conectarán al servidor googleapis.com a través del puerto 443. Al inicio de la ejecución ambos módulos actualizan el *access token*. Durante el registro y consulta de datos, los módulos actualizarán el *access token* cuando la API les indique que no están autorizados, es decir, cuando obtengan un código de estado HTTP igual a 401.

Esta práctica sólo ejemplifica como el módulo ESP puede leer y escribir una hoja de cálculo. Para saber más acerca de cómo utilizar esta API se recomienda consultar la página:

https://developers.google.com/sheets/api/samples .

Código

Para ambos microcontroladores PIC, en el archivo esp.h, en la zona de OAuth20, se deben colocar los siguientes parámetros (ver figura P22-B.4):

- ID de cliente
- Secreto del cliente

El código que deberá ejecutar el microcontrolador PIC que se encarga de registrar información, es el mostrado en la figura P22-B.5, figura P22-B.6 y figura P22-B.7, mientras para el PIC que consulta tiene que ejecutar el mostrado en la figura P22-B.8. Para cada uno de estos programas, se debe especificar los siguientes parámetros:

- Nombre y contraseña de AP con acceso a internet
- Un access token
- Un refresh token válido

Circuito

El circuito para el microcontrolador PIC que registra los valores RSSI es el que se muestra en la figura P22-B.9, mientras que el del microcontrolador PIC que consulta estos valores es el que se indica en figura P22-B.10.

Resultados

Primero se debe ejecutar el PIC-ESP encargado de configurar y registrar los valores RSSI. Lo primero que se observará en la hoja de cálculo es que se insertan los nombres de las columnas, se colocan los valores iniciales e inserta la función del cálculo del promedio (ver figura P22-B.2). Posteriormente, conforme se registran lecturas RSSI, se calculará el promedio (ver figura P22-B.3).

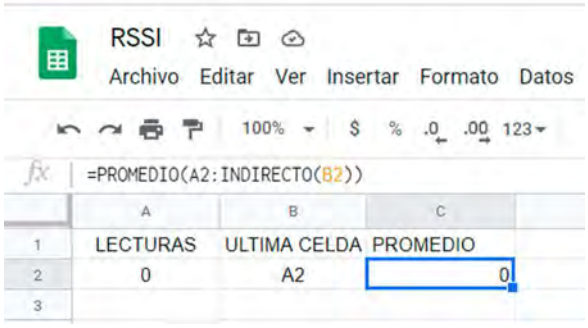


Figura P22-B.2 *El módulo inserta los nombres de las columnas y la función para calcular el promedio.*

Una vez observado lo anterior, se ejecutará el código del microcontrolador PIC que realiza la consulta del valor promedio RSSI de la hoja de datos, dicho valor se mostrará en la LCD

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2BjYG7F

	A	B	C
1	LECTURAS	ULTIMA CELDA	PROMEDIO
2	-65	A24	-66.65217391
3	-67		
4	-65		
5	-66		
6	-66		
7	-66		
8	-68		
9	-65		
10	-68		
11	-69		
12	-64		
13	-64		
14	-69		
15	-67		
16	-68		
17	-66		
18	-70		
19	-65		
20	-64		
21	-69		
22	-66		
23	-71		
24	-65		

Figura P22-B.3 *Registro de lecturas RSSI y cálculo del promedio.*

```
/*****************************************************************************************
 *                                  OAUTH2.0
*****************************************************************************************
*/
#define CLIENT_ID_OAUTH20 "ID de usuario"
#define CLIEND_SECRET_OAUTH20 "Secreto de usuario"
#define SCOPE_OAUTH20 "https:\/\/www.googleapis.com/auth/spreadsheets"

#define HOST_OAUTH20 "accounts.google.com"
#define HOST_OAUTH20_ "www.googleapis.com"
#define PORT_OAUTH20 443
#define URL_OAUTH20 "/o/oauth2/v2/auth?"
#define URL_OAUTH20_ "/oauth2/v4/token"
```

Figura P22-B.4 *Zona de ingreso de parámetros para la práctica 22-B en esp.h.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define spreadsheetId "ID de la hoja de cálculo"
char access_token[]="access token";
char refresh_token[]="refresh token";

int valor_rssi=0;
long celda=2;
long codigo_estado_http=0;
long content_length=0;
char str_content_length[4];
char json_1[]="{\"range\":\"Hoja 1!A";
char str_celda[5];
char json_2[]="\",\"majorDimension\":\"ROWS\",\"values\":[[";
char str_rssi[10];
char json_3[]="]]}";

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Actualiza access token
    refresh_token_oauth20(0,access_token,refresh_token);
    /* En el canal de comunicación 0, se establece un cliente SSL*/
    create_ssl_pclient("googleapis.com",443,0);
    /*Se envía petición HTTP para colocar las etiquetas de las columnas
    y las funciones para obtener el promedio de las lecturas*/
    send_client(0,"PUT /v4/spreadsheets/");
    send_client(0,spreadsheetId);
    send_client(0,"/values/");
    send_client(0,"Hoja%201!A1:C2");//A1:C1
    send_client(0,"?valueInputOption=USER_ENTERED HTTP/1.1\r\n");
    send_client(0,"Authorization: Bearer ");
    send_client(0,access_token);
    send_client(0,"\r\nContent-Length: 139");
    send_client(0,"\r\nHost: sheets.googleapis.com\r\nContent-Type: application/json\r\n\r\n");
    send_client(0,"{\"range\":\"Hoja 1!A1:C2\",\"majorDimension\":\"ROWS\",\"values\":");
    send_client(0,"[[\"LECTURAS\",\"ULTIMA CELDA\",\"PROMEDIO\"],[\"0\",\"A2\",");
    codigo_estado_http=finish_request_http_client(0,"\"=PROMEDIO(A2:INDIRECTO(B2))\"]]}");
    delete_client(0);
    delay_ms(5000);
    while(1)
    {
```

Figura P22-B.5 *Primera parte del código de la práctica 22-B para registro de datos.*

```
//Se obtiene valor rssi
valor_rssi=get_rssi();
// En el canal de comunicación 0, se establece un cliente SSL
create_ssl_pclient("googleapis.com",443,0);
//Se envía petición HTTP para registrar la lectura RSSI
send_client(0,"PUT /v4/spreadsheets/");
send_client(0,spreadsheetId);
send_client(0,"/values/");
send_client(0,"Hoja%201!A");
sprintf(str_celda,"%lu",celda);
send_client(0,str_celda);
send_client(0,"?valueInputOption=USER_ENTERED HTTP/1.1\r\n");
send_client(0,"Authorization: Bearer ");
send_client(0,access_token);
sprintf(str_rssi,"%d",valor_rssi);
content_length=get_str_length(json_1)+get_str_length(str_celda)+
content_length=content_length+get_str_length(str_rssi)+get_str_length(json_3);
sprintf(str_content_length,"%lu",content_length);
send_client(0,"\r\nConnection: keep-alive\r\nContent-Length: ");
sprintf(str_content_length,"%lu",content_length);
send_client(0,str_content_length);
send_client(0,"\r\nHost: sheets.googleapis.com\r\nContent-Type: application/json\r\n\r\n");
send_client(0,json_1);
send_client(0,str_celda);
send_client(0,json_2);
send_client(0,str_rssi);
codigo_estado_http=finish_request_http_client(0,json_3);
```

Figura P22-B.6 *Segunda parte del código de la práctica 22-B para registro de datos.*

```
        //La conexión se recicla
        //Se envía petición HTTP para actualizar la última celda
        content_length=get_str_length(str_celda)+58;
        sprintf(str_content_length,"%lu",content_length);
        create_ssl_pclient("googleapis.com",443,0);//
        send_client(0,"PUT /v4/spreadsheets/");
        send_client(0,spreadsheetId);
        send_client(0,"/values/");
        send_client(0,"B2?valueInputOption=USER_ENTERED HTTP/1.1\r\n");
        send_client(0,"Authorization: Bearer ");
        send_client(0,access_token);
        send_client(0,"\r\nContent-Length: ");
        send_client(0,str_content_length);
        send_client(0,"\r\nHost: sheets.googleapis.com\r\nContent-Type: application/json\r\n\r\n");
        send_client(0,"{\"range\":\"B2\",\"majorDimension\":\"COLUMNS\",\"values\":[[\"A");
        send_client(0,str_celda);
        send_client(0,"\"]]");
        codigo_estado_http=finish_request_http_client(0,"}");
        delete_client(0);

        //Si se ejecutó correctamente
        if(codigo_estado_http==200)
            celda++;
        //Si no está autorizada entonces actualiza el access token
        else if(codigo_estado_http==401)
            refresh_token_oauth20(0,access_token,refresh_token);
        delay_ms(1000);
    }
}
```

Figura P22-B.7 *Tercera parte del código de la práctica 22-B para registro de datos.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define spreadsheetId "ID de la hoja de cálculo"
char access_token[]="access token";
char refresh_token[]="refresh token";

long codigo_estado_http=0;
char json_buffer[20];

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Inicialización de la LCD
    lcd_init();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    //Actualiza access token
    refresh_token_oauth20(0,access_token,refresh_token);
    delay_ms(500);
    while(1)
    {
        /* En el canal de comunicación 0, se establece un cliente SSL*/
        create_ssl_pclient("googleapis.com",443,0);
        /*Se envía petición HTTP para solicitar el contenido de una celda
        de la hoja de cálculo indicada*/
        send_client(0,"GET /v4/spreadsheets/");
        send_client(0,spreadsheetId);
        send_client(0,"/values/C2 HTTP/1.1\r\nAuthorization: Bearer ");
        send_client(0,access_token);
        send_client(0,"\r\nHost: sheets.googleapis.com");
        codigo_estado_http=get_matrix_spreadsheets(0,"\r\n\r\n",json_buffer, sizeof(json_buffer));
        delete_client(0);

        //Limpia la LCD
        printf(lcd_putc,"\f");
        //Posiciona escritura en la LCD
        lcd_gotoxy(2,1);
        //Imprime en la LCD
        lcd_putc("Promedio RSSI:");
        //Posiciona escritura en la LCD
        lcd_gotoxy(1,2);
        //Imprime en la LCD
        printf(lcd_putc,"%s",json_buffer);

        //Si no está autorizada entonces actualiza el access token
        if(codigo_estado_http==401)
            refresh_token_oauth20(0,access_token,refresh_token);
        delay_ms(1000);
    }
}
```

Figura P22-B.8 *Código de la práctica 22-B para consulta de datos.*

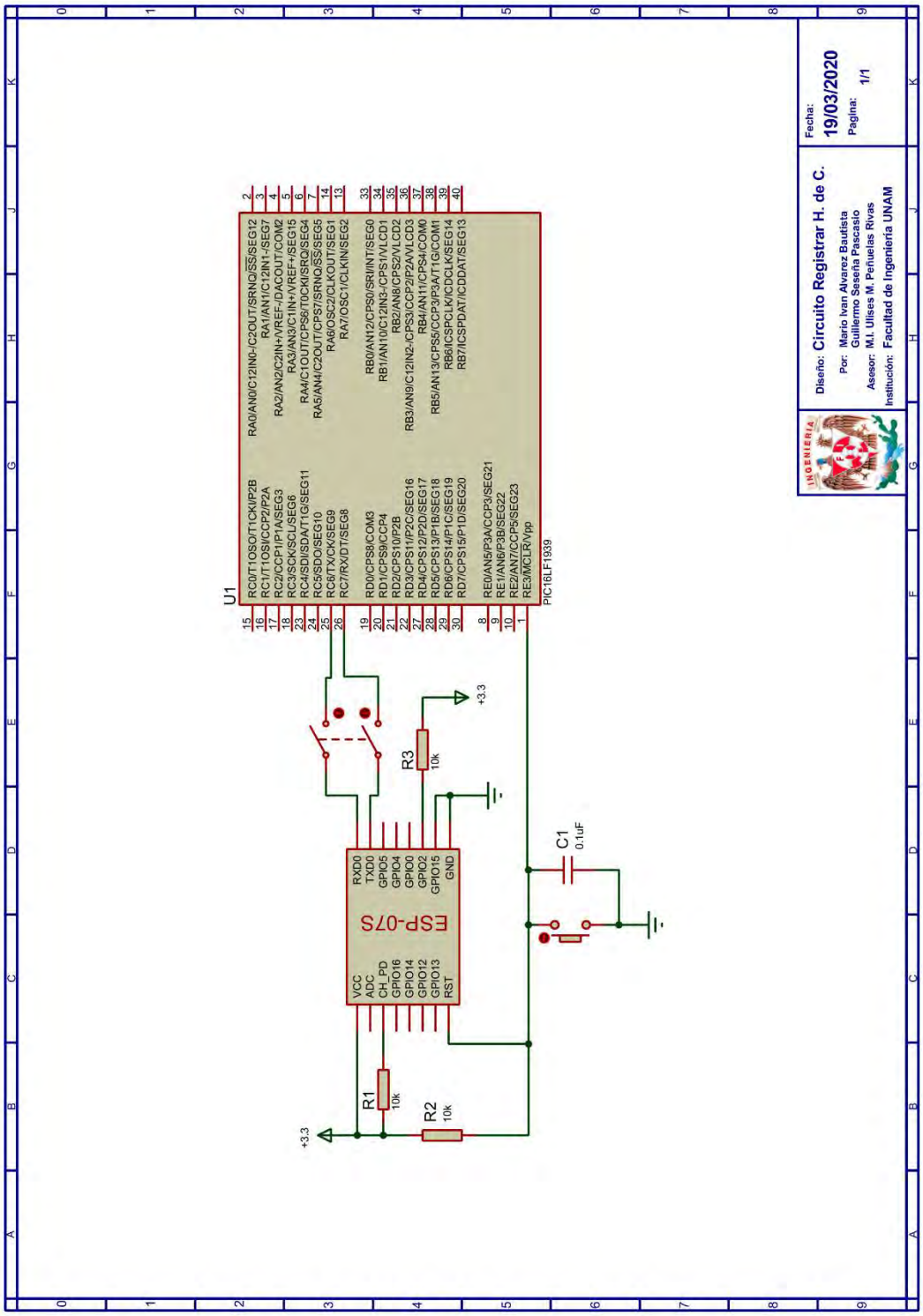


Figura P22-B.9 *Circuito de la práctica 22-B para registro de datos.*

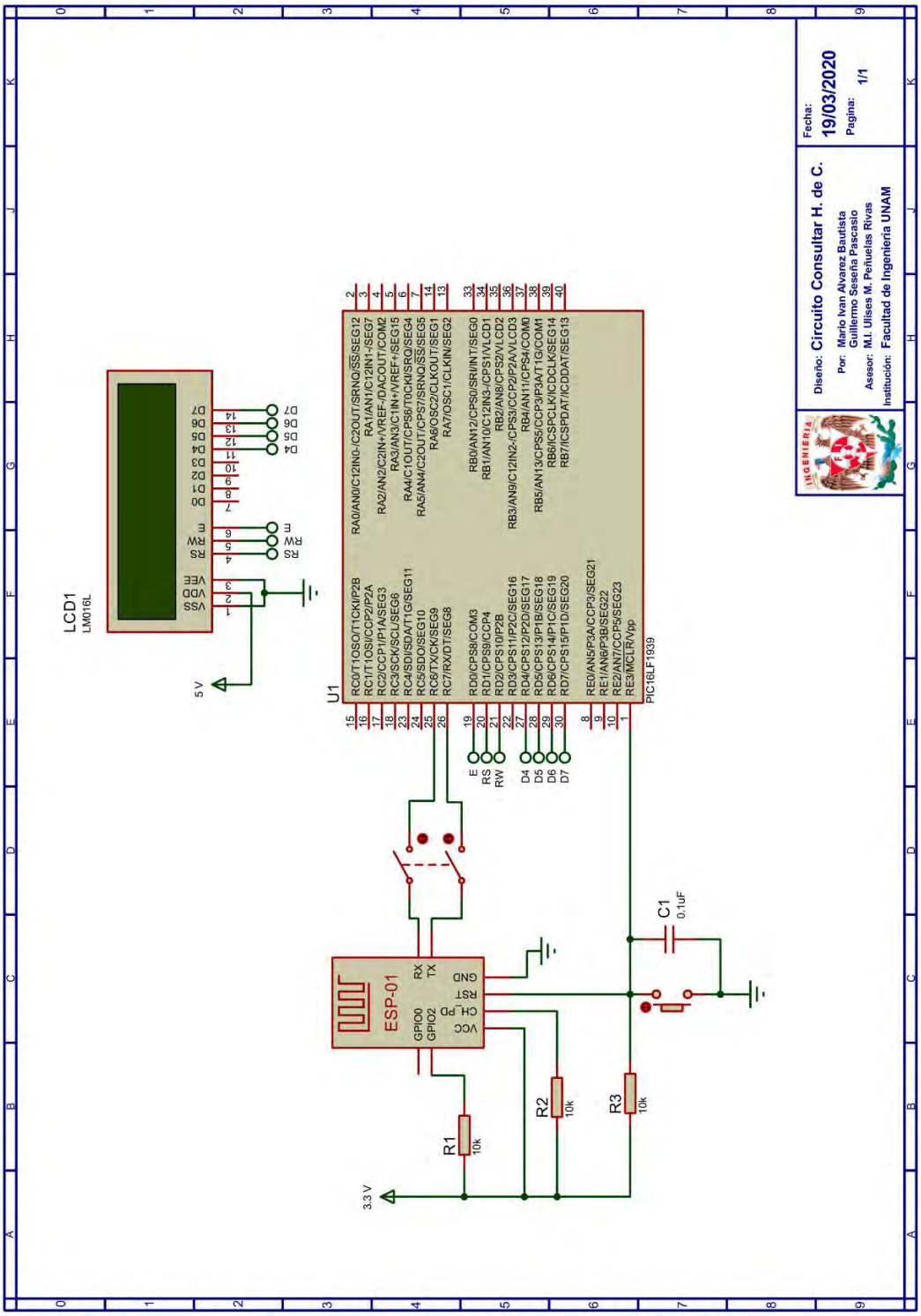


Figura P22-B.10 *Circuito de la práctica 22-B para consulta de datos.*

Referencias

1. Google Developers. *Introduction to the Google Sheets API*. [Citado 2020 Mayo]; Disponible en:https://developers.google.com/sheets/api/guides/concepts.

Práctica 22-C APIs de Google con OAuth 2.0: envío de correo electrónico a través de la API de Gmail

Introducción

La API de Gmail es otra de las API del catálogo de Google, la cual también está basada en REST. Esta API permite a las aplicaciones realizar diversas acciones, entre las que destacan: redactar, enviar y revisar correos electrónicos. Todas las peticiones que fluyen a la API requieren de autorización a través de OAuth2.0, para la cual se ofrecen hasta diez *scopes* que se describen en la guía que se encuentra en [1].

El PIC-ESP puede hacer uso de la API de Gmail al construir y ejecutar peticiones HTTP utilizando las funciones de la biblioteca ESP. Sin embargo, presenta el defecto de que el almacenamiento y la extracción de información útil del contenido de las respuestas, no es óptimo. En consecuencia, desarrollar aplicaciones que involucren respuestas con grandes cantidades de información, como revisar correos, pueden resultar tareas complicadas que consumirían gran parte de los recursos del microcontrolador PIC.

Objetivo

Enviar correos electrónicos a través de la API de Gmail.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION, E_HTTP_CLIENT y E_GOOGLE_OAUTH20

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulos WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF

Descripción

Esta práctica es muy semejante a las prácticas 17-A y 17-B, en la que se envía un correo electrónico cada vez que se presiona un botón, pero en lugar de utilizar el protocolo SMTP se utilizará la API de Gmail. Por lo que, cada vez que

se presione el botón, se ejecutará una petición HTTP cuya composición se muestra en la figura P22-C.2.

En esta actividad se deberá escribir y cifrar en base64 todo el contenido del correo que desee enviar antes de colocarlo en el código del microcontrolador PIC. Para escribir el contenido, se utiliza algún editor de texto como un bloc de notas. Se debe especificar al menos los encabezados:

- *From:,* que contiene la dirección de correo de la cuenta Google usada;
- *To:,* que contiene la dirección de correo de quien recibirá el mensaje;
- *Subject,* que es el asunto del correo.

Y en el cuerpo se escribirá lo que se desee.

La longitud total del contenido no debe ser mayor 1528 caracteres incluyendo no imprimibles, ya que cuando se codifique incrementa en su longitud un 33%. En el caso de que se requiera enviar un contenido con una longitud mayor, entonces se debe modificar el código de la práctica para enviarlo en fragmentos (ya codificados) menores de 2047 bytes.

En la figura P22-C 1 se muestra un ejemplo del contenido escrito para un correo electrónico con contenido de texto plano.

```
From: biblioteca.esp.fi.unam@gmail.com
To: esp.3266.32@gmail.com
Subject: Correo electronico desde API
MIME-Version: 1.0
Content-Type: text/plain;

Contenido del correo.
```

Figura P22-C 1 *Ejemplo de contenido de un correo electrónico.*

Cuando ya se haya terminado de escribir el contenido del correo, se debe codificar éste en base64, para lo cual se puede usar alguna página como:

https://base64.guru/converter/encode/text

```
POST /gmail/v1/users/{dirección de correo gmail}/messages/send HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer {access token}
Content-length: {longitud de contenido}
Content-type: application/json

{"raw":"{contenido codificado en base 64}"}
```

Figura P22-C.2 *Composición de petición HTTP para enviar un mensaje (correo) con Gmail API.*

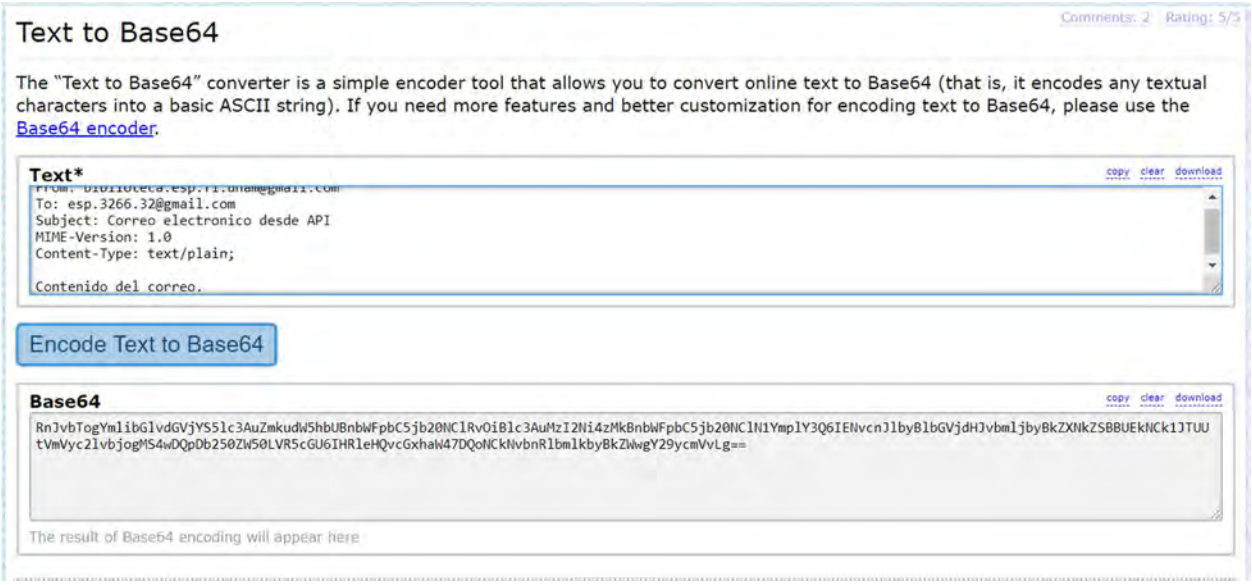


Figura P22-C.3 *Ejemplo de codificación en base64 del contenido de un correo en la página Base64 Guru.*

En esta página sólo se coloca el texto escrito y se presiona codificar (ver figura P22-C.3). El resultado de la codificación debe ser incluida en el código de la práctica.

Para esta práctica ya se debe contar con un *access token* y un *refresh token*, los cuales se obtienen de la misma manera como en la práctica 22-A, pero con el *scope*:

https://www.googleapis.com/auth/gmail.send

Y habilitando Gmail API en la consola de Google. El *refresh token* es utilizado para actualizar el *access token* en cuanto se rechace una petición (debido al vencimiento del *access token*).

Para comprobar que se ha elaborado correctamente el contenido del correo se puede acceder al enlace:

https://developers.google.com/gmail/api/v1/reference/users/messages/send

Y realizar una prueba colocando la dirección de la cuenta de Google y en el cuerpo de la petición, incluir el resultado de la codificación del contenido. Al ejecutarla con éxito, se tiene que obtener un código igual a 200 (ver figura P22-C.4), además, el correo se debería de haber enviado a su destino.

Código

En el archivo esp.h, en la zona de OAuth20 se colocan los siguientes parámetros (ver figura P22-C.6):

- ID de cliente

- Secreto del cliente
- *Scope*

El código que deberá ejecutar el microcontrolador PIC para esta práctica se muestra en la figura P22-C.7, en el que se debe indicar los siguientes parámetros:

- Nombre y contraseña de AP con acceso a internet
- Un *access token*
- Un *refresh token* válido
- Dirección de correo electrónico de Gmail que envía el correo electrónico
- Contenido codificado en base64 de un correo electrónico a enviar

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P22-C.8.

Resultados

Cada vez que se presione el botón, se enviará el correo electrónico con el contenido y destino especificado, el cual está codificado en el programa del uC PIC. El correo se mostrará en el buzón recibidos del receptor y en el de enviados del emisor (ver figura P22-C.5).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2Yf3unx

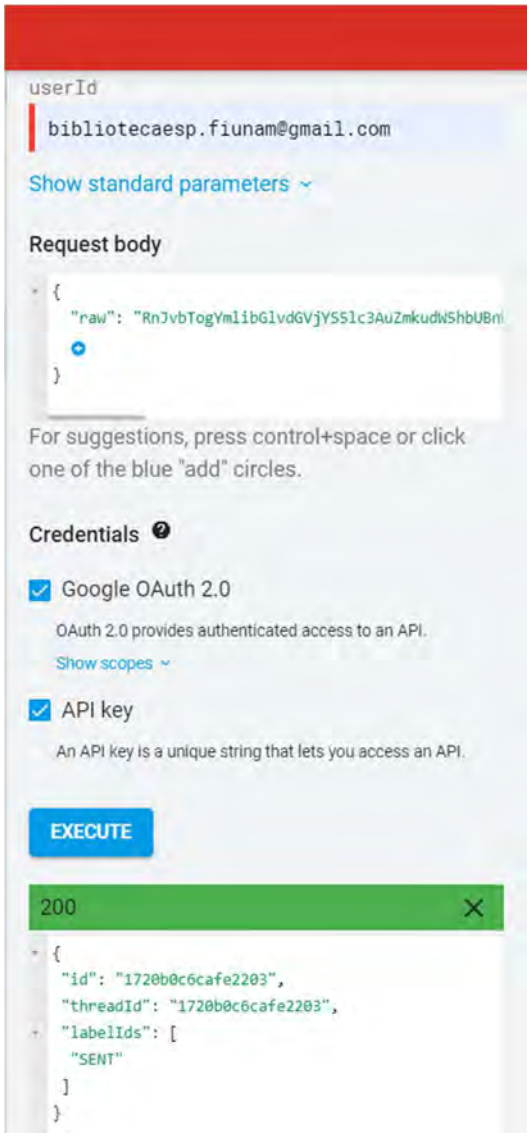


Figura P22-C.4 *Ejemplo de ejecución exitosa de envío de correo desde la documentación de la API.*

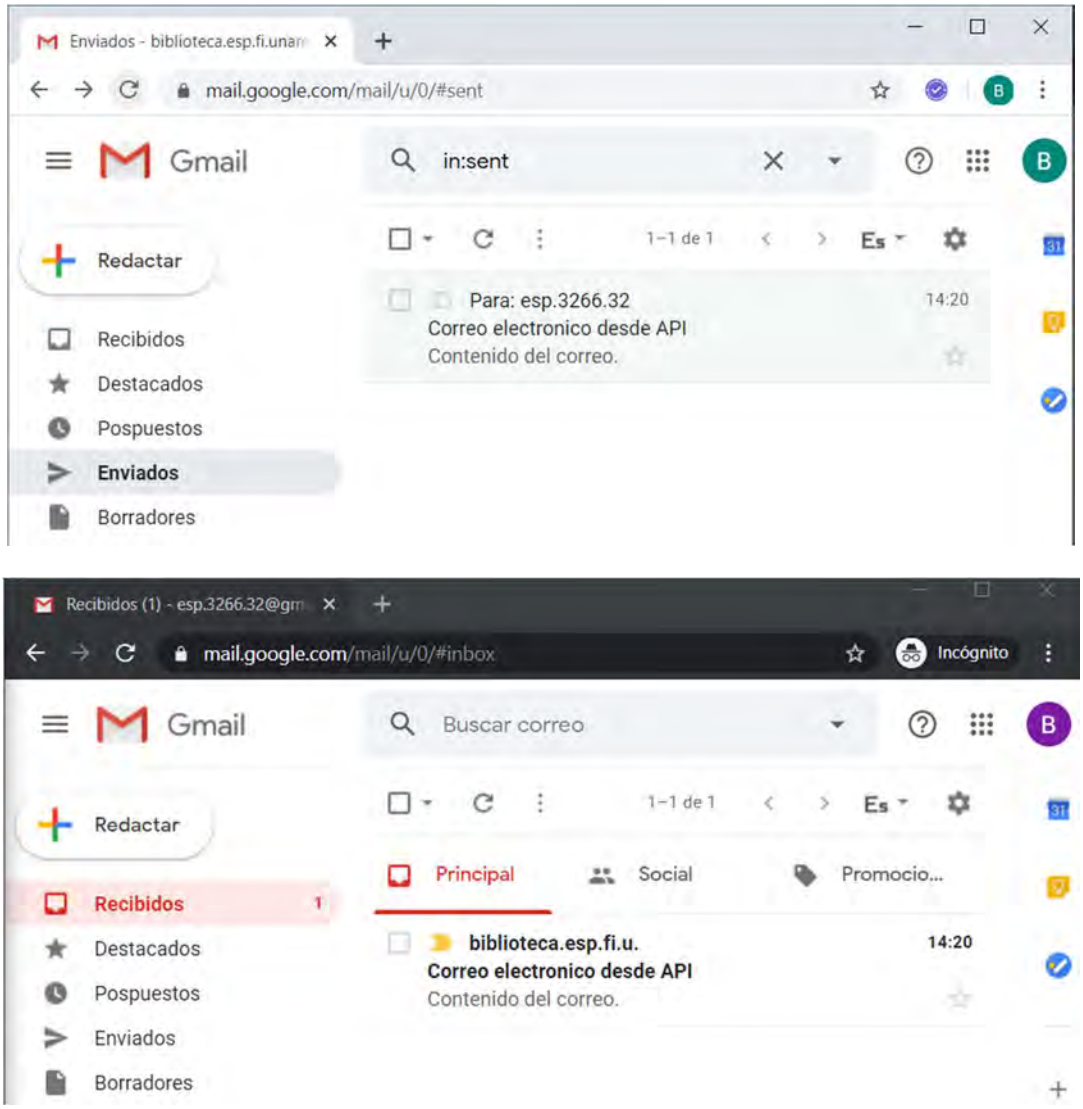


Figura P22-C.5 *Correo electrónico enviado por el módulo mediante la API de Gmail, mostrado en la carpeta de enviados de la cuenta utilizada y mostrada en recibidos del destinatario.*

```
/*****************************************************************************************
 *                                OAUTH2.0
*****************************************************************************************
*/
#define CLIENT_ID_OAUTH20 "ID de usuario"
#define CLIEND_SECRET_OAUTH20 "Secreto de usuario"
#define SCOPE_OAUTH20 "https:\/\/www.googleapis.com/auth/gmail.send"

#define HOST_OAUTH20 "accounts.google.com"
#define HOST_OAUTH20_ "www.googleapis.com"
#define PORT_OAUTH20 443
#define URL_OAUTH20 "/o/oauth2/v2/auth?"
#define URL_OAUTH20_ "/oauth2/v4/token"
```

Figura P22-C.6 *Zona de ingreso de parámetros para la práctica 22-C en esp.h.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define correo_emisor "direccion_de_correo_emisor@gmail.com"
char access_token[]="access token";
char refresh_token[]="refresh token";
const char contenido[]="contenido codificado en base64";

long codigo_estado_http=0;
long content_length=0;
char str_content_length[4];
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    while(1)
    {
        //Al presionar botón
        if(1==input_state(PIN_B7))
        {
            //Actualiza token cuando anteriormente no se estuvo autorizado
            if(codigo_estado_http==401)
                refresh_token_oauth20(0,access_token,refresh_token);
            // En el canal de comunicación 0, se establece un cliente SSL/TSL
            create_ssl_pclient("googleapis.com",443,0);
            /*Se envía petición HTTP para enviar un correo electrónico*/
            send_client(0,"POST /gmail/v1/users/");
            send_client(0,correo_emisor);
            send_client(0,"/messages/send HTTP/1.1\r\n");
            send_client(0,"Host: www.googleapis.com\r\n");
            send_client(0,"Authorization: Bearer ");
            send_client(0,access_token);
            send_client(0,"\r\nContent-length: ");
            //calcula longitud del contenido de la petición
            content_length=(sizeof(contenido)-1)+10;
            sprintf(str_content_length,"%lu",content_length);
            send_client(0,str_content_length);
            send_client(0,"\r\nContent-type: application/json\r\n\r\n");
            send_client(0,"{\"raw\":\"");
            send_mclient_const(0,contenido);
            codigo_estado_http=finish_request_http_client(0,"\"}");
            delete_client(0);

            while(1==input_state(PIN_B7))
            {
                //Se mantiene hasta que se suelte
            }
        }
    }
}
```

Figura P22-C.7 *Código de la práctica 22-C.*

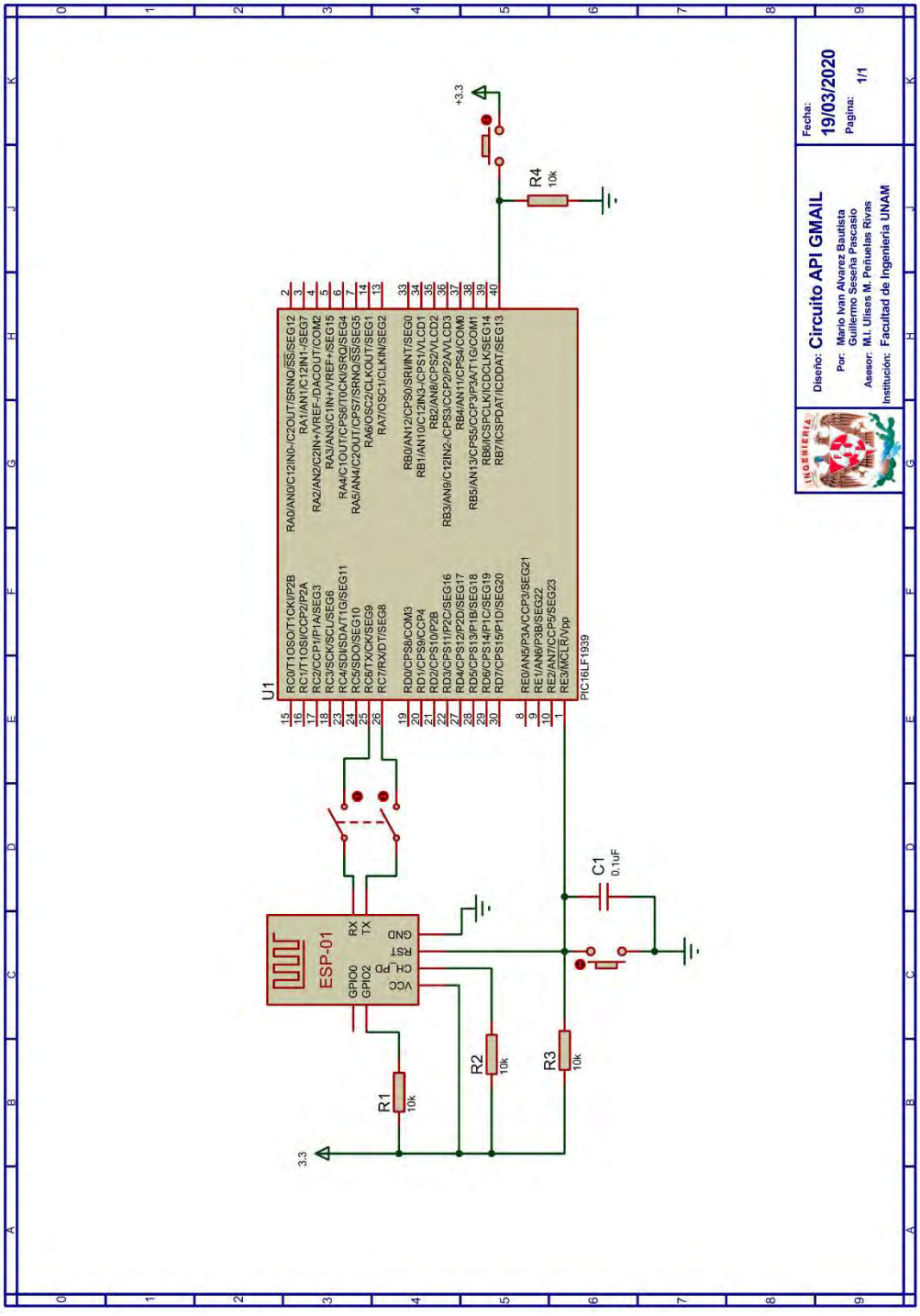


Figura P22-C.8 *Circuito de la práctica 22-C.*

Referencias

1. *Gmail API Overview*. Disponible en:https://developers.google.com/gmail/api/guides.

Práctica 23-A MQTT Publicador

Introducción

Transporte de telemetría de cola de mensajes (*Message Queue Telemetry Transport,* MQTT) es un protocolo de aplicación binario que fue concebido inicialmente para comunicar sensores en oleoductos de petróleo con satélites. Su nombre no fue dado en representación a su arquitectura ya que no es un protocolo de colas de mensajería, si no fue dado como parte de una serie de productos desarrollados por IBM. En la actualidad MQTT está siendo implementado en comunicación máquina-máquina (M2M) y aplicaciones IoT, ya que es liviano, abierto y de fácil implementación [1].

MQTT ofrece un modelo de comunicación diferente a la clásica relación cliente-servidor, en vez de ello utiliza el modelo de publicación-suscripción. En este modelo, la información es emitida por clientes conocidos como publicadores (*publishers*) y es recibida por clientes llamados suscriptores (*subscribers*). La información es enviada en mensajes llamados PUBLISH, los cuales tienen como destino algún tema (*topic*) de interés. A estos temas, los suscriptores se dan de alta para recibir toda la información publicada en dichos rubros. Los dos tipos de clientes MQTT nunca se conectan entre sí de manera directa, para ello interviene un tercero conocido como bróker, el cual se encarga de recibir todos los mensajes PUBLISH enviados por los publicadores y distribuirlos a los subscriptores (ver figura P23-A.1). La conexión entre un cliente y el bróker se mantiene abierta durante toda una sesión, permitiendo una comunicación rápida y bidireccional [1].

En las versiones 3.1 y 3.1.1 del protocolo MQTT, especificadas en [2] y [3] respectivamente, describen catorce tipos

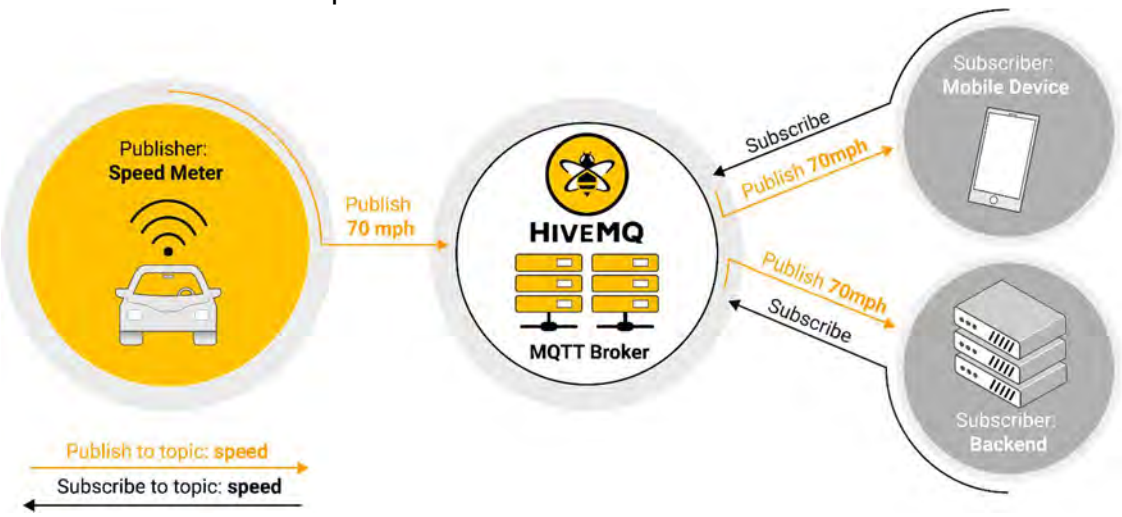


Figura P23-A.1 *Arquitectura de publicación / suscripción de MQTT [1].*

de mensajes que están definidos por encabezados binarios, estos mensajes fluyen entre los clientes y el bróker, permitiendo a los clientes realizar las siguientes acciones:

- Conectarse al bróker (Iniciar sesión)
- Suscribirse a uno o varios temas
- Darse de baja de uno o varios temas
- Publicar mensajes
- Recibir mensajes de acuerdo con las suscripciones del cliente
- Realizar ping

Antes de que un cliente MQTT pueda iniciar sesión debe establecer una conexión TCP/SSL con el bróker. Posteriormente en el inicio de sesión, el cliente tiene que especificar: un ID de usuario, que es un indicador único que señala al bróker con que cliente se está comunicando; el nombre de usuario; contraseña de usuario; y el *keepalive,* que es el tiempo máximo en segundos establecido por el cliente en que el bróker puede mantener activa la conexión con el cliente cuando éste se encuentre inactivo.

Durante el proceso de inicio de sesión se puede especificar si la sesión a la que se accede es o no persistente (*cleanSession*). En una sesión persistente el cliente conserva sus suscripciones aun cuando haya finalizado la sesión y recibir mensajes pendientes al momento de regresar a la sesión, mientras que, en una sesión no persistente, el bróker no guarda ningún dato al finalizar sesión, por lo que el cliente tiene que volver a suscribirse a los temas de interés en una nueva sesión.

En el inicio de sesión también se puede especificar si el cliente desea establecer un mensaje de última voluntad (*last will*), es decir, el cliente publicará un mensaje PUBLISH en un tema específico cuando pierda la conexión abruptamente, lo cual resulta útil en el caso de que el cliente esté funcionando como publicador, puesto que indica a los suscriptores del tema, que el publicador ha dejado de funcionar. El mensaje de última voluntad puede ser establecido con cualquiera de los tres niveles de calidad de servicio (*Quality of Service*) y/o marcar como mensaje retenido.

Estos tres niveles de calidad QoS0, QoS1 y QoS2. están presentes en todos los mensajes PUBLISH que se emiten y retransmiten.

QoS0, este nivel es conocido como “entrega y olvida”, los mensajes se envían una sola vez sin ningún mecanismo que asegure la entrega. En este nivel de calidad, los mensajes son enviados con la mayor rapidez, pero con la misma fiabilidad de la una conexión TCP, ideal para aplicaciones en las que se puede garantizar una buena conexión con el bróker o en aplicaciones en las

que no existe problema si ocasionalmente se pierde información.

QoS1, este nivel de calidad asegura que la entrega del mensaje se realice al menos una vez. El receptor puede recibir dos veces un mismo mensaje. Este nivel de calidad es ideal para aplicaciones que requieren de rapidez y fiabilidad, y no tenga problemas con recibir mensajes duplicados.

QoS2, este nivel calidad asegura que el mensaje sea entregado solo una vez al receptor, ya que, los mensajes son procesados por el receptor hasta que el emisor sepa que no es necesario enviar duplicados. Este nivel de calidad es el más fiable del protocolo, pero el más lento, es ideal para aplicaciones que requieren de un estricto nivel de fiabilidad [1].

Un mensaje PUBLISH puede estar marcado como retenido, esto le indica al bróker que debe almacenarlo como el último valor útil para un tema, valor que se sobrescribe cada vez que se publica un nuevo mensaje retenido. Este tipo de mensajes PUBLISH sirven para que un cliente pueda inmediatamente conocer la información de un tema al momento de suscribirse y no esperar a que un publicador lo actualice.

Un cliente MQTT puede suscribirse o cancelar su suscripción de uno o varios temas en una misma sesión. Por cada tema que se suscriba, el cliente debe indicar cuál es la calidad máxima con la que puede recibir los mensajes PUBLISH, eligiendo cualquiera de los niveles que ofrece el protocolo. La calidad con la que llegan los mensajes al suscriptor depende de la calidad con la que el publicador la emitió, si el publicador emitió un mensaje con una calidad mayor a la que subscriptor ha establecido, entonces el bróker entregará al suscriptor el mensaje con la máxima calidad que puede soportar dicho suscriptor, en cambio cuando la calidad del mensaje es igual o menor a la calidad máxima establecida por el suscriptor, el bróker entregará el mensaje con la calidad original.

Los nombres de los temas se representan como cadenas de caracteres UTF-8, que comúnmente emplean sólo caracteres imprimibles (caracteres ASCII), con la excepción que no deben iniciar con el carácter “$” que tiene un uso interno o temas que inicien con los comodines “+” y “#”.

Los temas se pueden especificar en distintos niveles de una estructura jerárquica de temas, cada uno de estos niveles está separado por una diagonal. Por ejemplo, si en un edificio se ha instalado sensores de temperatura y humedad para cada una de sus salas, y cada uno de los sensores tiene designado un tema en el cual publican los valores obtenidos, entonces los temas se podrían definir de la siguiente forma: edificio/piso”n”/sala”n”/”sensor”, si

un cliente necesita obtener la información publicada por el sensor de temperatura de la sala número dos del segundo piso se tendría que suscribir al tema edificio/piso2/sala2/temperatura.

Los clientes MQTT se pueden suscribir uno a uno a los temas de interés o suscribirse a varios temas a la vez usando comodines. Uno de los dos comodines que proporciona el protocolo es utilizado a través del carácter “+” que remplaza un nivel de tema, permitiendo al cliente suscribirse a los temas en común de un solo nivel, retomando el ejemplo del edificio, si un cliente deseara conocer los valores de humedad de todas las salas del piso dos, tendría que suscribirse al tema edificio/piso2/+/humedad. El segundo comodín es accedido con el carácter “#” que remplaza el nivel a partir del cual se suscribirá a todos los niveles restantes, si un cliente quisiera conocer los valores de temperatura y humedad de todas las salas del segundo piso tendría que suscribirse al tema edificio/piso2/#.

Cuando un cliente MQTT se ha mantenido inactivo por un tiempo mayor al establecido en el *keepalive*, el bróker termina la conexión con el cliente de manera abrupta. Para evitar que el cliente MQTT sea desconectado por inactividad, éste debe enviar un mensaje PING al bróker antes de que se venza el *keepalive*, esto indica al bróker que el cliente sigue activo y desea mantener activa la conexión.

Para las versiones 3.1 y 3.1.1, MQTT envía información completamente binaria transmitida en bytes, por lo que, a diferencia de protocolos como HTTP, la información carece de formato, por lo que los usuarios se encargan de establecer la estructura y el formato que requieran para transmitir distintos tipos de datos.

La biblioteca ESP implementa MQTT en sus versiones 3.1 y 3.1.1, aunque en el 2019 fue lanzada la última especificación (versión 5.0), no fue tomada en cuenta para la biblioteca ESP por tres motivos:

- Aún predominan las plataformas IoT que emplean las dos anteriores versiones
- Algunas mejoras en esta versión son en características que no pueden ser aprovechadas por el diseño de la biblioteca como códigos de respuesta que informan las causas de errores, ya que la biblioteca ESP sólo procesa las respuestas como exitosas o no;
- Por simplicidad se ha limitado la utilidad a ciertas características de las versiones implementadas que en esta nueva versión han mejorado.

La biblioteca, permite utilizar al PIC-módulo como un cliente MQTT, ya sea

como publicador y/o suscriptor, el cual establece una conexión TCP o SSL/TLS con algún servidor que funge como un bróker MQTT, dicha conexión se recomienda efectuarse en el canal número cuatro del módulo para tener compatibilidad con otras aplicaciones, aunque se puede seleccionar cualquier otro canal. La biblioteca es capaz de emitir y/o recibir los catorce tipos de mensajes definidos en las versiones mencionadas, pero con algunas restricciones, además por facilidad la diferencia entre estas dos versiones para la biblioteca sólo que se ha reflejado en el proceso de inicio sesión. A continuación, se señalan las limitaciones que tiene el usuario al usar la biblioteca:

- Todas las sesiones son no persistentes.
- Se puede activar mensaje de última voluntad, pero éste sólo puede ser establecido con calidad QoS0 y como no retenido.
- La longitud máxima de envío/recepción de *strings* es de 127 bytes, por lo que recomienda publicar mensajes cortos al igual que establecer nombres de temas concisos.
- Para los mecanismos de entrega de mensajes PUBLISH para QoS1 y QoS2, la biblioteca intenta enviar/recibir el mensaje en un máximo de tres intentos, en caso de que no se complete con estos intentos el mensaje es descartado.

Objetivo

Establecer al módulo como un cliente MQTT que funcione como publicador utilizando la plataforma CloudMQTT

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_MQTT_v31

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF

Descripción

En esta práctica el módulo será establecido como un publicador, el cual proporcionará el valor de un contador dentro del programa del PIC. Para mostrar la diferencia entre los tres niveles de calidad de servicio que ofrece el protocolo, la práctica se ha dividido en tres actividades:

- Publicar con QoS0 marcado como no retenido
- Publicar con QoS1 marcado como retenido
- Publicar con QoS2 marcado como retenido

El módulo usará un ID aleatorio colocado manualmente en el código, establecido un *keepalive* de 10 segundos, el cual es un valor pequeño, ya que la función de envió se ejecuta cada 20 *ms*. También se activará el mensaje de última voluntad el cual será para el mismo tema al que el módulo publicara valor del contador y el mensaje hace alusión que se ha desconectado el publicador.

Para esta práctica se utiliza la plataforma CloudMQTT y se emplea la versión 3.1 del protocolo. Para utilizar dicha plataforma hay que registrarse en la página: https://www.cloudmqtt.com/ seleccionando algunos de sus planes. En esta práctica se ilustra desde un plan gratuito llamado “Cat”.

Ya registrado se crea una nueva instancia a la cual se nombra como se desee, posteriormente en “*DETAILS*” de la instancia se debe mostrar como en la figura P23-A.2, en ésta se señala el nombre del servidor y el puerto TCP que se utilizan en la práctica.

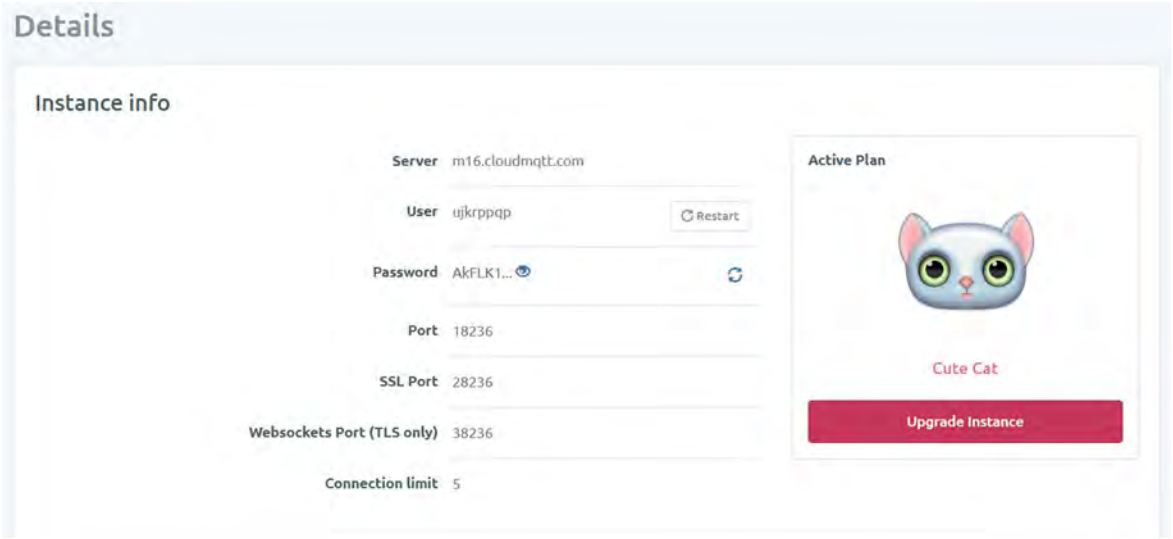


Figura P23-A.2 *Detalles de una instancia creada en CloudMQTT.*

Para que el módulo inicie sesión como un cliente MQTT no se utilizan las credenciales mostradas en la figura P23-A.2, sino que se crean unas nuevas para acceder a los temas que se crearán, para ello se debe ingresar a la opción "*USERS & ACL*", en la sección "*Users*" se escribe un nombre de usuario y una contraseña (ver figura P23-A.3), las cuales son utilizadas en el código de la práctica.

En la misma página, pero en la sección "*ACLs*" (*Access Control List*) se crea el tema al cual el módulo publicará los valores de un contador. Se selecciona la opción "*topic*" de recuadro azul, se selecciona el nombre de usuario creado anteriormente y como "*pattern*" se coloca el nombre del tema, que en la figura P23-A.4 se llama pic_esp/cont, se habilita "*Read Access*" y "*Write Access*" para acceso a lectura y escritura respectivamente. El nombre del tema está usando dos niveles. Ya creado el tema, el módulo podrá enviar información a la plataforma, la cual podrá ser observada en la sección de "*WEBSOCKET UI*" de la misma plataforma.

Para más información de esta plataforma se recomienda revisar:

https://www.cloudmqtt.com/docs/index.html

Código

El código que deberá ejecutar el microcontrolador PIC en la primera actividad es el mostrado en la figura P23-A.8, para la segunda es el mismo, pero se reemplaza la línea:

```
publishQoS0_mqtt(topic_name,str_
contador,0);
```

Por:

```
publishQoS1_mqtt(topic_name,str_
contador,1);
```

Finalmente, en la tercera actividad ésta misma es reemplazada por:

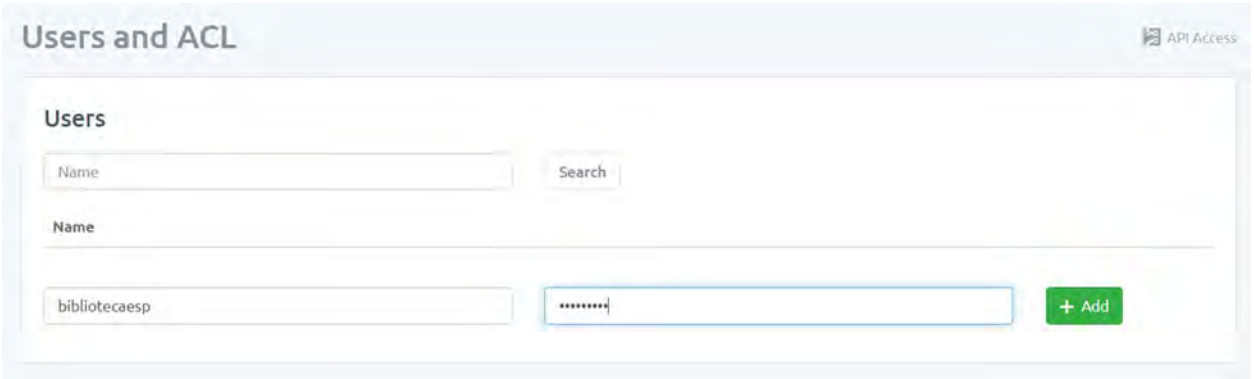


Figura P23-A.3 *Creación de usuario de una instancia de CloudMQTT.*

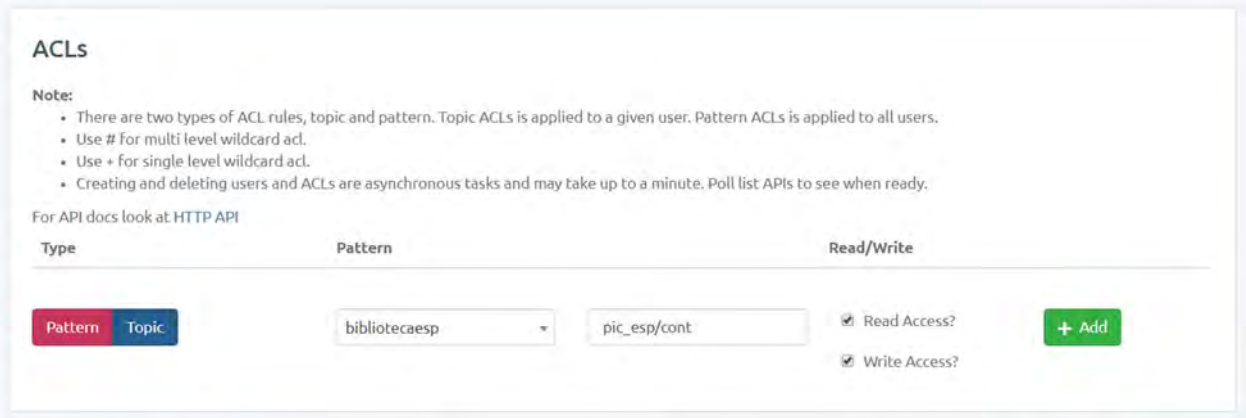


Figura P23-A.4 *Creación de un tema en una instancia de CloudMQTT.*

`publishQoS2_mqtt(topic_name,str_contador,1);`.

Para todas las actividades se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet.
- Nombre de usuario y contraseña de “USERS & ACL”.
- Nombre de tema.

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P23-A.9.

Resultados

Para todas las actividades de la práctica, se accederá a la página de la instancia creada en la plataforma CloudMQTT y se deberá ingresar a la opción de “*WEBSOCKET UI*” para visualizar todos los mensajes que están siendo publicados en esta instancia.

En cada una de las actividades se publicará un valor numérico que va incrementando en una unidad hasta llegar a 255, posteriormente se desbordará, reiniciando el valor y volviendo a incrementar, esto lo realiza hasta que se pierda la conexión con el bróker o se apague la fuente de alimentación al conjunto PIC-módulo (ver figura P23-A.5).

La diferencia que se nota entre las tres actividades es que los mensajes con QoS0 son publicados con la mayor rapidez, mientras que con QoS2 son publicados con la menor rapidez.

También para comprobar el funcionamiento se puede hacer uso de alguna aplicación móvil que permita crear un cliente MQTT, en esta demostración se utiliza IoT MQTT Panel que se encuentra gratuitamente en la *Play Store* de Android. En ella se ha creado un *Gauge* el cual está suscrito al tema

utilizado en esta práctica, como se observa en la figura P23-A.6, éste va incrementando de acuerdo con los valores publicados por el módulo. Para la primera actividad si se apaga la fuente de alimentación del módulo, y se reinicia la aplicación, no se guardará el ultimo valor publicado, mientras que, para la segunda y la tercera actividad si se guardarán, ya que los mensajes han sido marcados como retenidos.

Finalmente, para cualquiera de las actividades, si se apaga la alimentación del módulo y se espera más de 10 segundos, se publicará el mensaje de última voluntad que indica que el publicador, en este caso el módulo, se ha desconectado (ver figura P23-A.7).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2V027qW

Received messages

Topic	Message
pic_esp/cont	000
pic_esp/cont	1
pic_esp/cont	2
pic_esp/cont	3
pic_esp/cont	4
pic_esp/cont	5
pic_esp/cont	6
pic_esp/cont	7
pic_esp/cont	8
pic_esp/cont	9

Figura P23-A.5 *Mensajes publicados por el módulo.*

Received messages

Topic	Message
pic_esp/cont	101
pic_esp/cont	102
pic_esp/cont	103
pic_esp/cont	104
pic_esp/cont	105
pic_esp/cont	5010_des

Figura P23-A.7 *Mensaje de última voluntad.*

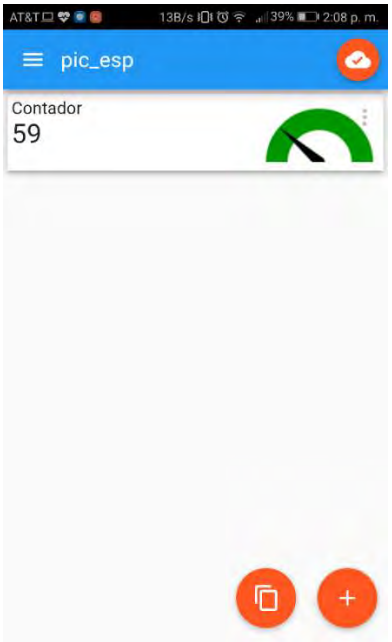


Figura P23-A.6 *Visualizando mensajes publicados desde IoT MQTT Panel.*

```
#include <16lf1939.h>
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define user_name "nombre de usuario"
#define user_password "contraseña de usuario"
#define topic "nombre del tema"

int str_contador[]="000";
int contador=0;

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Se crea el cliente MQTT (se conecta al servidor de la plataforma)
    para el canal numero cuatro*/
    create_mqtt("m16.cloudmqtt.com",18236,4);
    /*El cliente se conecta a una cuenta de CloudMQTT.Se establece
    el ID de cliente MQTT como 5010 , un keepalive de 10s, se activa el
    mensaje de última volutad para el tema seleccionado con mensaje "5010_des"*/
    connect_mqtt("5010",user_name,user_password,10,topic,"5010_des");
    while(1)
    {
        /*Se publica el valor del contador para el tema seleccionado y
        se marca como no retenido*/
        publishQoS0_mqtt(topic,str_contador,0);
        contador++;
        //Valor del contador es colocado en el string que se envía
        sprintf(str_contador,"%u",contador);
        delay_ms(20);
    }
}
```

Figura P23-A.8 *Código de la primera actividad de la práctica 23-A.*

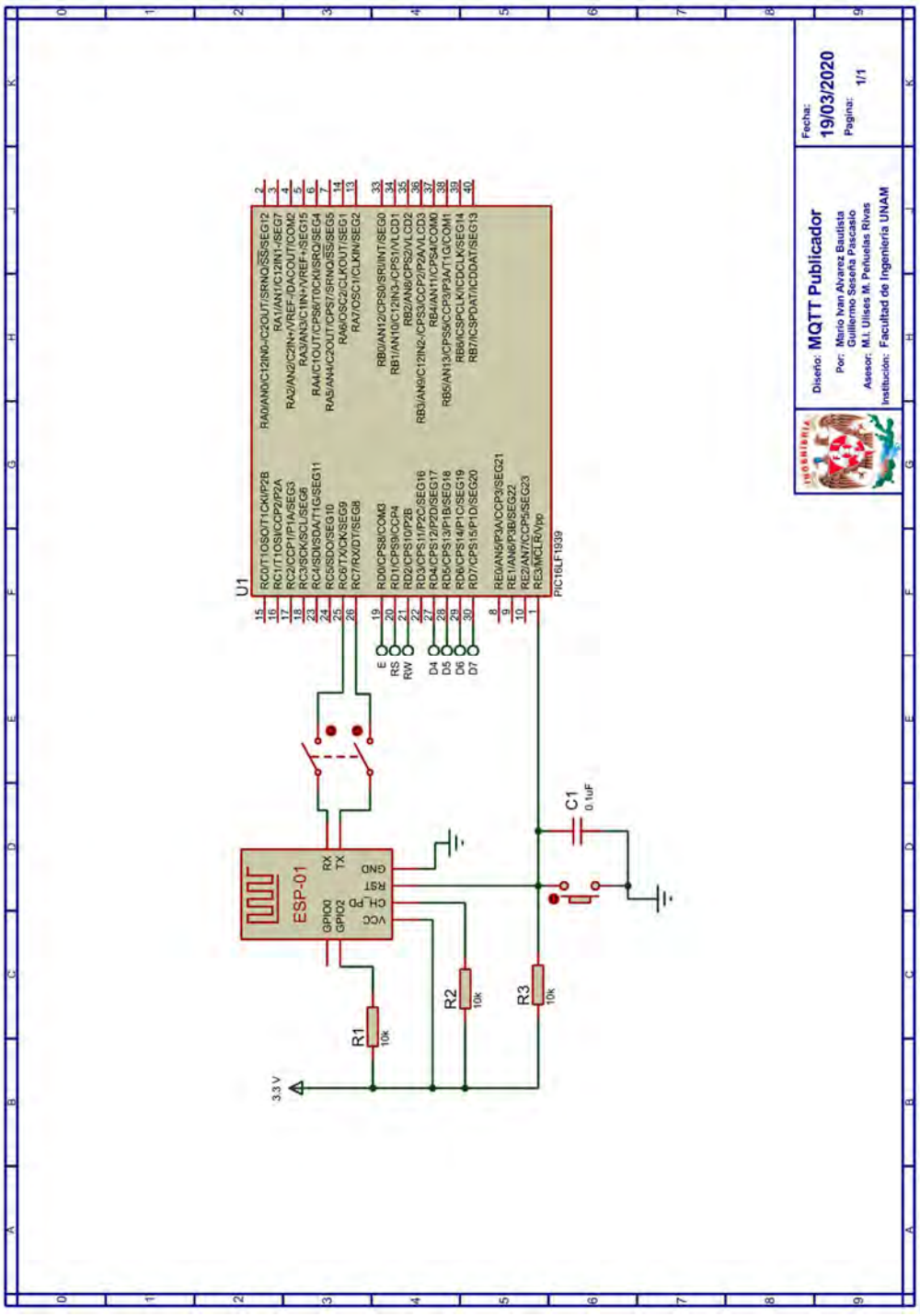


Figura P23-A.9 *Circuito de la práctica 23-A.*

Referencias

1. HiveMQ. *MQTT Essentials*. [Citado 2020 Marzo]; Disponible en:https://www.hivemq.com/mqtt-essentials/.
2. International Business Machines Corporation (IBM). *MQTT V3.1 Protocol Specification*. 2010 [Citado 2020 Marzo]; Disponible en:http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.
3. OASIS Standard. *MQTT Version 3.1.1*. 2014 [Citado 2020 Marzo]; Disponible en:http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

Práctica 23-B MQTT Suscriptor

Introducción

La biblioteca ESP permite establecer al módulo como un suscriptor de uno o varios temas, seleccionando para cada uno de éstos la calidad máxima con la que recibe los mensajes.

La biblioteca ESP al recibir un mensaje PUBLISH lo almacena en un *buffer*. Para que la carga útil pueda ser extraída del *buffer*, se tiene que completar con el mecanismo de entrega que especifica el protocolo. La biblioteca proporciona una función que atiende la recepción de mensajes, revisa si hay un nuevo mensaje PUBLISH almacenado en el *buffer*, si el mensaje cuenta con calidad de servicio igual a cero (QoS0) entonces lo marca como “listo” para su extracción, pero si el mensaje tiene QoS1 o QoS2, entonces completa el mecanismo de entrega que implica el intercambio de mensajes con el bróker, una vez concluido el procesamiento del mensaje, éste es marcado como “listo”.

Cuando un mensaje PUBLISH ya haya sido procesado, se puede extraer su carga útil, para ello el usuario puede solicitarlo para un tema en específico o extraerlo para cualquier tema. Para un tema en específico, la biblioteca revisa que en el mensaje almacenado en el *buffer* esté destinado al tema que se solicita, si es así, entonces lo copia a un *string* que el usuario proporciona, de no ser así, no modifica el *string* del usuario. Mientras que, para cualquier tema, la biblioteca extrae la carga útil sin importar a que tema esté destinado.

Ya que los mensajes PUBLISH son almacenados en el *buffer* durante la interrupción por recepción de datos, es posible que un mensaje sea sobrescrito antes de ser procesado, esto depende de la frecuencia con la que se está recibiendo mensajes PUBLISH y cuánto tiempo demora el programa en llamar la función de procesamiento. Para mensajes con QoS1 y QoS2, no existe mayor problema porque su misma especificación permite el envío de duplicados hasta que el cliente le indique al bróker que lo ha recibido, que es lo que sucede durante el procesamiento del mensaje en la biblioteca. Sin embargo, para mensajes con QoS0 que no cuentan con mecanismo de aseguramiento es posible la perdida de información si en el programa del PIC se especifican tareas que demoren más tiempo que el intervalo de llegada entre mensajes, naturalmente esto es acorde con lo que especifica el protocolo al utilizar este nivel de calidad.

Objetivo

Establecer al módulo como un cliente MQTT que funcione como suscriptor utilizando la plataforma CloudMQTT.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se

puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_MQTT_v31

Materiales

- 1 computadora
- 1 dispositivo móvil Android/iOS
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 pantalla LCD 16x2
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 56 Ω

Descripción

En esta práctica el módulo será establecido como un suscriptor de dos temas, uno destinado al control de estado de un LED y otro para recepción de mensajes que serán mostrados en una LCD, esto se realizará utilizando la plataforma CloudMQTT.

En la página de CloudMQTT se crearán dos nuevos temas para el control del LED

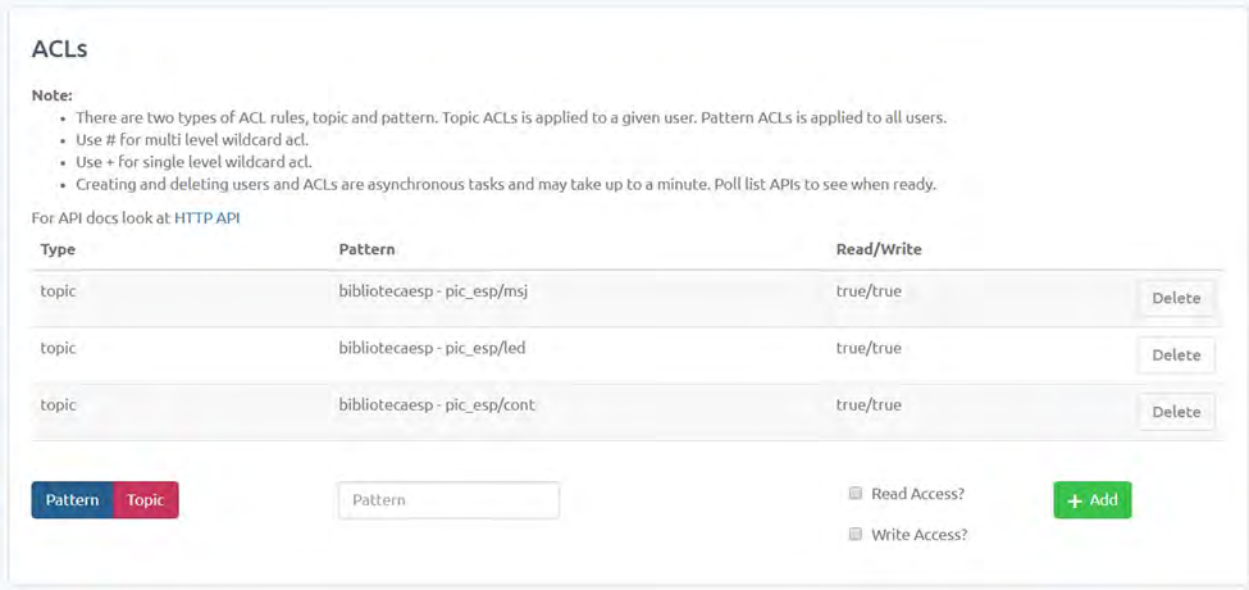


Figura P23-B.1 *Creación de temas para control de estado de un LED y envío de mensajes.*

se nombra “pic_esp/led” y para los mensajes como “pic_esp/msj” (ver figura P23-B.1).

El módulo usará un ID aleatorio colocado manualmente en el código, estableciendo un *keepalive* de 900 segundos. y se suscribe al tema de control de LED con QoS0, mientras que el otro tema con QoS1. La publicación de mensajes y en consecuencia del control del PIC-ESP se efectuará desde la sección “*WEBSOCKET UI*” de la misma plataforma o desde alguna aplicación móvil como IoT MQTT Panel.

Código

El código que deberá ejecutar el microcontrolador PIC en esta práctica es el mostrado en la figura P23-B.4 y en la figura P23-B.5, en cual se deben ingresar los siguientes parámetros:

- Nombre de un AP con acceso a internet
- Contraseña del AP
- Nombre de usuario
- Contraseña de usuario
- Nombre de tema de control de estado del LED
- Nombre de tema de recepción de mensajes

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P23-B.6.

Resultados

Para publicar información en los temas a los que suscribió el módulo, se deberá acceder a la página de la plataforma CloudMQTT y se ingresará a la opción de “*WEBSOCKET UI*” de la instancia utilizada. En la sección “*Send message*”, se escribirá el tema “pic_esp/led” y se enviará “*on*” para encender el LED y para apagar se enviará “*off*” para el mismo tema (ver figura P23-B.2).

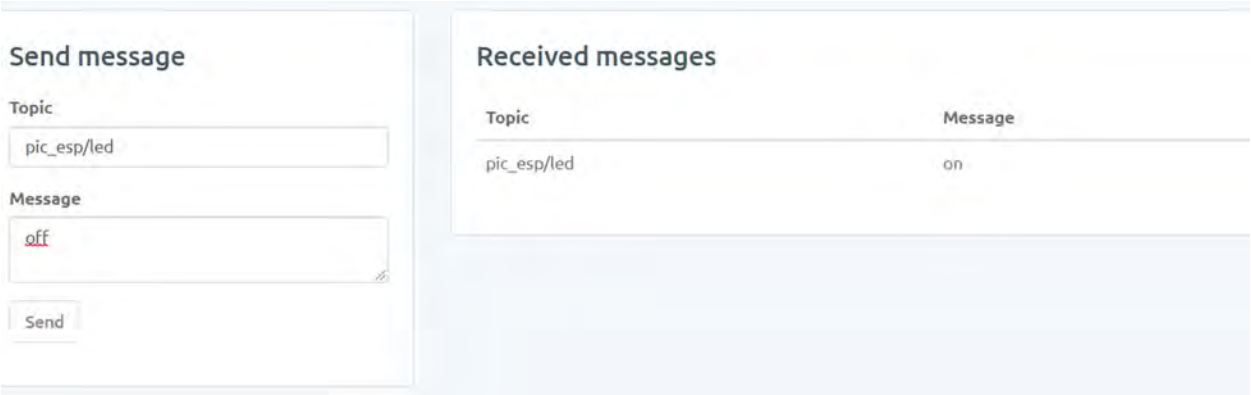


Figura P23-B.2 *Publicación de mensajes para controlar estado del LED.*

Cuando se envíe un mensaje para el tema “pic_esp/msj” éste se mostrará en la pantalla LCD.

La publicación de información se puede efectuar también desde alguna aplicación móvil como IoT MQTT Panel, que para ilustrar esta práctica se ha creado un panel, en el que se agrega un *Switch* y un *Text input*, que funcionan como publicadores para controlar el estado del LED y enviar mensajes a la LCD respectivamente (ver figura P23-B.3).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3diJDYS

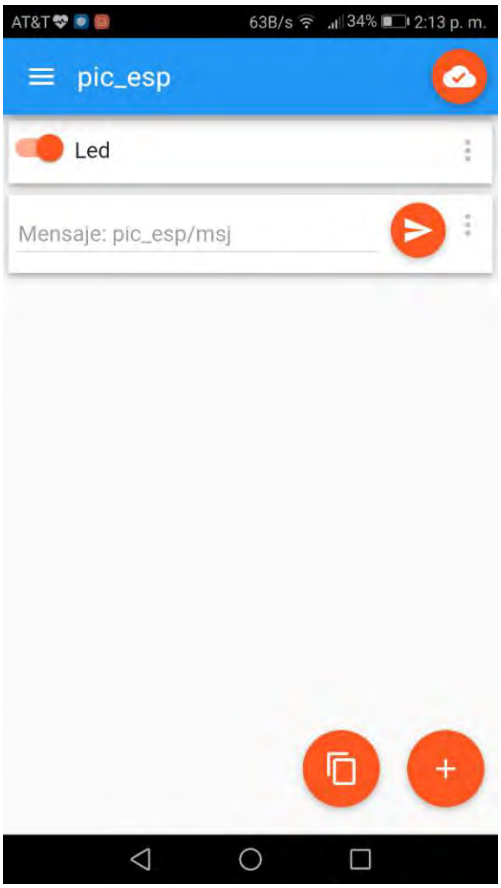


Figura P23-B.3 *Panel en IoT MQTT Panel para controlar LED y enviar mensajes a la LCD.*

```
#include <16lf1939.h>
#include "esp.h"
#include<lcd.c>

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define user_name "nombre de usuario"
#define user_password "contraseña de usuario"
#define topic_1 "nombre del tema"
#define topic_2 "nombre del tema"

int buffer_mqtt[30]={0};
int led_buffer[4]={0};
int msj_buffer[10]={0};

int aux0=0;
int aux1=0;
void main()
{
    /*Establece al PIN B7 como salida y al resto del puerto B
    como entradas*/
    set_tris_b(0b01111111);
    //Establece en cero todo el puerto B
    output_b(0);
    //Inicialización de la LCD
    lcd_init();
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Establece buffer de mensajes PUBLISH*/
    set_buffer_mqtt(buffer_mqtt,sizeof(buffer_mqtt));
    /*Se crea el cliente MQTT (se conecta al servidor de la plataforma)
    para el canal numero cuatro*/
    create_mqtt("m16.cloudmqtt.com",18236,4);
    /*El cliente se conecta a una cuenta de CloudMQTT.Se establece
    el ID de cliente MQTT como 9875 , un keepalive de 900s*/
    connect_mqtt("9875",user_name,user_password,900);
    //Se subscribe al tema indicado con QoS0
    subscribe_mqtt(topic_1,0);
    //Se subscribe al tema indicado con QoS1
    subscribe_mqtt(topic_2,1);
    while(1)
    {
        //Atiende la recepción de mensajes PUBLISH
        aux0=refresh_mqtt();
        /*Extrae la carga útil del mensaje PUBLISH para el tema
        que controla el estado del LED*/
        get_payload_mqtt(topic_1,led_buffer,sizeof(led_buffer));
        //Enciende LED
        if(led_buffer[0]=='o' && led_buffer[1]=='n')
            output_bit(PIN_B7,1);
        //Apaga LED
        else if(led_buffer[0]=='o' && led_buffer[1]=='f' && led_buffer[2]=='f')
            output_bit(PIN_B7,0);
        /*Extrae la carga útil del mensaje PUBLISH para el tema
        de mensajes a mostrar en la LCD*/
```

Figura P23-B.4 *Primera parte del código de la práctica 23-B.*

```
        if(aux0==0 && aux1==0)
        {
            //Limpia la LCD
            printf(lcd_putc,"\f");
            //Posiciona escritura en la LCD
            lcd_gotoxy(5,1);
            //Imprime en la LCD
            lcd_putc("Mensaje:");
            //Posiciona escritura en la LCD
            lcd_gotoxy(1,2);
            //Imprime en la LCD
            printf(lcd_putc,"%s",msj_buffer);
        }
        delay_ms(20);
    }
}
```

Figura P23-B.5 *Segunda parte del código de la práctica 23-B.*

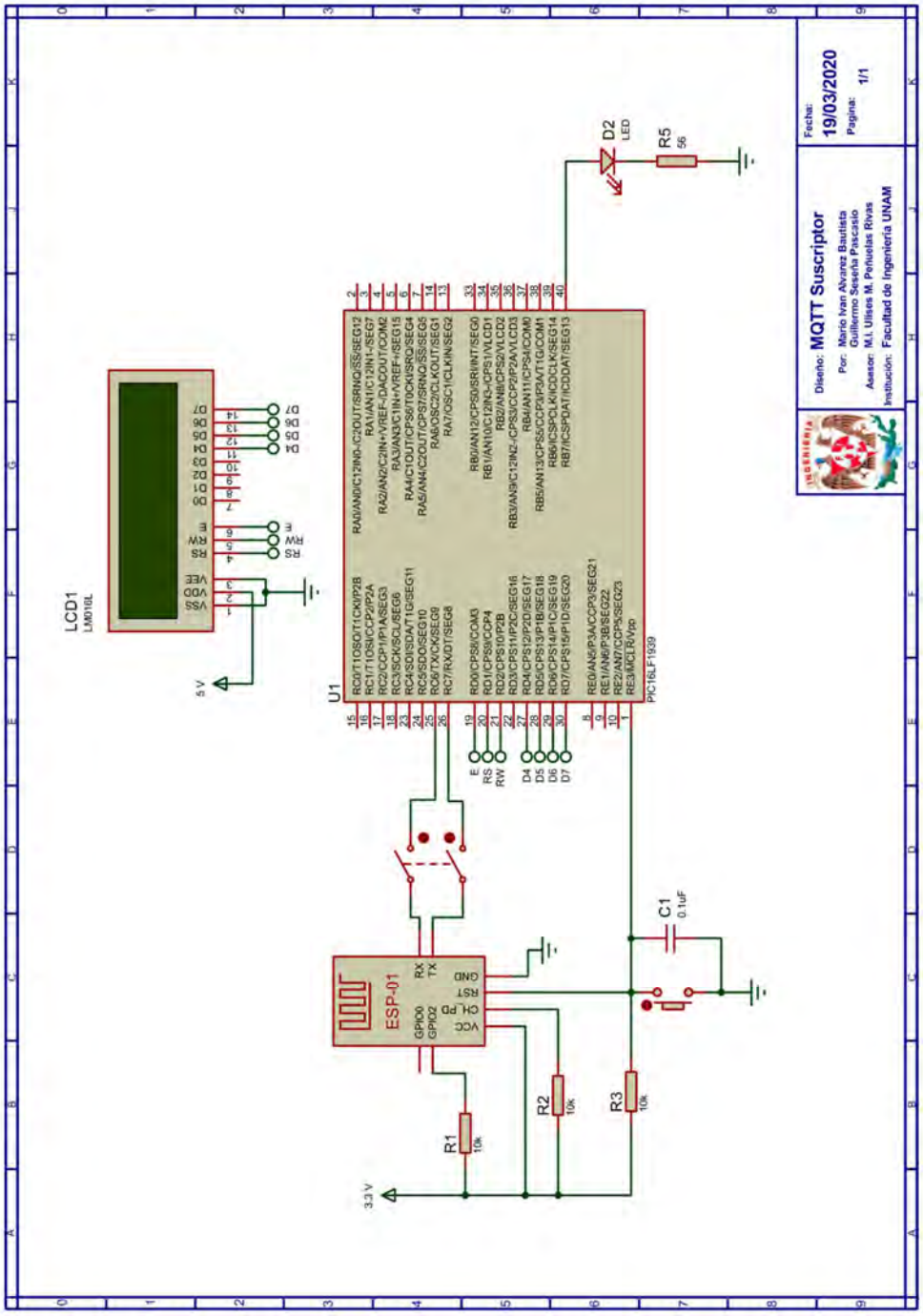


Figura P23-B.6 *Circuito de la práctica 23-B.*

Referencias

Para más detalles sobre MQTT consultar:

- HiveMQ. *MQTT Essentials*. [Citado 2020 Marzo]; Disponible en: https://www.hivemq.com/mqtt-essentials/

Práctica 24-A Plataformas IoT: *thingspeak*

Introducción

Las plataformas IoT son aquellas a las cuales los dispositivos envían y/o reciben información para su almacenamiento, visualización y procesamiento, con el fin darle utilidad a dicha información. Estas plataformas ofrecen servidores a los cuales los dispositivos se conectan para intercambiar información utilizando protocolos de aplicación como MQTT, HTTP y CoAP.

En la actualidad existe un mercado vasto de opciones, algunas de ellas son parcialmente gratuitas, algunas están enfocadas a la implementación a gran escala y otras al desarrollo de prototipos. También existen plataformas que ofrecen APIs para el desarrollo de soluciones web e incluso algunas ofrecen integración con otros servicios en línea.

Thingspeak es una plataforma IoT de código abierto que integra Matlab para análisis y visualización de datos. Esta plataforma ofrece un bróker y una API basada en REST para que los dispositivos puedan comunicarse con ella mediante los protocolos MQTT y HTTP respectivamente.

Thingspeak se base en canales (*channels*), en los cuales se almacenan toda la información de una aplicación. Un canal está compuesto de hasta de ocho campos de datos (*fields*), los cuales pueden ser utilizados para recopilar información de uno o varios dispositivos, además de contener metadatos como la ubicación de la fuente de información.

A cada uno de los canales se le asigna un ID (*Channel ID*), una clave de lectura (*Read API Key*) y escritura (*Write API Key*) para que los dispositivos puedan hacer uso dichos canales. Además, se les otorga una vista privada (*Private View*) y una vista pública (*Public View*), en las cuales se puede agregar gráficos y widgets para visualizar la información recopilada y procesada[1].

Como ya se ha mencionado, *Thinspeak* integra Matlab por lo que es posible elaborar códigos que accedan a la información almacenada en sus canales para su análisis y visualización, e incluso es posible generar nuevos datos a partir de la ejecución de estos códigos. También, en función de la información recopilada, *Thinspeak* puede desencadenar acciones como: ejecutar cálculos en Matlab; registrar datos selectivamente; enviar peticiones HTTP que activen acciones dentro de la plataforma o en algún servicio web externo; activar alertas; e incluso publicar tuits automáticamente.

Thingspeak ofrece distintos tipos de cuentas, la más básica es gratuita y ofrece: hasta 3 millones de mensajes en un año, envío/recepción de mensajes

con un intervalo minino de 15 segundos, 4 canales máximos y suscripción simultánea de hasta 3 temas en MQTT.

Así como muchas otras plataformas IoT, *Thingspeak* ofrece más de una forma de comunicarse con los dispositivos. Esta flexibilidad busca adecuarse mejor a las necesidades de los usuarios. Por ejemplo, MQTT es mucho más liviano que HTTP, por lo que el uso de MQTT es más conveniente en dispositivos con capacidad computacional reducida. Además, si se busca una aplicación más cercana a la comunicación en tiempo real, entonces MQTT es una mejor opción, ya que el dispositivo mantiene la conexión abierta, obteniendo información casi inmediatamente después de su publicación, caso contrario en HTTP que es necesario realizar una solicitud de manera periódica. Sin embargo, utilizar HTTP puede ser útil en aplicaciones que no requieren mantener una conexión abierta con el servidor, puesto que la conexión solo puede ser utilizada durante el envío y/o consulta de información, también es conveniente en aplicaciones que intercambian información en tiempos claramente definidos.

El módulo ESP puede comunicarse con esta plataforma, utilizando el protocolo HTTP o MQTT. Para HTTP, la biblioteca ESP permite el envío de información mediante peticiones, sin embargo, para consultar información no ofrece una opción óptima, ya que se tiene que almacenar todo el contenido de la respuesta y extraer de ella información útil, ante esta situación se recomienda el uso de HTTP exclusivamente para el envío de información a esta plataforma. Mientras que para MQTT, la biblioteca ESP permite él envío y/o recepción de información al margen de su adaptación del protocolo MQTT en su versión 3.1.1.

Objetivo

Comunicar el módulo ESP con la plataforma *Thingspeak* utilizando el protocolo HTTP y MQTT.

***Firmware* de comandos AT**

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_HTTP_CLIENT para la primera actividad

- ACCESS_POINT y E_MQTT_V311 para la segunda actividad

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 capacitor de 0.1 µF
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 55 Ω
- 1 potenciómetro de 10 kΩ

Descripción

En esta práctica el módulo ESP se comunicará con la plataforma *Thingspeak* para enviar lecturas de una señal analógica generada por la variación de la posición del eje de un potenciómetro que está conectado a una de las entradas analógicas del microcontrolador PIC. En una aplicación real esta señal puede provenir de algún sensor, pero para esta práctica sólo se muestra el proceso para comunicar el módulo a la plataforma.

Además, se controlará el estado de un LED en función a los valores registrados en la plataforma. Este control se realizará mediante aplicaciones creadas en *Thingspeak*, las cuales comparan las lecturas recibidas con un valor definido como límite, al sobrepasar éste se enviará información al módulo para encender el LED, mientras que cuando las lecturas están por debajo del límite, se enviará información para apagarlo.

La práctica se ha divido en dos actividades, en la primera el módulo únicamente enviará valores de la señal analógica mediante el protocolo HTTP. Como ya se ha mencionado es posible consultar información de la plataforma mediante este protocolo, sin embargo, es más complicado ya que se debe capturar todo el contenido de la respuesta HTTP que incluye metadatos. No obstante, al final de la práctica se muestra un enlace, en el cual se ha agregado un código para consultar información del control del LED, este código obtiene toda la respuesta de la petición y la imprime en una terminal.

En la segunda actividad el módulo se comunicará con la plataforma utilizando el protocolo MQTT, por lo que el módulo publica los valores leídos por el PIC y se suscribe a un tema para controlar el estado de un LED.

El primer paso por realizar en esta práctica es crear una cuenta gratuita en *Thingspeak* accediendo a la página:

https://thingspeak.com/ .

Como se utiliza una cuenta gratuita, el intercambio de información está limitado cada 15 segundos por canal, como en esta práctica se enviará y se recibirá

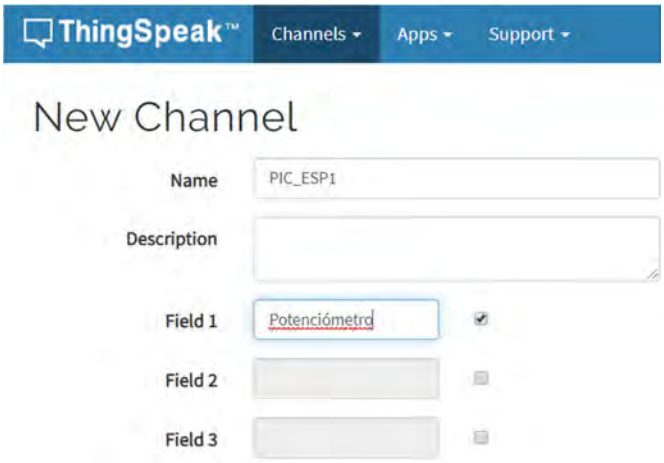


Figura P24-A.1 *Creación de un canal (Channel)*

información de la plataforma, es posible que se sobrepase lo establecido para este tipo de cuenta. Para evitar esto, se crean dos canales, uno para registrar lecturas cada 15 segundos y otro para controlar el estado de un LED. Se podría utilizar un solo canal y utilizar dos campos para este fin, sin embargo, se debe colocar un intervalo de registro de lecturas de más de 30 segundos para garantizar no superar el límite.

Para crear el primer canal asociada al registro de lecturas, en la parte superior de la página se oprime la opción de "*Channels*" y después "*My Channels*", posteriormente se presiona "*New Channel*", en esta opción se llenan los datos del canal, basta con colocar un nombre al canal y una etiqueta el *Field 1* (Campo 1), tal como se muestra en la figura P24-A.1, en ésta la etiqueta del campo se ha llamado potenciómetro en alusión al origen de la señal que la produce.

Una vez creado el primer canal se mostrará la pestaña de *Private View* (vista privada), en la que por defecto se agrega una gráfica para el campo 1 (ver figura P24-A.4). A cada canal se le asigna un ID (*Channel ID*) que es utilizado para acceder a él.

Figura P24-A.2 *Selección del Lamp Indicator.*

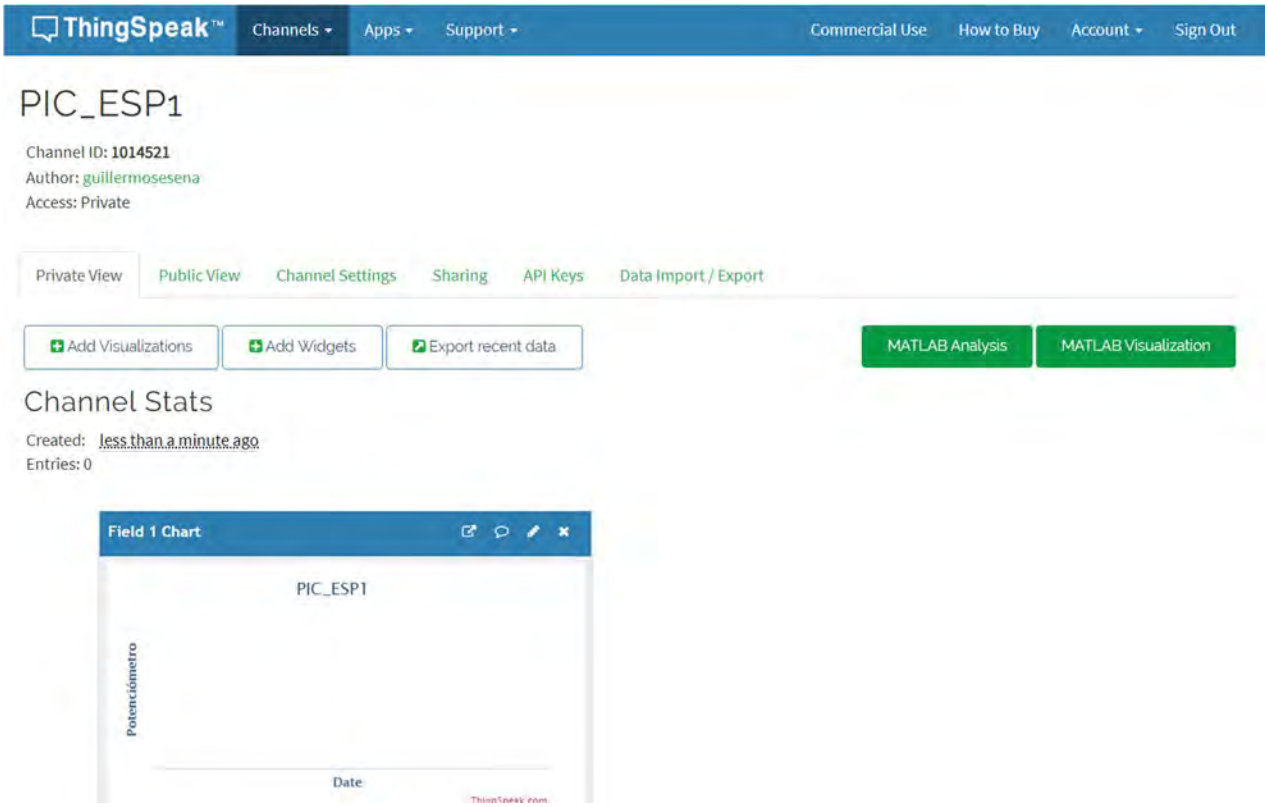


Figura P24-A.4 *Private View de un canal en Thingspeak.*

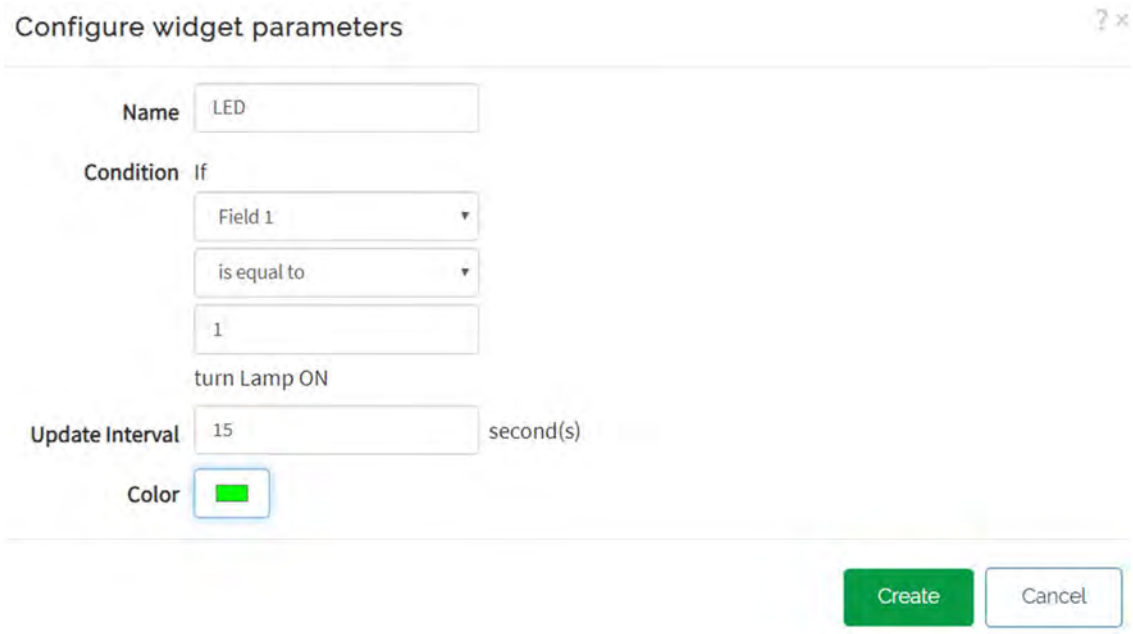


Figura P24-A.3 *Configuración del Lamp Indicator.*

Después se crea un segundo canal, al cual su *Field 1* se destina para el control del LED. En *Private View* del segundo canal se agrega un indicador de estado del LED, por lo que se oprime "*Add Widgets*" y después "*Lamp Indicator*" (ver figura P24-A.2), el siguiente paso es establecer que el indicador se active cuando el valor del *Field 1* sea igual a uno, tal como se muestra en la figura P24-A.3. De esta forma, el LED se encenderá cuando este campo tenga un valor unitario.

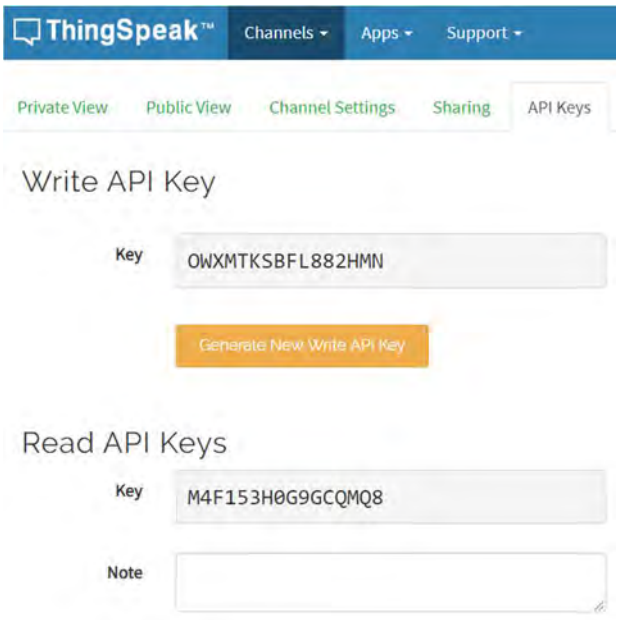


Figura P24-A.5 *Ejemplo de una API Keys.*

Cuando los dos canales estén listos, se tiene que consultar las claves API para que el módulo pueda hacer uso de dichos canales. Para cada canal se selecciona la pestaña de "*API Keys"* en la que muestra tanto la clave de escritura (*Write API Key*) como la de lectura (*Read API Key*) (ver figura P24-A.5).

I Envío de información mediante HTTP

Para la primera actividad, el módulo se conectará al servidor api.thingspeak.com en el puerto 80 (TCP) y registrará las lecturas enviando peticiones HTTP mediante el método GET. En dichas peticiones el URL contiene la clave de escritura, el campo del canal destinado a registrar y el valor a registrar, tal como se muestra en la figura P24-A.6. El *keepalive* es enviado para que el módulo controle la desconexión y no para reutilizar la conexión.

II Envío de información mediante MQTT

Para la segunda actividad, el módulo se conectará al servidor mqtt.thingspeak.com en el puerto 1883 (TCP). El módulo que funciona como cliente MQTT, inicia sesión utilizando el ID de usuario (*User ID*) otorgado por

```
GET /update?api_key={write api key}&field1={valor de lectura} HTTP/1.1
Host: api.thingspeak.com
Connection: keep-alive
```

Figura P24-A.6 *Estructura de la petición HTTP para registro de lecturas de la primera actividad.*

Thingspeak y como contraseña se coloca la *MQTT API Key*, estas credenciales se consultan al presionar el icono de usuario y seleccionar "*My profile*" (ver figura P24-A.7). El módulo usará un ID aleatorio colocado manualmente en el código y se establece un *keepalive* de 60 segundos.

Las lecturas serán publicadas con QoS0 al tema con la estructura: channels/{id_del canal_de_lecturas}/publish/fields/field1/{ write_api_key}.

El módulo se suscribe al tema para controlar el LED con la forma: channels/{id_del_canal_del_led}/subscribe/fields/field1/{read_api_key}.

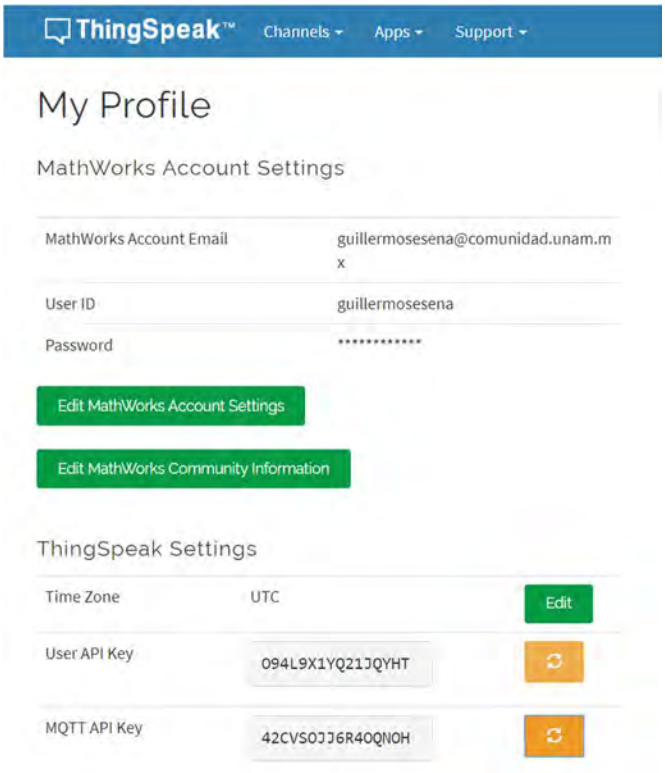


Figura P24-A.7 *Ejemplo del perfil de una cuenta de Thingspeak, donde se muestra nombre de usuario y las claves para ingresar a las APIs de la plataforma.*

En la plataforma Thingspeak se crean cuatro aplicaciones para controlar el LED. Las dos primeras son de tipo ThingHTTP, las cuales realizan peticiones HTTP para enviar información al canal y campo destinado al control del LED. Mientras que, las otras dos son de tipo *React*, las cuales comparan los valores de lecturas enviados con una condición definida, al cumplirse alguna de estas condiciones activan alguna de las aplicaciones ThingHTTP.

Para crear las aplicaciones ThingsHTTP se accede a la página de la plataforma, se selecciona "*Apps*", después "*ThingsHTTP*" y finalmente "*New ThingsHTTP*". La primera aplicación, se nombra como *Encender LED* y se configura una petición como la mostrada en la figura P24-A.6 pero utilizando la *Write API Key* del canal que controla el LED y un valor igual a uno para encender dicho LED (ver figura P24-A.8). La otra ThingHTTP, se nombra como *Apagar LED* y se configura una petición igual a la anterior, pero con la diferencia que el valor a registrar es el número cero.

Para crear las aplicaciones *Reacts*, se selecciona "*Apps*", después "*React*" y finalmente "*New React*". Para la primera aplicación se nombra como *supera límite* y se configura como una condición numérica durante la inserción de un dato,

dicha condición establece que, si el valor del campo y canal destinado a las lecturas es mayor o igual a 200, entonces ejecuta la ThingHTTP que enciende el LED, esta acción se ejecuta para el primer valor que supere dicha condición (ver figura P24-A.9).

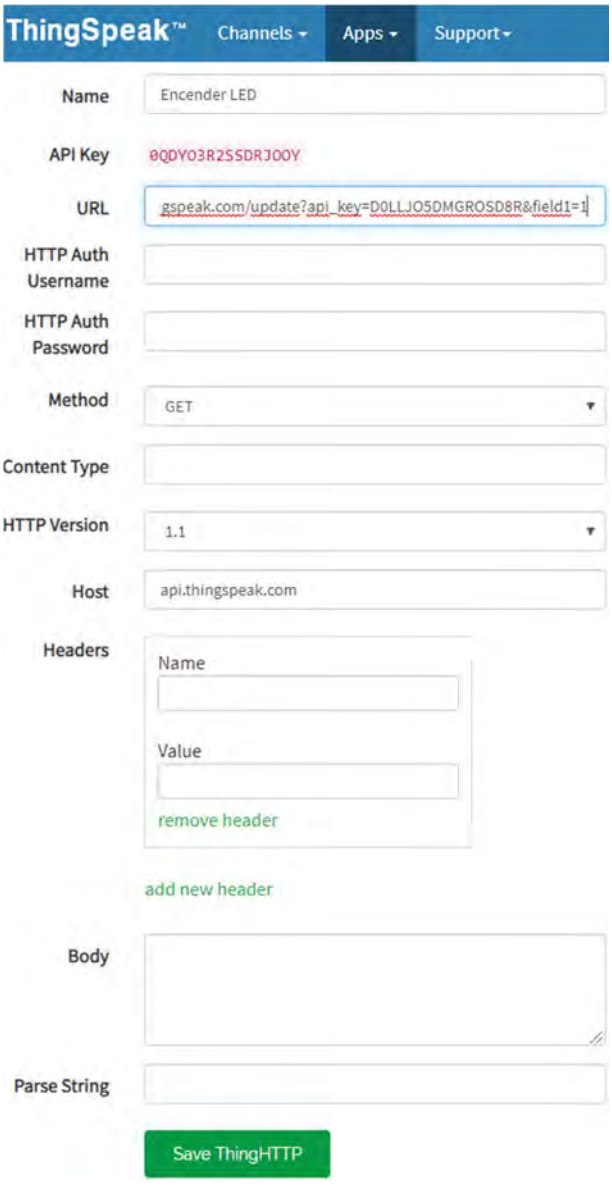


Figura P24-A.8 *ThingHTTP para encender LED.*

Código

El código que deberá ejecutar el microcontrolador PIC para la primera actividad es el mostrado en la figura P24-A.16, y para la segunda actividad es el mostrado en la figura P24-A.17. En ambas actividades se deberá ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet.

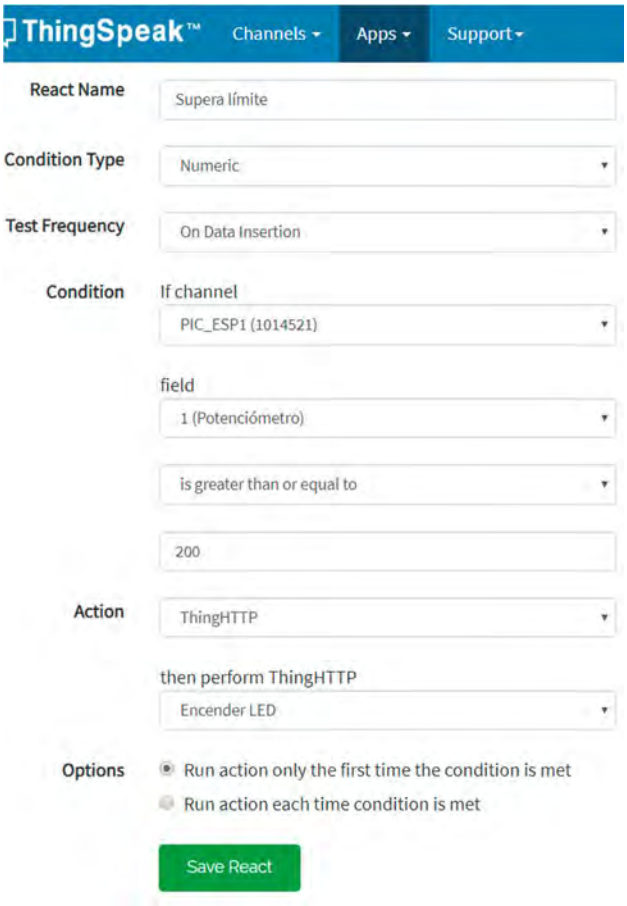


Figura P24-A.9 *React para encender LED.*

En la primera actividad se debe especificar la clave de escritura del canal al que se envían las lecturas, mientras que para la segunda actividad se ingresa:

- ID de usuario
- MQTT API Key
- ID y clave de lectura del canal destinado al control del LED, en el tema de suscripción
- ID y clave de escritura del canal reservado a las lecturas, en el tema de publicaciones

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P24-A.18.

Resultados

Para la primera actividad, en la página de la plataforma se accederá al canal de las lecturas y en la pestaña de "*Private View*" se observará que cada 15 segundos se gráfica un nuevo valor (ver figura P24-A.10).

Thingspeak permite utilizar Matlab para analizar y visualizar datos. Si se oprime "*MATLAB Visualizations*" se mostrarán plantillas y ejemplos de códigos para visualizar la información que se envían a la plataforma. Por ejemplo, si se selecciona "*Create a discrete sequence data plot*" se mostrará un código al cual solo basta con añadir el ID del canal, la

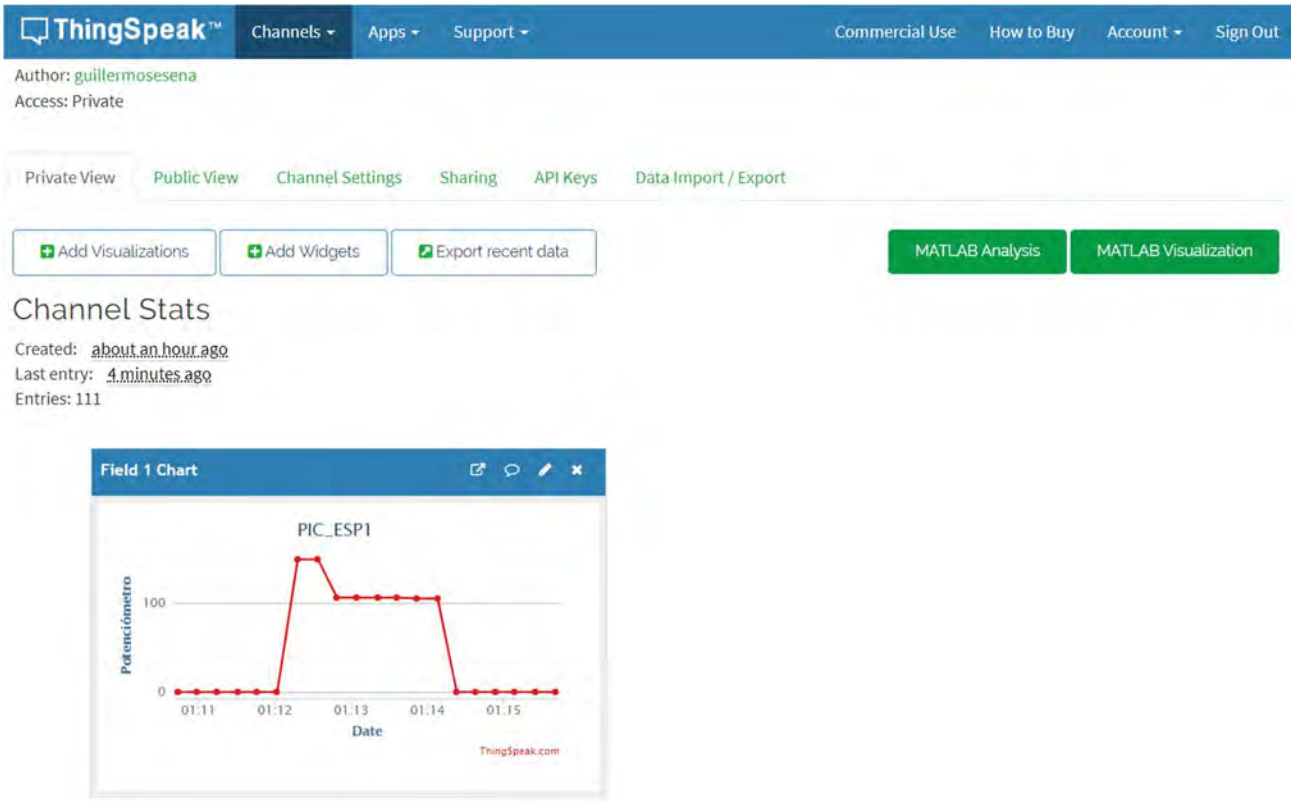


Figura P24-A.10 *Private View del primer canal para el campo de lecturas.*

clave de lectura y el ID del campo para utilizarlo (ver figura P24-A.11), al ejecutarlo se mostrará el gráfico, el cual puede ser agregado a la *Private View* (ver figura P24-A.13).

También *Thingspeks* proporciona ejemplos de códigos para analizar los datos enviados a la plataforma, por ejemplo, si se selecciona "*Calculate and display average humidity*", se mostrará un código al cual se ingresan la misma clase de parámetros utilizados en el ejemplo anterior y que al ejecutarlo se mostrará el promedio de las lecturas registradas en los últimos 60 minutos (ver figura P24-A.12).

Para la segunda actividad también se graficarán las lecturas que tome el microcontrolador PIC (ver figura P24-A.14), pero si se llega a un valor superior a 200 entonces se encenderá el LED y en el *Private View* del canal del LED, el indicador también se encenderá (ver figura P24-A.15), al bajar el valor 200 entonces el LED se apagará.

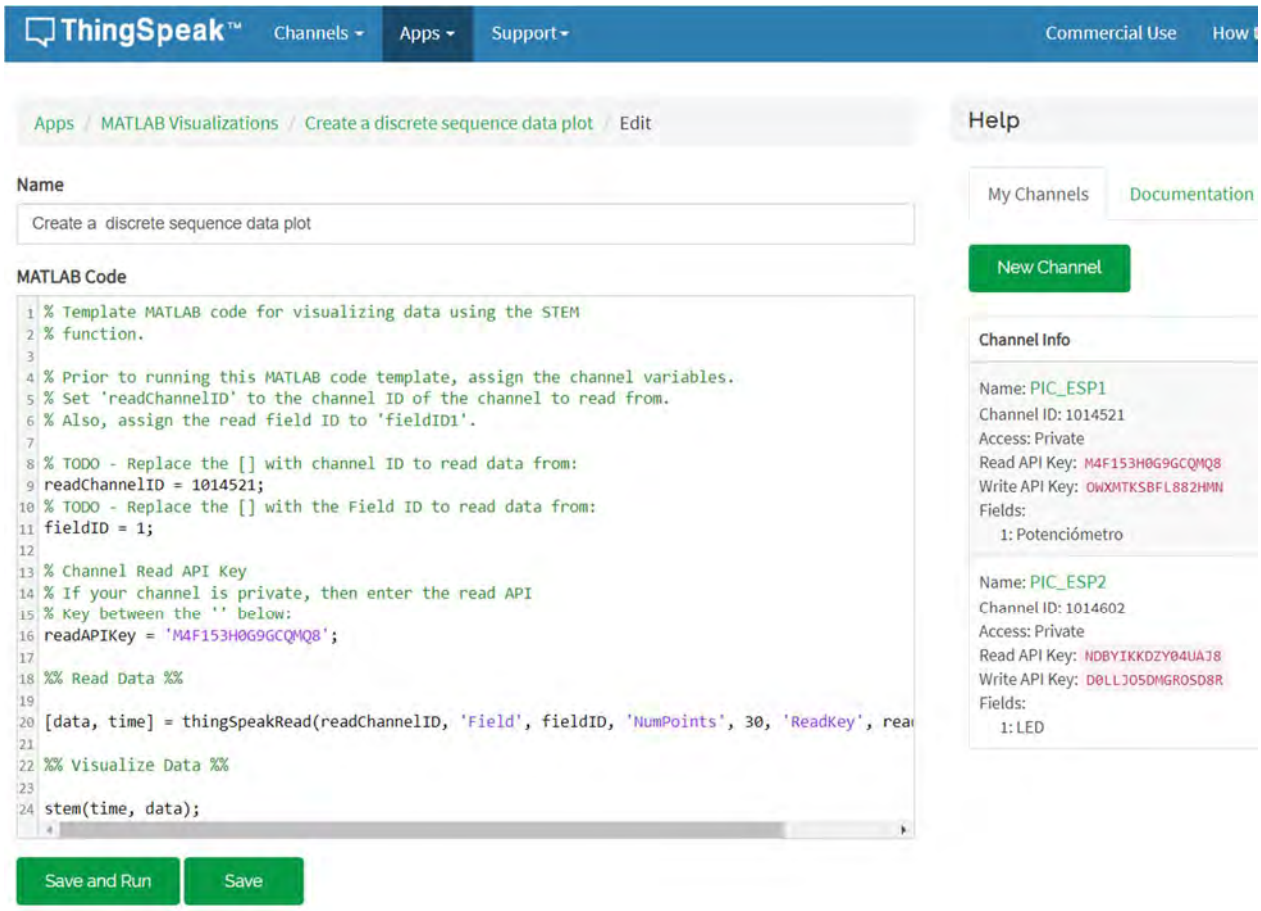


Figura P24-A.11 *Template que proporciona Thingspeak para graficar una secuencia discreta de datos.*

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2YSiK9e

Figura P24-A.13 *Private View del primer canal para el campo de lecturas con una gráfica discreta.*

Figura P24-A.12 *Salida de la ejecución de un código ejemplo que proporciona Thingspeak.*

Figura P24-A.15 *Private View cuando la lectura sobrepasa el valor límite de la aplicaciones React.*

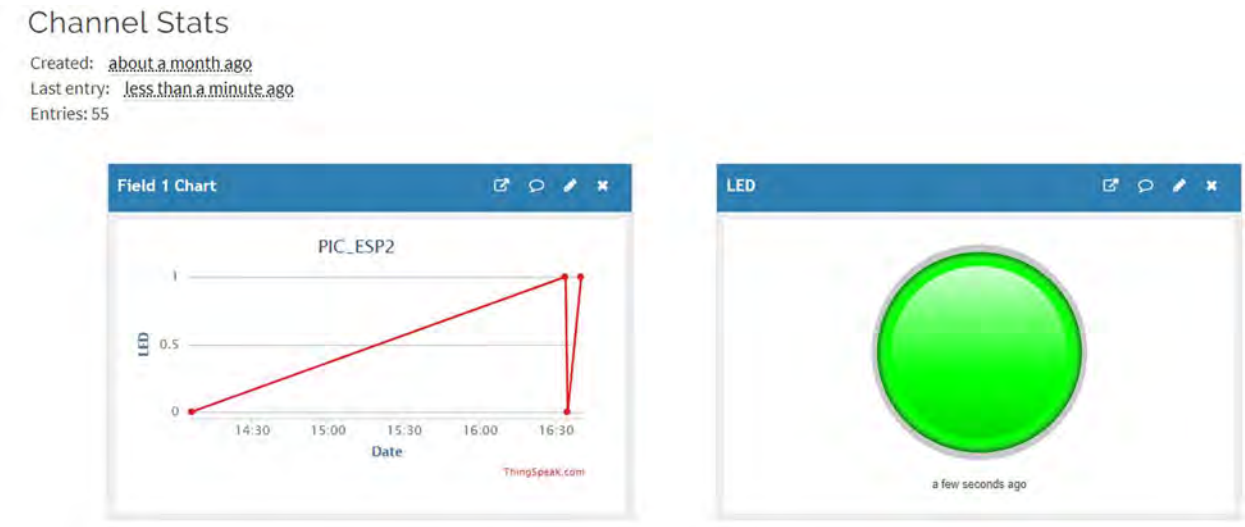


Figura P24-A.14 *Private View del segundo canal cuando se sobrepasa el valor límite de la aplicaciones React.*

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define write_api_key "clave de escritura"

char str_valor[4];
long codigo_estado_http=0;
long valor=0;

void main()
{
    setup_adc(adc_clock_div_64);
    delay_us(10);
    setup_adc_ports(sAN10);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    while(1)
    {
        //Obtención de valor
        set_adc_channel(10);
        valor=(read_adc()>>1);
        //Valor leído es colocado en un string
        sprintf(str_valor,"%04lu",valor);
        /* En el canal de comunicación 0, se establece un cliente TCP*/
        create_tcp_pclient("api.thingspeak.com",80,0);
        /*Se envía petición HTTP obtener registrar la lectura
        al campo número uno del canal especificado*/
        send_client(0,"GET /update?api_key=");
        send_client(0,write_api_key);
        send_client(0,"&field1=");
        send_client(0,str_valor);
        send_client(0," HTTP/1.1\r\n");
        send_client(0,"Host: api.thingspeak.com\r\n");
        codigo_estado_http=finish_request_http_client(0,"Connection: keep-alive\r\n\r\n");
        delete_client(0);
        delay_ms(15000);
    }
}
```

Figura P24-A.16 *Código de la primera actividad de la práctica 24-A.*

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define user_id "nombre de usuario"
#define mqtt_api_key "mqtt api key"
#define subscribe_topic "channels/{id canal led}/subscribe/fields/field1/{clave lectura}"
#define publish_topic "channels/{id canal lectura}/publish/fields/field1/{clave escritura}"

char buffer_mqtt[80]={0};
char led_buffer[45]={0};
char str_valor[5]={0};
long valor=0;

void main()
{
  setup_adc(adc_clock_div_64);
  delay_us(10);
  setup_adc_ports(sAN10);
  /*Establece al PIN B7 como salida y al resto del puerto B
  como entradas*/
  set_tris_b(0b01111111);
  //Establece en cero todo el puerto B
  output_b(0);
  //Inicialización del módulo Wi-Fi
  begin_esp();
  //Conexión al punto de acceso indicado
  begin_station("Estacion",SSID,password);
  /*Establece buffer de mensajes PUBLISH*/
  set_buffer_mqtt(buffer_mqtt,sizeof(buffer_mqtt));
  /*Se crea el cliente MQTT (se conecta al servidor de la plataforma)
  utilizando el puerto TCP y el canal 4*/
  create_mqtt("mqtt.thingspeak.com",1883,4);
  /*El cliente se conecta a una cuenta de Thingspeak.Se establece
  el ID de cliente MQTT como 4001 y un keepalive de 60s*/
  connect_mqtt("4001",user_id,mqtt_api_key,60);
  subscribe_mqtt(subscribe_topic,0);
  while(1)
  {
      //Obtención de valor
      set_adc_channel(10);
      valor=(read_adc()>>1);
      //Valor leído es colocado en el string que se envía
      sprintf(str_valor,"%04lu",valor);
      //Atiende la recepción de mensajes PUBLISH
      refresh_mqtt();
      /*Extrae la carga útil del mensaje PUBLISH para el tema
      que controla el estado del LED*/
      get_payload_mqtt(subscribe_topic,led_buffer,sizeof(led_buffer));
      //Enciende LED
      if(compare_string("1",led_buffer))
        output_bit(PIN_B7,1);
      //Apaga LED
      else if(compare_string("0",led_buffer))
        output_bit(PIN_B7,0);
      /*Se publica el valor leído para el tema seleccionado*/
      publishQoS0_mqtt(publish_topic,str_valor);
      delay_ms(15000);
  }
}
```

Figura P24-A.17 *Código de la segunda actividad de la práctica 24-A.*

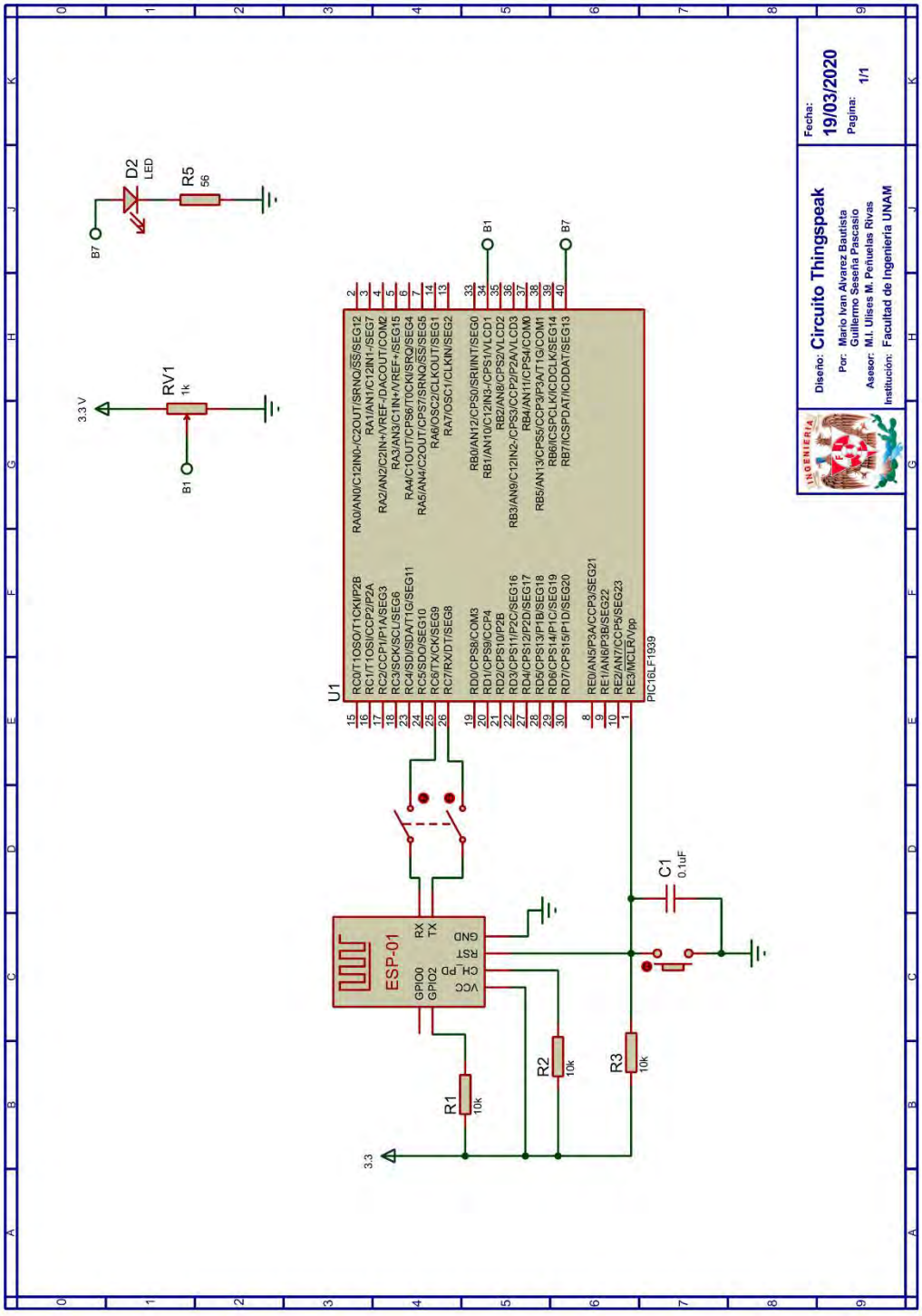


Figura P24-A.18 *Circuito de la práctica 24-A.*

Referencias

1. *Thingspeak*. Disponible en:https://thingspeak.com/.

Práctica 24-B Plataformas IoT: thingsboard

Introducción

Thingsboard es una plataforma IoT de código abierto, que permite a los dispositivos comunicarse mediante los protocolos MQTT, HTTP y CoAP. Esta plataforma ofrece dos ediciones, una es la *ThingsBoard Community Edition* que es totalmente gratuita y otra es *ThingsBoard Professional Edition* que requiere de la compra de alguno de sus planes de suscripción [1].

La versión gratuita comparada con otras plataformas parcialmente gratuitas permite acceder a una mayor cantidad de características, ya que permite gestionar, clientes (usuarios), bienes y dispositivos, para modelar escenarios reales. También ofrece límites de transferencia mucho más flexibles, con MQTT se ha alcanzado hasta más de 100 mensajes por minuto. Sin embargo, la información que se envía a la plataforma es temporal para esta versión, ya que no es almacenada en alguna base de datos.

No obstante, esta versión permite acceder a características muy interesantes para adentrarse al mundo IoT. Una de las más relevantes es la introducción a cadenas de reglas, esta permite desencadenar acciones a partir de telemetría (datos provenientes de los dispositivos) o de eventos. Algunas de las acciones que se puede realizar son el enviar correos electrónicos mediante SMTP, crear alarmas, ejecutar peticiones HTTP para servicios web externos, enviar información con MQTT, además de realizar llamadas de procedimiento remoto (*Remote Procedure Calls,* RPC) para controlar eventos internos o dispositivos.

También esta plataforma permite la elaboración de paneles, en los cuales se agregan widgets para visualizar y controlar a los dispositivos conectados. Para saber más de esta plataforma se recomienda consultar [1].

La comunicación entre el módulo ESP y esta plataforma utilizando la biblioteca ESP solo fue verificada para MQTT en su versión 3.1.1. Sin embargo, la biblioteca ESP proporciona funciones para ejecutar libremente peticiones HTTP, por lo que es posible el envío de información a la plataforma, mientras que para la consulta información, se tendría la misma dificultad que ya se explicaba en la práctica 24-A, que implicaba almacenar todo el contenido de la respuesta y extraer de ella la información útil.

Objetivo

Comunicar el módulo ESP con la plataforma Thingsboard utilizando el protocolo MQTT.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se

puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_HTTP_CLIENT para la primera actividad
- ACCESS_POINT y E_MQTT_V311 para la segunda actividad

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 capacitor de 0.1 µF
- 2 LED de 2.2V y 20 *mA*
- 2 resistores de 55 Ω
- 1 potenciómetro de 10 kΩ

Descripción

Esta práctica mostrará sólo algunas de las características que proporciona la plataforma *Thingsboard* y como el módulo ESP se comunica a ésta mediante MQTT. El PIC-ESP enviará lecturas de una señal analógica, generada por la variación de la posición del eje de un potenciómetro que está conectado a una de las entradas analógicas del microcontrolador PIC. Además, desde esta plataforma se controlará el estado de dos LED.

Las lecturas enviadas a la plataforma se visualizarán en un panel que contendrá una gráfica en tiempo real, junto con un *display* que muestra la última lectura registrada. En ese panel se agregará un interruptor para encender o apagar uno de los LED y se añadirá un tablero de alarmas.

Se creará una cadena de reglas, que cuando el valor de las lecturas supere un límite establecido, activará una alarma. Al activarse ésta, se enviará un correo electrónico mediante SMTP, indicando que se ha excedido el valor y se ejecutará una RPC para encender el otro de los LED, el cual se apagará hasta que las lecturas vuelvan a valores por debajo del límite definido.

El primer paso por realizar en esta práctica es crear una cuenta en la edición gratuita de Thingsboard, accediendo a la página

https://demo.thingsboard.io/ .

Ya con la cuenta creada, se selecciona el apartado de "*dispositivos"* en la que se mostrará una lista de los dispositivos que están listos para usarse. Para esta práctica se creará uno nuevo, por lo que se presiona la opción de "*agregar dispositivo"* que se encuentra con un signo más. El dispositivo por crear se nombra como PIC_ESP (ver figura P24-B.2).

Figura P24-B.2 *Creación de un dispositivo en Thingsboard.*

Una vez creado aparecerá en la lista de dispositivos y al presionar su nombre, se mostrarán los detalles de éste y es aquí donde se encuentra la opción de "*Gestionar credenciales*" que, al presionarla, se mostrará un *token* de acceso (ver figura P24-B.1), el cual servirá para que el cliente MQTT inicie sesión.

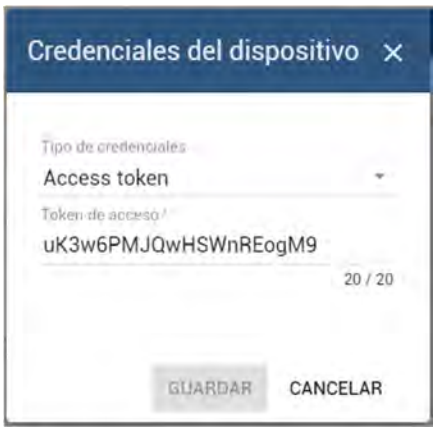


Figura P24-B.1 *Credenciales de un dispositivo.*

Ahora se creará un panel para el dispositivo PIC_ESP, por lo que se accede a la sección de "*paneles*" y se presiona agregar uno nuevo. Al crearse el panel, éste se mostrará en la lista y al presionar sobre éste aparecerán sus detalles entre los cuales está la opción de abrir el panel, la cual permite visualizar y modificar su contenido. Al presionar la opción anterior, se indica que el panel aún no contiene *widgets*.

Primero se agregará el *widget* de la gráfica, para esto se debe ingresar al modo edición al presionar un icono de color naranja. Posteriormente se selecciona "*agregar un nuevo widget*" y después "*crear nuevo widget*", se exhibirán todos los paquetes disponibles. Las gráficas se encuentran en el paquete *Char* y se selecciona el *widget Times series-flot.* Al agregar el *widget*, se abrirá una ventana que requiriere especificar un origen de datos. Se selecciona "*Entidad*"

como tipo y para el recuadro de alias se escribe "*datos_pic_esp*", posteriormente aparecerá la opción de crear este nuevo alias. Al crearlo se abre una ventana en la que se selecciona entidad única como tipo de filtro y se selecciona el dispositivo anteriormente creado (ver figura P24-B.4).

Una vez agregado el alias, en el recuadro de claves se escribe "valor_pot" en alusión a las lecturas de la señal producida por la variación de un potenciómetro, se oprime la tecla '*Enter*' (ver figura P24-B.3), y se termina de configurar agregando títulos o modificando la apariencia del *widget.*

Después de agregar la gráfica, se agrega el *widget* que fungirá como *display* de la última lectura registrada, para esto se selecciona del paquete de *Cards* (cartas) a *Temp* que se encuentra en la clase de

Figura P24-B.4 *Agregar alias de una entidad.*

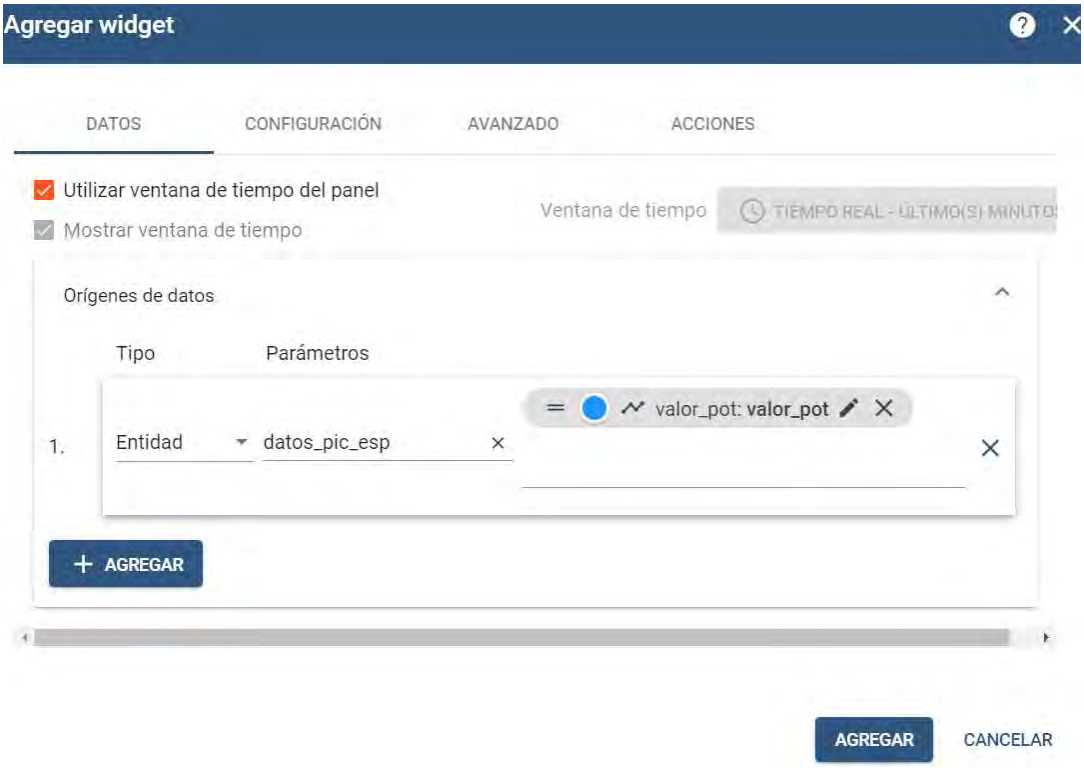


Figura P24-B.3 *Origen de datos de un widget.*

últimos valores. Al agregar el *widget* se solicitará un origen de datos, se especifica el mismo origen que el de la gráfica, esta vez el alias de la entidad ya aparecerá como una opción y se configura el nombre y apariencia.

Para agregar el tablero de alarmas se selecciona el paquete *Alarm widget* (Widget de alarma) y se selecciona el widget disponible. Al agregarlo se solicitará un origen, para el cual se coloca la entidad anteriormente creada (ver figura P24-B.5).

Al panel se agrega un interruptor *Round switch* del paquete de *Control widgets*. Cuando se agregue surgirá una ventana, que en la pestaña de datos se colocará como entidad a "datos_pic_esp", mientras que, en la pestaña de "*avanzado*", se modifica el nombre del valor del método a LED (ver figura P24-B.6).

Así cada vez que se accione el interruptor se ejecutará una RPC para el método llamado LED cuyo parámetro será verdadero o falso de acuerdo al valor del interruptor, por lo que el módulo recibirá mensajes con la siguiente forma: {"method":"LED","params":true/false}.

Finalmente se aplican los cambios al panel, teniendo como resultado algo semejante a lo mostrado en la figura P24-B.7.

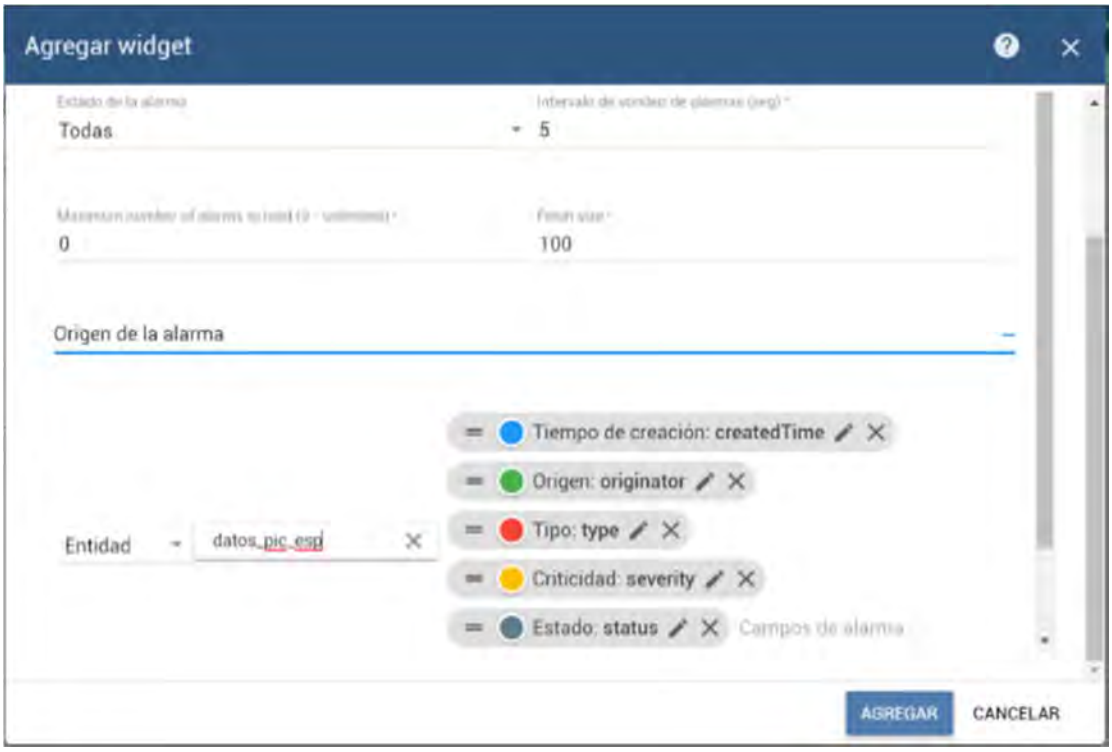


Figura P24-B.5 *Origen de alarma.*

Ahora se construye la cadena de reglas, por lo que se accede al apartado de "*cadenas de reglas*" y en éste se crea una nueva que se llamará "*Alarma*". Se abre dicha cadena y se mostrará el espacio de trabajo que contiene un nodo *input* (entrada) y al costado izquierdo se listan todos los nodos disponibles.

El primer nodo por agregar es un *script* que establece una condición numérica, el cual compara que los valores de lectura tengan un valor igual o mayor a un límite, que para esta práctica se estableció en 200 (ver figura P24-B.8). Este nodo se conecta a la salida del nodo *input*.

El resultado del *script* del límite será verdadero o falso. Para el caso verdadero, significa que la lectura ha superado el valor y por ello hay que activar una alarma, entonces se agrega un bloque de tipo *create alarm* (crear alarma). En este nodo se establece los parámetros de la alarma. El tipo de alarma se indica como "Alarma pot", se configura el nivel de severidad en alto y se señala que la alarma se propaga (ver figura P24-B.10). Este bloque se conecta a la salida del *script* límite mediante la etiqueta de verdadero.

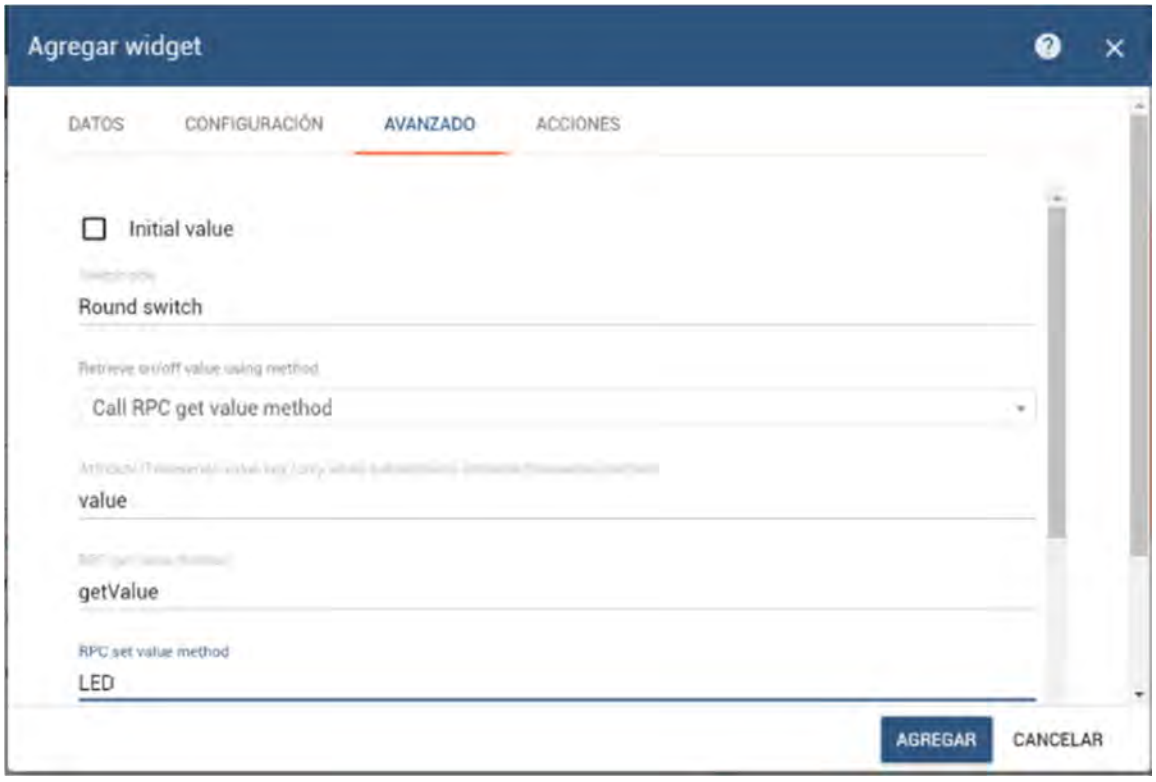


Figura P24-B.6 *Configuración del Round switch.*

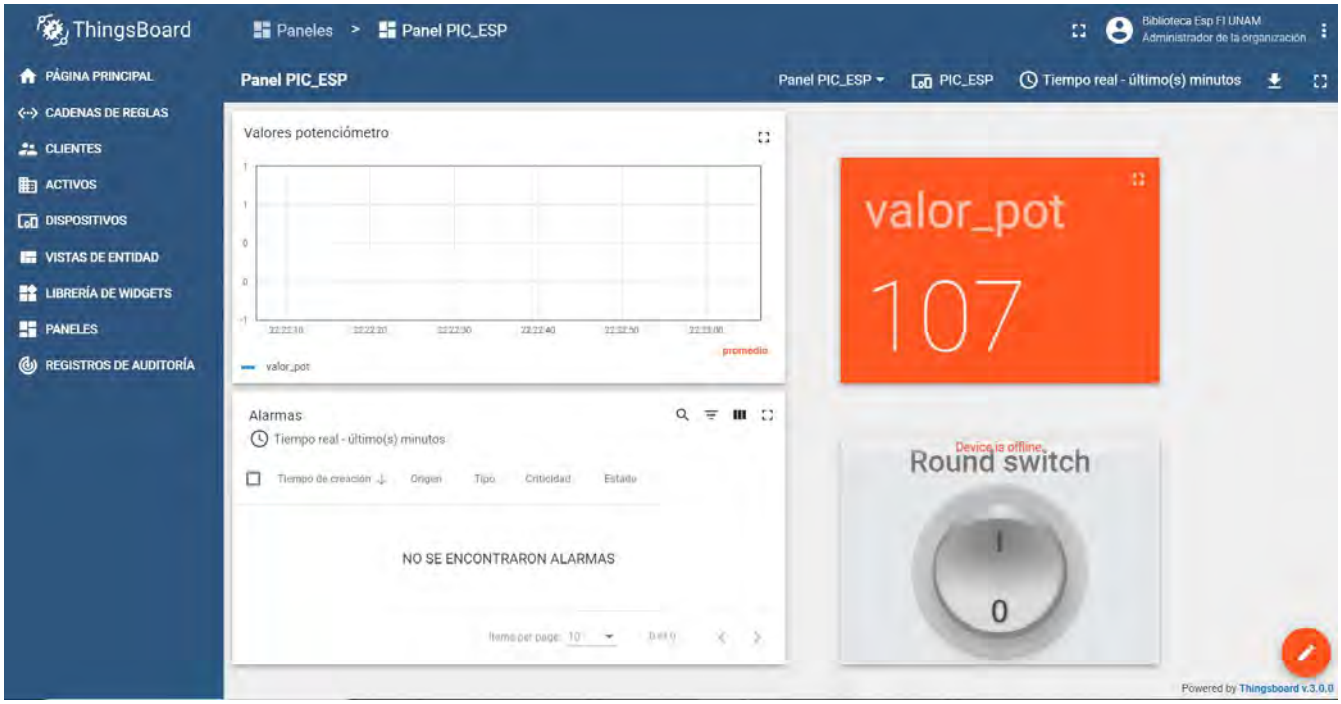


Figura P24-B.7 *Panel finalizado.*

Figura P24-B.8 *Script que establece el límite de las lecturas.*

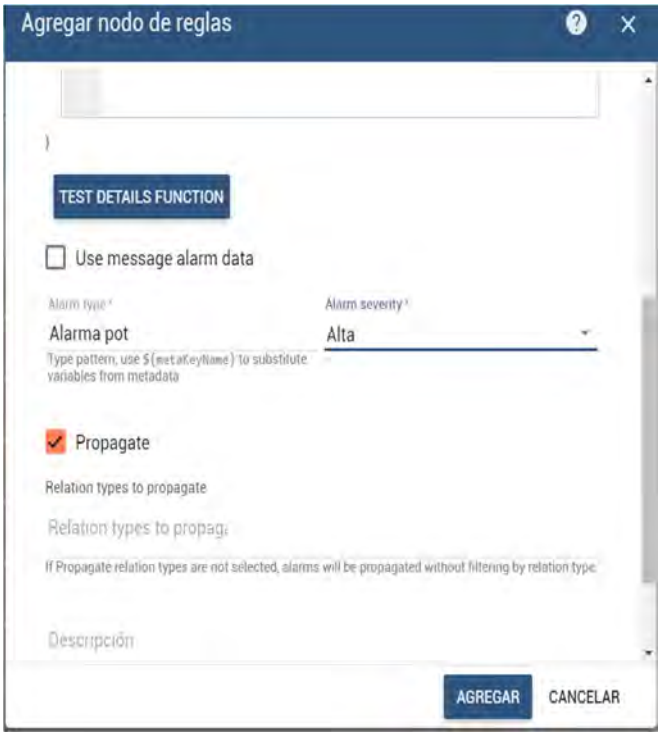


Figura P24-B.10 *Configuración de creación de alarma.*

Figura P24-B.9 *Mensaje para encender LED de la alarma.*

A la salida del último nodo agregado se bifurcan, uno que conduce a la ejecución de una RPC que encenderá el LED y el otro que envía un correo electrónico avisando que se activó la alarma (ver figura P24-B.12).

Para el flujo de la RPC que enciende el LED, se agrega un nodo *script* en el que se establece que el mensaje que se enviará al módulo WiFi será {"method":"Alarma","params":true} (ver figura P24-B.9).

Este nodo se conecta a la salida del nodo de crear alarma con la etiqueta *created*. Posteriormente se agrega el nodo *rpc call request*, que ejecutará la RPC, al cual sólo se le colocará un nombre ya que por defecto ya cuenta con un *timeout* de 60 segundos, y se conectará a la salida del *script* del mensaje RPC con la etiqueta *Success*.

Para el flujo que envía el correo electrónico, se agrega el nodo "*to mail*", en el cual se configura el contenido del correo. Se colocará la dirección de una cuenta de correo que pueda enviar correos mediante SMTP, se recomienda usar cuentas de proveedores que otorguen contraseña o *token* únicamente para enviar correos, ya que así no se tendrá que compartir las credenciales de toda la cuenta con *Thinsgboard*.

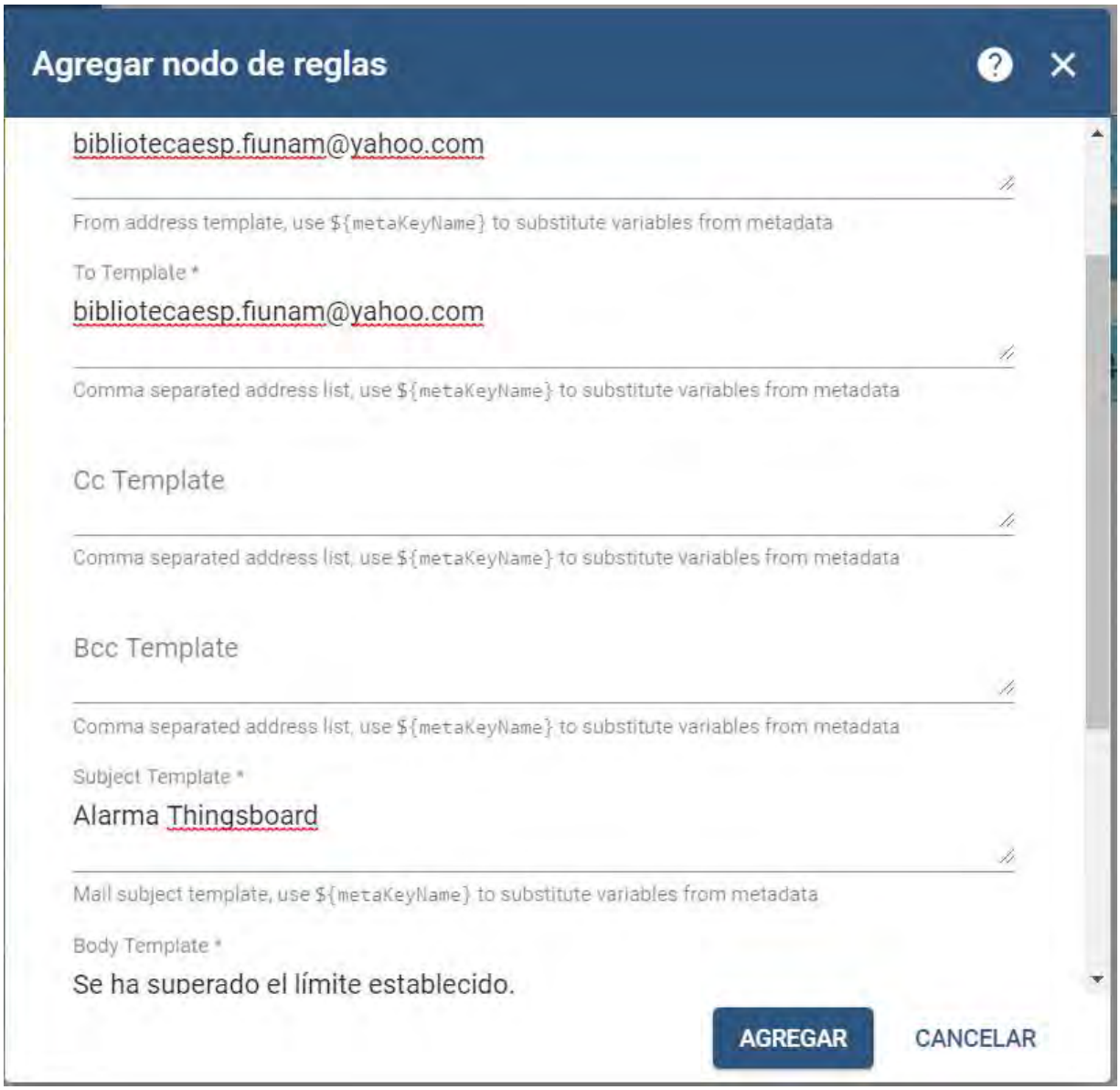


Figura P24-B.11 *Ejemplo de contenido de correo.*

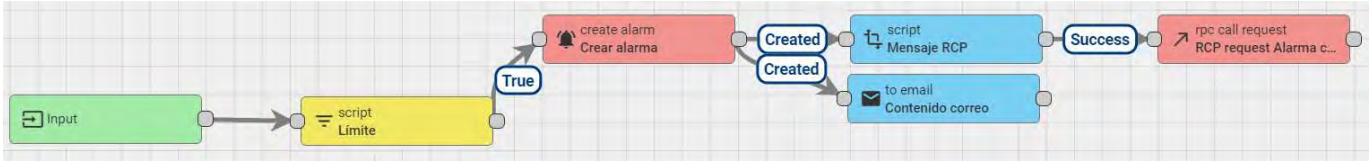


Figura P24-B.12 *Conectando contenido de correo con la salida de crear alarma.*

En este nodo también se coloca la dirección de correo destino, que puede ser la misma del emisor, se coloca asunto y cuerpo del correo (ver figura P24-B.12). Este nodo se conecta a la salida del nodo de crear alarma con la etiqueta *created (creado)* (ver figura P24-B.12). A la salida del nodo *to mail* se conecta un nodo *send mail (enviar correo)* con la etiqueta *success (éxito)*, en este nodo se especifica el nombre del servidor SMTP y el puerto que se utilizará (ver figura P24-B.13), se ingresa la dirección de correo y la contraseña para enviar el correo.

Regresando al *script* de límite, se crea un nuevo flujo con la etiqueta *false* que se conecta a un nodo *clear alarm*. En este nodo se coloca como tipo de alarma el mismo utilizado en la creación de alarma (*Alarma pot*), este nodo se encarga de limpiar (borrar) la alarma al momento en que las lecturas vuelvan a valores inferiores al límite establecido. Después se replica el flujo que ejecuta RPC para encender LED, pero esta vez el valor del parámetro será falso.

La regla de cadena quedaría como la que muestra en la figura P24-B.15. Se aplican todos los cambios y sólo faltaría agregar dicha cadena a la cadena raíz. Para esto,

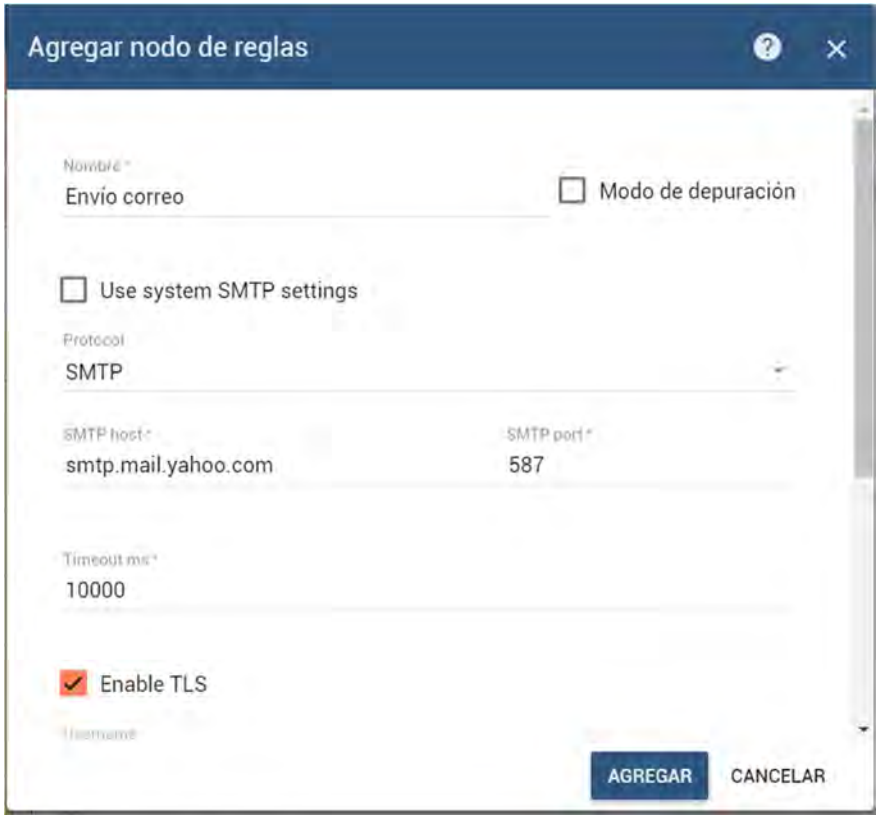


Figura P24-B.13 *Ejemplo de configuración de envío de correo mediante SMTP.*

Figura P24-B.15 *Resultado final de la cadena de reglas creada para la práctica.*

se regresa a las listas de cadenas de reglas, se localiza la cadena raíz, se abre y se agrega un nodo *rule chain,* se selecciona la cadena que se acaba de crear y se conecta *save Time s*eries (Guardar series de tiempo) de donde se publican los datos de telemetría proveniente de los dispositivos conectados (ver figura P24-B.15).

El módulo se comunicará con *Thingsboard* mediante el protocolo MQTT en su versión 3.1.1. por lo que el módulo se conectará al servidor demo.thingsboard.io en el puerto 1883 (TCP). El módulo iniciará sesión como el dispositivo creado en la plataforma, por ello, utiliza como nombre de usuario el *token* de acceso y una contraseña vacía (“\0”).

Para publicar información en la plataforma, el módulo lo realizará para el tema v1/devices/me/telemetry. Además, dicha información se enviará en formato JSON. Para esta práctica las lecturas se publican con la siguiente forma: {"valor_pot":"{valor de lectura}"}.

Para recibir información, el módulo se suscribirá al tema v1/devices/me/rpc/request/+. La razón de utilizar el comodín “+” es que para

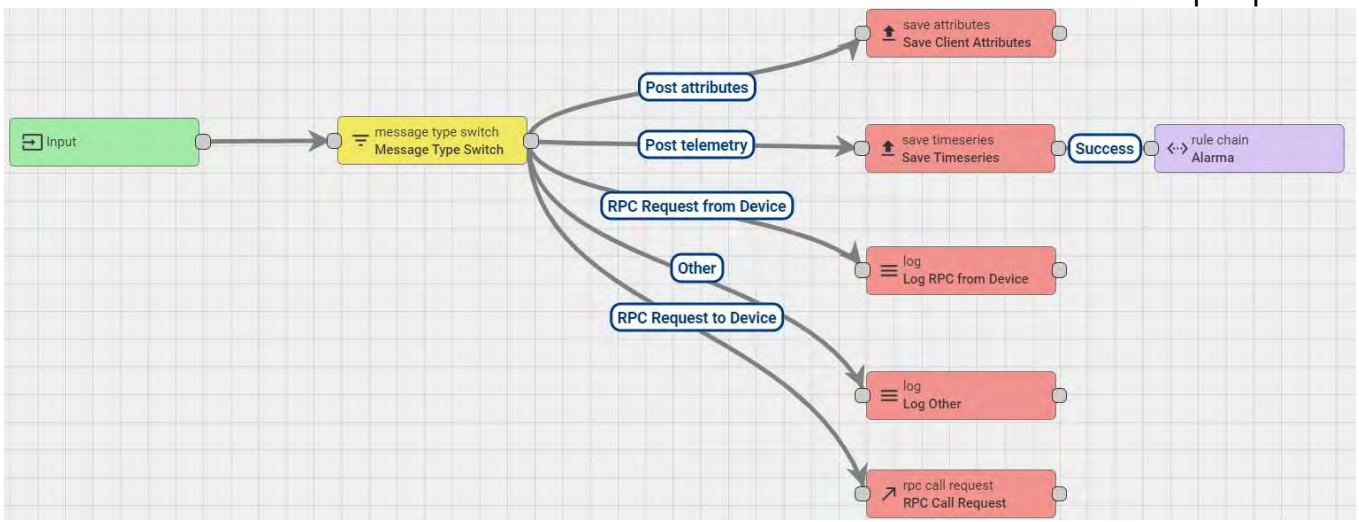


Figura P24-B.14 *Resultado de la cadena raíz.*

cada una de las RPC le corresponde un tema en concreto basado en su ID (v1/devices/me/rpc/request/id_rcp) y el módulo no necesita hacer distinción al ID, ya que lo que le interesa es el contenido del mensaje, al menos para esta práctica. Los mensajes que el código espera para esta práctica son:

- {"method":"LED","params":true} para encender el LED controlado por el interruptor
- {"method":"LED","params":false} para apagar el LED controlado por el interruptor
- {"method":"Alarma","params":true } para encender LED controlado por la alarma
- {"method":"Alarma","params":fals e} para apagar LED controlado por la alarma

Código

El código que deberá ejecutar el microcontrolador PIC para esta práctica es el mostrado en la figura P24-B.20y en la figura P24-B.21, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet
- *Token* de acceso para el dispositivo creado en Thingsboard

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P24-B.22.

Resultados

En el panel creado, se graficarán y mostrarán los valores de lectura que el microcontrolador PIC registre (ver figura P24-B.16). Al accionar el interruptor se encenderá o apagará uno de los LED.

Si se modifica el valor de la señal hasta un valor igual o superior a 200, entonces se activará la alarma (ver figura P24-B.17), se enviará un correo electrónico avisando que se ha superado el límite (ver figura P24-B.19) y se enviará un mensaje que encenderá el LED indicador. Al reducir los valores por debajo del límite se limpiará la alarma (ver figura P24-B.18) y se apagará el LED.

Durante la ejecución se podrá modificar el valor límite, con ello, el LED indicador se encenderá para distintos valores sin necesidad de modificar el programa del microcontrolador PIC.

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/2YUTeQN

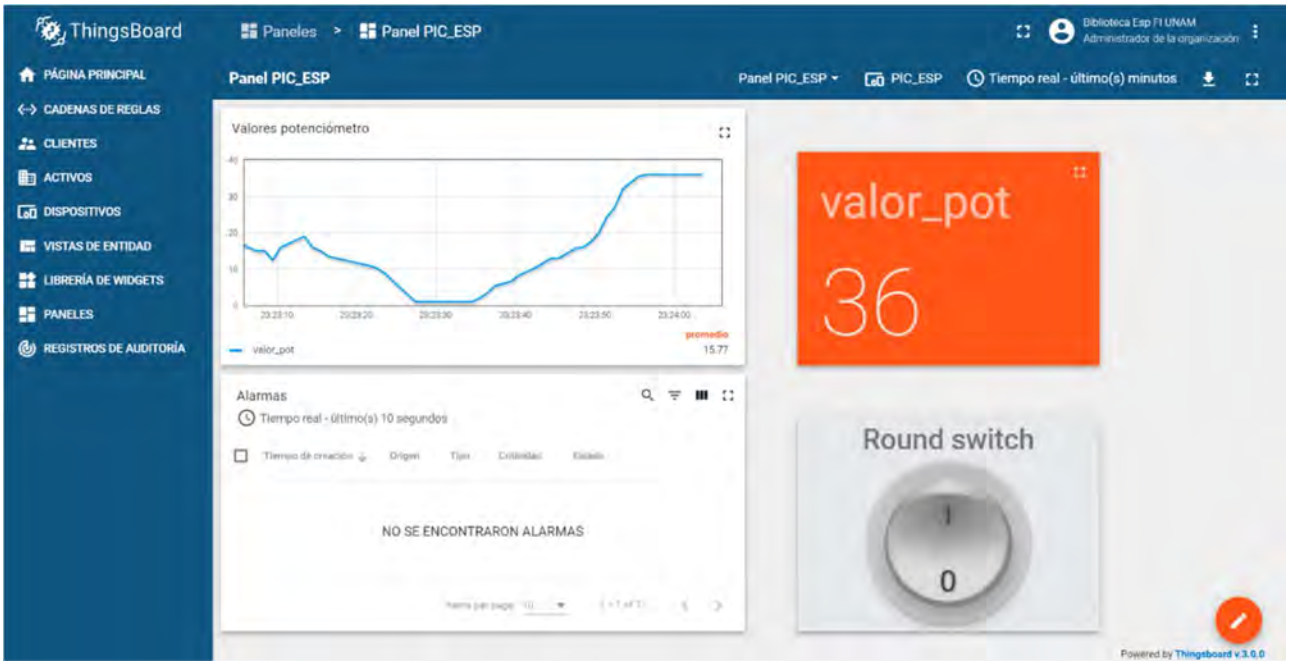


Figura P24-B.16 *Panel del PIC-ESP graficando en tiempo real las lecturas de una entrada analógica.*

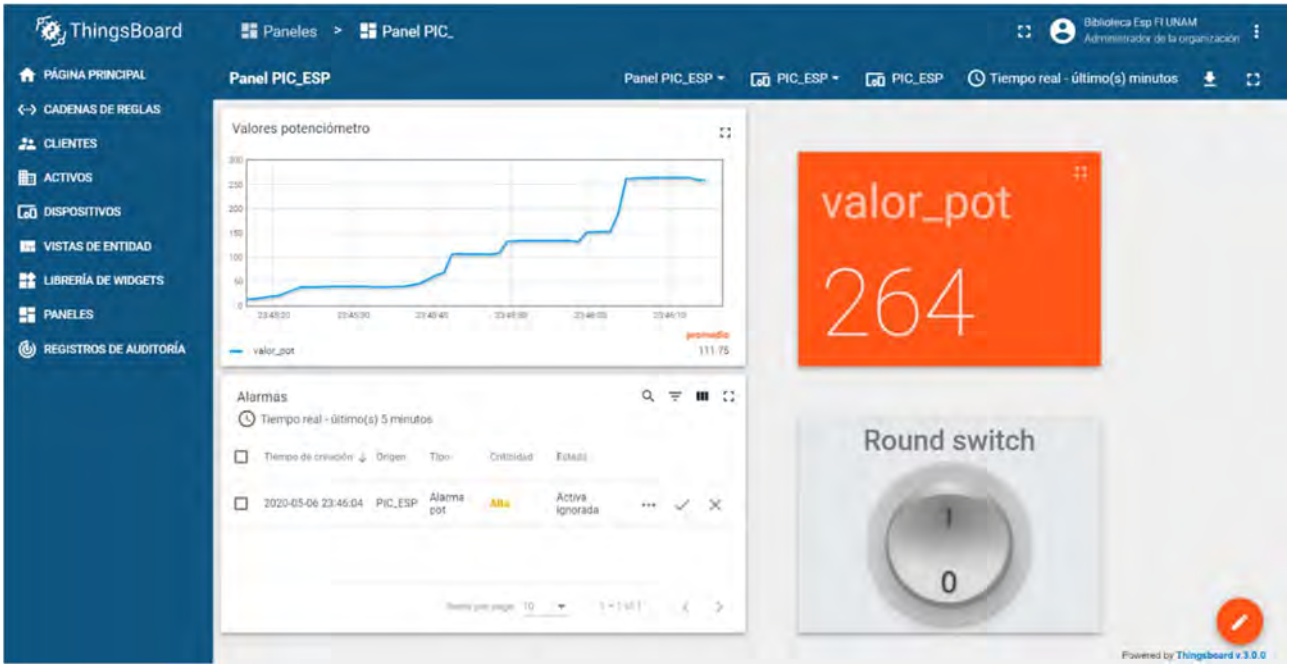


Figura P24-B.17 *Alerta activa cuando las lecturas sobrepasan el límite establecido.*

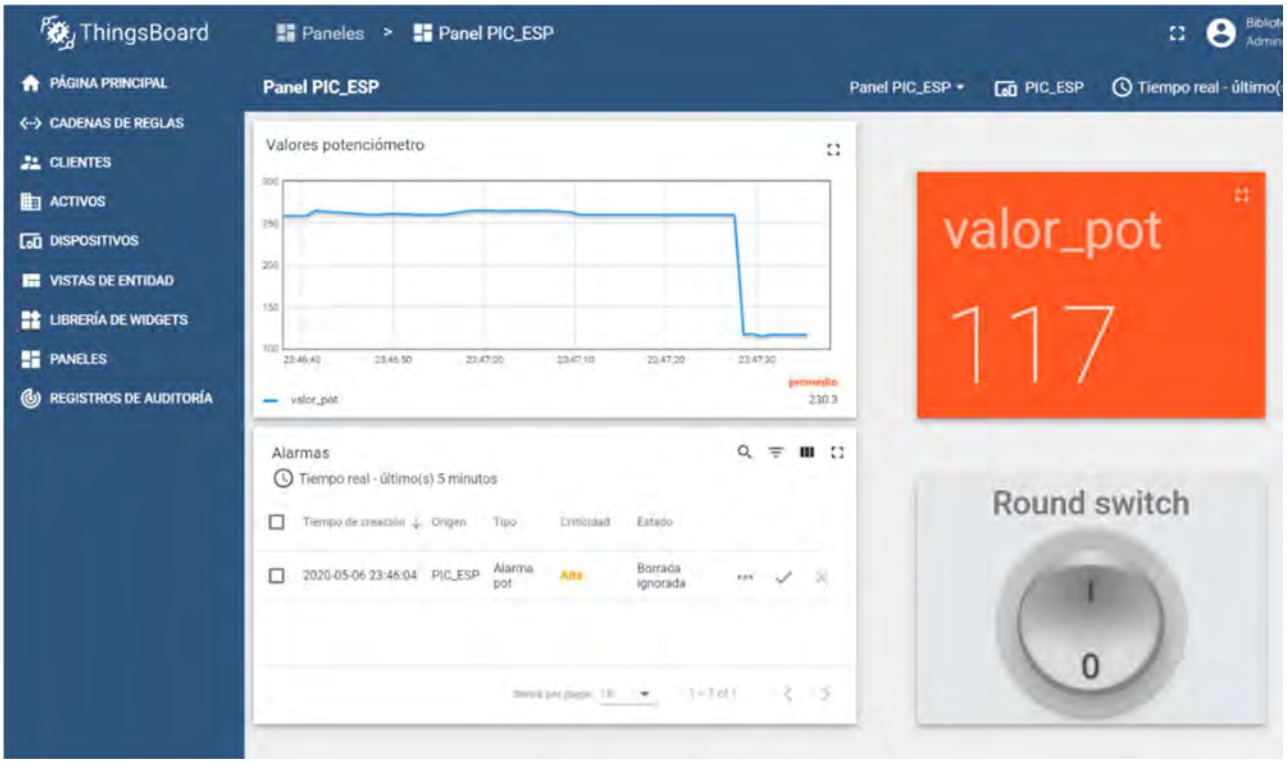


Figura P24-B.18 *Alerta borrada cuando las lecturas regresan a un valor inferior al límite establecido*

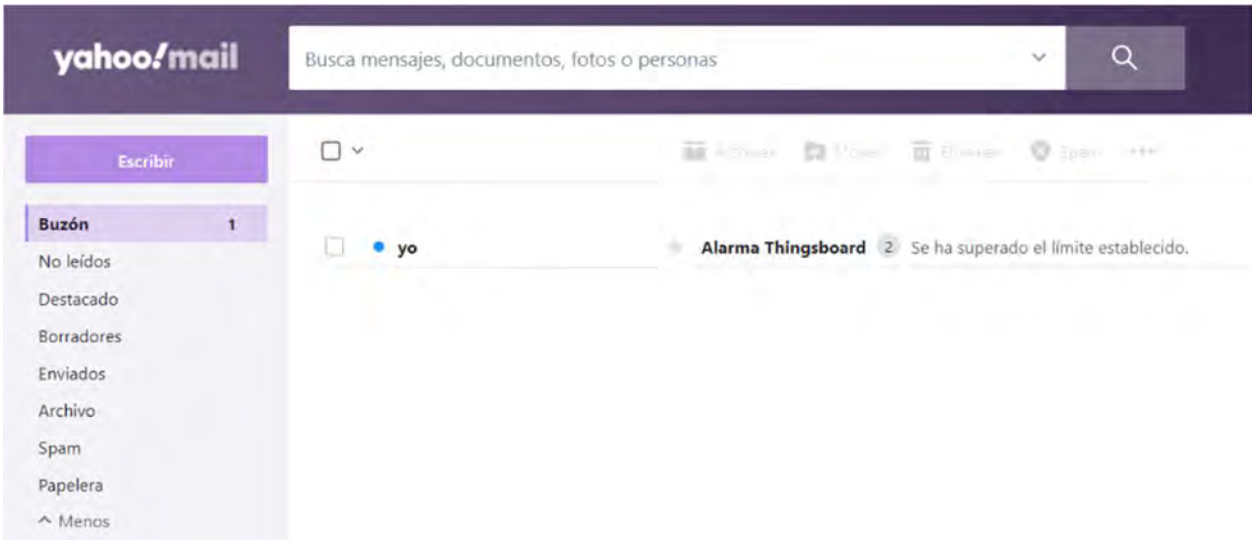


Figura P24-B.19 *Correo electrónico avisando que se ha superado el límite establecido.*

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define access_token "token de acceso"

#define subscribe_topic "v1/devices/me/rpc/request/+"
#define publish_topic "v1/devices/me/telemetry"
#define led_on_json "{\"method\":\"LED\",\"params\":true}"
#define led_off_json "{\"method\":\"LED\",\"params\":false}"
#define alarm_on_json "{\"method\":\"Alarma\",\"params\":true}"
#define alarm_off_json "{\"method\":\"Alarma\",\"params\":false}"

char json[]="{\"valor_pot\":\"0000\"}";
char buffer_mqtt[80]={0};
char buffer_led[40]={0};
char str_valor[5]={0};
long valor=0;

void main()
{
  setup_adc(adc_clock_div_64);
  delay_us(10);
  setup_adc_ports(sAN10);
  /*Establece al PIN B7 y PIN B6 como salidas, al resto del
  puerto B como entradas*/
  set_tris_b(0b00111111);
  //Establece en cero todo el puerto B
  output_b(0);
  //Inicialización del módulo Wi-Fi
  begin_esp();
  //Conexión al punto de acceso indicado
  begin_station("Estacion",SSID,password);
  /*Establece buffer de mensajes PUBLISH*/
  set_buffer_mqtt(buffer_mqtt,sizeof(buffer_mqtt));
  /*Se crea el cliente MQTT (se conecta al servidor de la plataforma)
  utilizando el puerto TCP y el canal 4*/
  create_mqtt("demo.thingsboard.io",1883,4);

/*El cliente se conecta como un dispositivo Thingsboard. Se establece
  el ID de cliente MQTT como 4001 y un keepalive de 1200s*/
  connect_mqtt("4001",access_token,"",1200);
  //Se subscribe al tema indicado con QoS0
  subscribe_mqtt(subscribe_topic,0);
  publishQoS0_mqtt(publish_topic,json,0);
  while(1)
  {
      //Obtención de valor
      set_adc_channel(10);
      //valor=(read_adc()>>1);
      //Valor leído es colocado en un string
      sprintf(str_valor,"%04lu",valor);
      //Atiende la recepción de mensajes PUBLISH
      refresh_mqtt();
      /*Extrae la carga útil del mensaje PUBLISH para cualquier
      tema (v1/devices/me/rpc/request/id)*/
      get_allpayload_mqtt(buffer_led,sizeof(buffer_led));
```

Figura P24-B.20 Primera parte del *código de la práctica 24-B.*

```
        //Enciende LED
        if(compare_string(led_on_json,buffer_led))
        {
          output_bit(PIN_B7,1);
        }
        //Apaga LED
        else if(compare_string(led_off_json,buffer_led))
        {
          output_bit(PIN_B7,0);
        }
        //Enciende LED de alarma
        if(compare_string(alarm_on_json,buffer_led))
        {
          output_bit(PIN_B6,1);
        }
        //Apaga LED de alarma
        else if(compare_string(alarm_off_json,buffer_led))
        {
          output_bit(PIN_B6,0);
        }
        //Coloca el valor obtenido en el string que se envía
        for(int i=0; i<5; i++)
        {
           if(str_valor[i]=='\0')
           {
             json[14+i]='"';
             json[15+i]='}';
             json[16+i]='\0';
             break;
           }
           else
            json[14+i]=str_valor[i];
        }
        //Publica con calidad 0
        publishQoS0_mqtt(publish_topic,json,0);
        delay_ms(200);
    }
}
```

Figura P24-B.21 Segunda parte del *código de la práctica 24-B.*

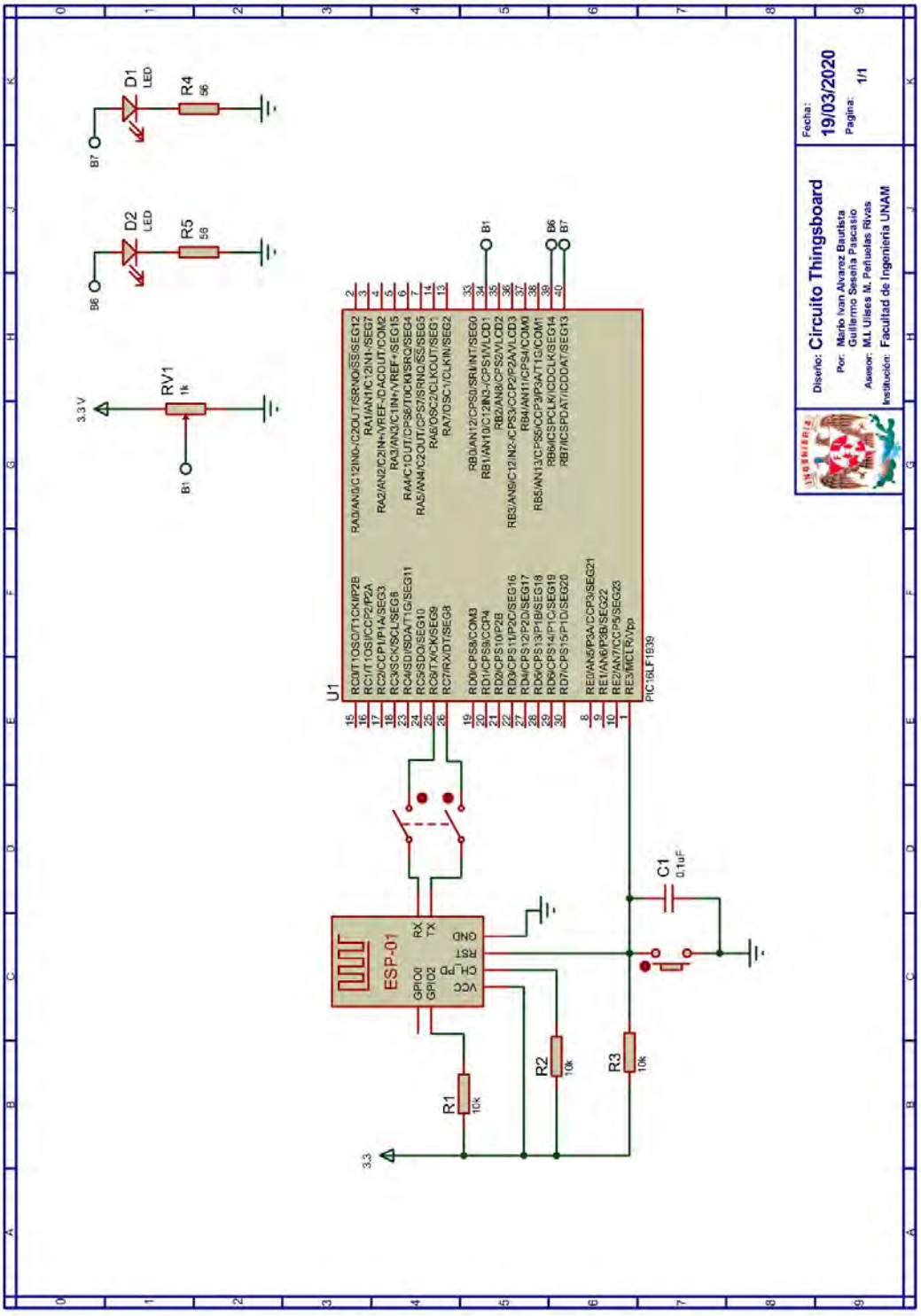


Figura P24-B.22 *Circuito de la práctica 24-B.*

Referencias

1. *ThingsBoard*. [Citado 2020 Marzo]; Disponible en:https://thingsboard.io/.

Práctica 24-C Plataformas IoT: adafruit

Introducción

Adafruit IO es una plataforma IoT que ofrece la popular empresa Adafruit Industries dedicada a la venta de *hardware* basado de código abierto. Esta plataforma ofrece dos formas de comunicación, una mediante una API basada en REST y otra con un bróker MQTT. Ambas formas permiten a los dispositivos enviar/recibir información. La diferencia entre usar una u otra, recae en el tipo de aplicación que se requiera o en las capacidades del dispositivo a conectar[1].

Adafruit IO está basada en *feeds,* cada uno de éstos representa un tipo de dato que intercambia con los dispositivos. Los *feeds* contienen información de los datos (*metadatos*) que fluyen, como lo son la fecha y hora de registro e inclusive coordenadas geográficas del dispositivo que los emite. Los *feeds* traducidos a MQTT corresponden a un tema, a los cuales los dispositivos pueden publicar o suscribirse. En la página https://io.adafruit.com/api/docs se encuentra más información de cómo utilizar MQTT y HTTP en esta plataforma.

Una característica relevante de esta plataforma es que permite la fácil creación de *dashboards* (tableros) que sirven para monitorizar y controlar los dispositivos conectados a la plataforma. Los *dashboard* se construyen a partir de *blocks* (bloques) que son *widgets*, los cuales el usuario agrega, asocia con un *feed* y especifica un par de parámetros para su funcionamiento. Además, esta plataforma integra servicios como IFTTT y Zapier para interactuar con otros servicios en línea. Para más información de las características de la plataforma re recomienda ingresar a la página https://learn.adafruit.com/welcome-to-adafruit-io.

Al momento de redactar esta práctica, la plataforma ofrece dos tipos de cuentas una completamente gratuita y otra de paga. La gratuita incluye: envío/ recepción de hasta 30 datos en un minuto, almacenamiento por 30 días, disponer de hasta 10 *feeds* y 5 *dashboard*. Mientras que, la cuenta de paga ofrece: envío/recepción de hasta 60 datos en un minuto, 60 días de almacenamiento, *feeds* y *dashboards* ilimitados.

El módulo ESP puede comunicarse con esta plataforma, utilizando el protocolo HTTP o MQTT. Pero al igual que con la plataforma *Thingspeak*, se recomienda no utilizar HTTP para consultar información, ya que resulta complicado extraer la información utilizando la biblioteca ESP.

Objetivo

Comunicar al módulo ESP a la plataforma Adafruit IO utilizando el protocolo MQTT.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_MQTT_v311

Materiales

- 1 computadora
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 capacitor de 0.1 µF
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 55 Ω
- 1 potenciómetro de 10 kΩ

Descripción

En esta práctica el módulo será establecido como un cliente MQTT usando la versión 3.1.1, el cual se comunicará con el bróker MQTT de la plataforma Adafruit IO. El módulo se suscribirá a un *feed* para recibir información que controle el estado de un LED y publicará valores leídos de una entrada analógica del microcontrolador PIC en otro *feed*. Aunque en esta práctica se utiliza MQTT, es posible emplear HTTP para enviar los valores leídos por el PIC, en el enlace al final de esta práctica se incluye un código para realizarlo.

El control del LED se realizará utilizando un interruptor desde un *dashboard* creado en la página de la plataforma, y en este mismo se visualizarán los valores leídos por el microcontrolador PIC mediante una gráfica.

El primer paso es crear una cuenta en Adafruit IO, por lo que se accede a la página https://io.adafruit.com/ y se selecciona algún tipo de cuenta, para esta práctica bastará con una cuenta gratuita.

Ya registrado, en la página se selecciona la opción "*Feed*" y después se selecciona "*view all*", se mostrarán todos los *feeds* disponibles tal y como se muestra en la figura P24-C.1. Se procede a crear los *feeds* requeridos en esta práctica, por lo que se oprime el botón "*Actions*", se

selecciona "*Create a new group*". Al grupo creado se le nombra "pic_esp" el cual contendrá los *feeds* de esta práctica. Ya creado el grupo, se selecciona el recuadro que está a un costado del nombre del grupo, dentro de este grupo, se presiona "*Actions*" y después "*Create a new feed*", se crea un *feed* llamado "led" y posteriormente se crea otro *feed* llamado "pot", quedando tal como se muestra en la figura P24-C.3.

Para que el módulo pueda acceder a la plataforma, el cliente MQTT creado en él deberá conectarse al servidor io.adafruit.com, utilizando ya sea el puerto TCP o el puerto SSL, en esta práctica se utiliza el puerto SSL que es que es el 8883. Para que el cliente MQTT inicie sesión tendrá que colocar las credenciales que la plataforma otorga a cada usuario, por lo que se debe acceder a la opción en Adafruit IO Key. En ésta se muestra el nombre de usuario y la *active key* (clave activa) que se utiliza como contraseña en el inicio de sesión, un ejemplo de estas credenciales se muestra en la figura P24-C.2.

El módulo usa un ID aleatorio colocado manualmente en el código y establece un *keepalive* de 300 segundos. El módulo se suscribirá al tema "{nombre de usuario}/feeds/pic-esp.led", y publicará

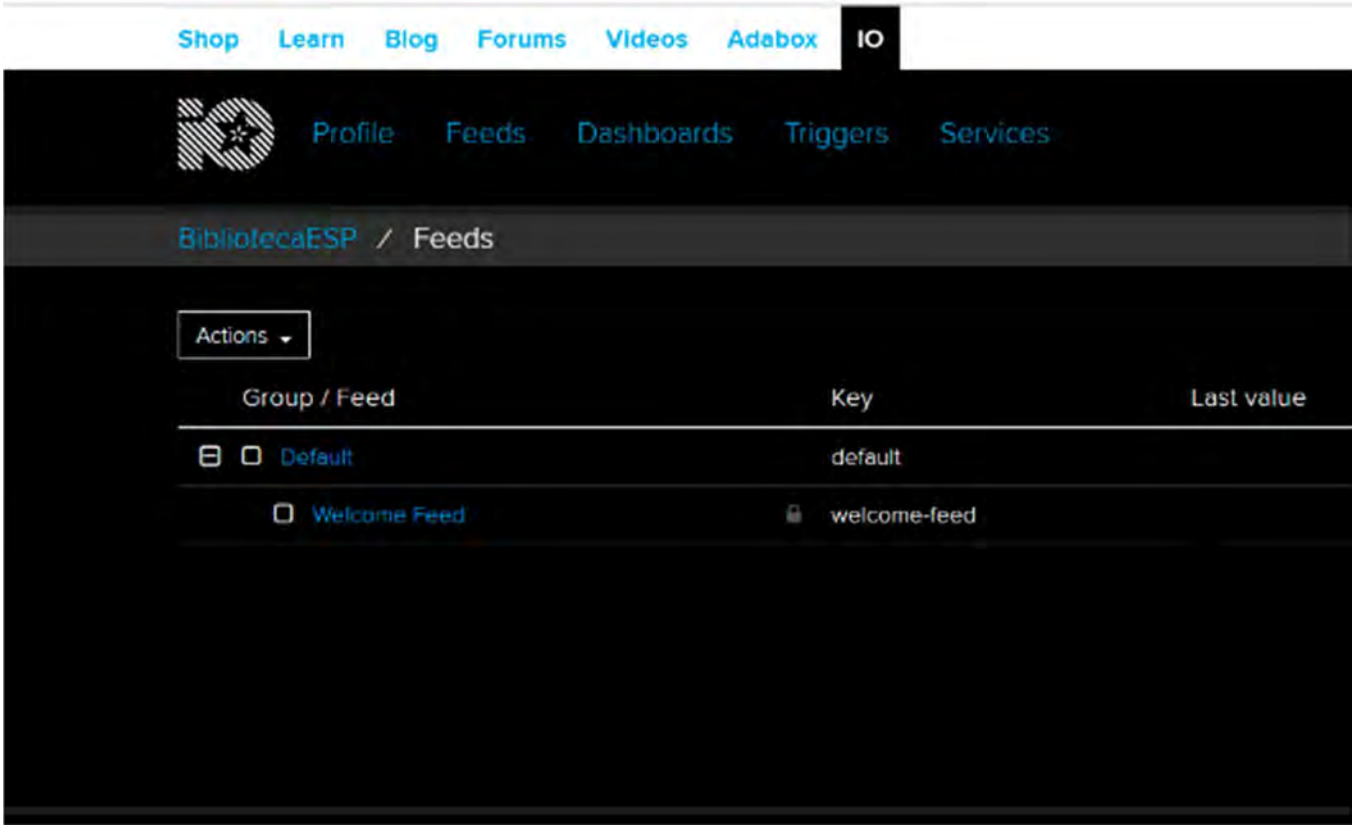


Figura P24-C.1 *Feeds disponibles al crear una cuenta en Adafruit IO.*

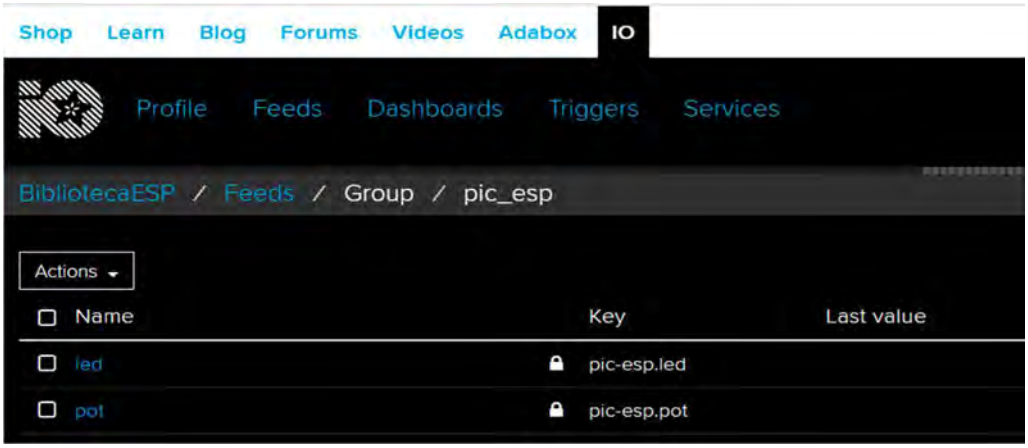


Figura P24-C.3 *Feeds utilizados en la práctica.*

los valores leídos en el tema "{nombre de usuario}/feeds/pic-esp.pot". Para esta plataforma se puede publicar/suscribir con QoS0 y QoS1, pero para esta práctica tanto la suscripción como las publicaciones se realiza con QoS0. También la información que se envía a la plataforma puede ser especificada en formato JSON, sin embargo, para la práctica se envía como un *string.*

Para crear el *dashboard*, en la página de la plataforma se selecciona "*Dashboard*" y a su vez se selecciona "*view all*", se mostrará todos los *dashboard* disponibles. Se crea uno nuevo, seleccionando "*Action*" y después "*Create a new databoard*".

El *dashboard* creado contendrá un interruptor que encienda y apague el LED junto con un indicador de estado, mientras que, para los valores leídos por

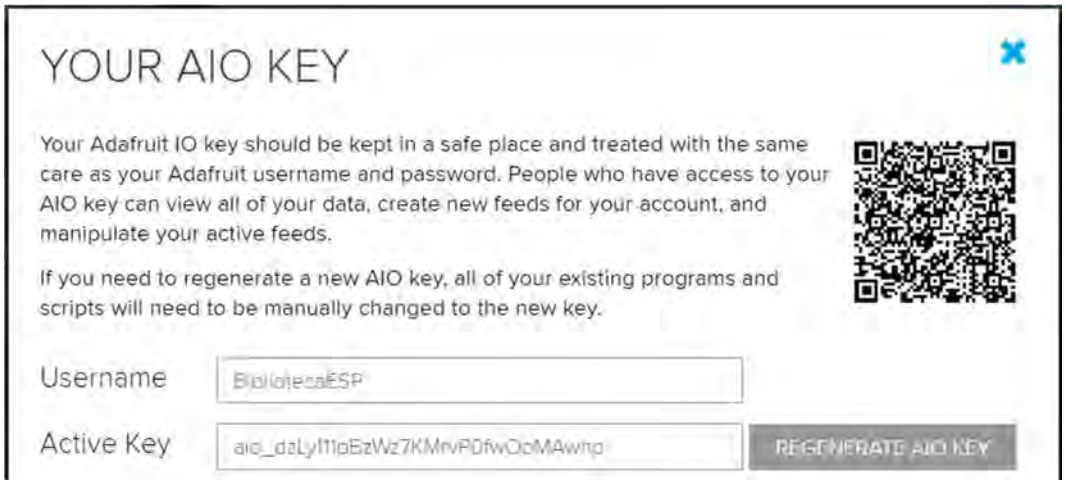


Figura P24-C.2 *Ejemplo de un AIO KEY.*

el PIC, se agregará una gráfica y un indicador. Para ello, se ingresa al *dashboard*, se oprime "*Create a new block*", se selecciona un *Toogle*, posteriormente se solicita el *feed* asociado a este *block*, por lo que se selecciona el *feed* llamado "led" (ver figura P24-C.4), en el siguiente paso se coloca el nombre del *block* y en el texto para encendido y apagado se coloca "*on*" y "*off*" respectivamente. Finalmente se termina de crear (ver figura P24-C.6). Posteriormente se agrega un *Indicator* (indicador) para el mismo *feed*, se configura los colores para cada estado y como condición se coloca la que se muestra en la figura P24-C.5.

Para la gráfica e indicador de los valores leídos, se agrega un *Line chart* y un *gauge* respectivamente, ambos se asocian al *feed* llamado "pot". Finalmente se ajusta el tamaño y ubicación de los *blocks* creados obteniendo algo semejante a lo que se muestra en la figura P24-C.7y se oprime "*Save*".

Código

El código que deberá ejecutar el microcontrolador PIC en esta práctica es el mostrado en la figura P24-C.10, en el cual se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet
- Nombre de usuario de la cuenta de Adafruit IO
- Adafruit IO Key
- Nombre del tema de control de estado del LED

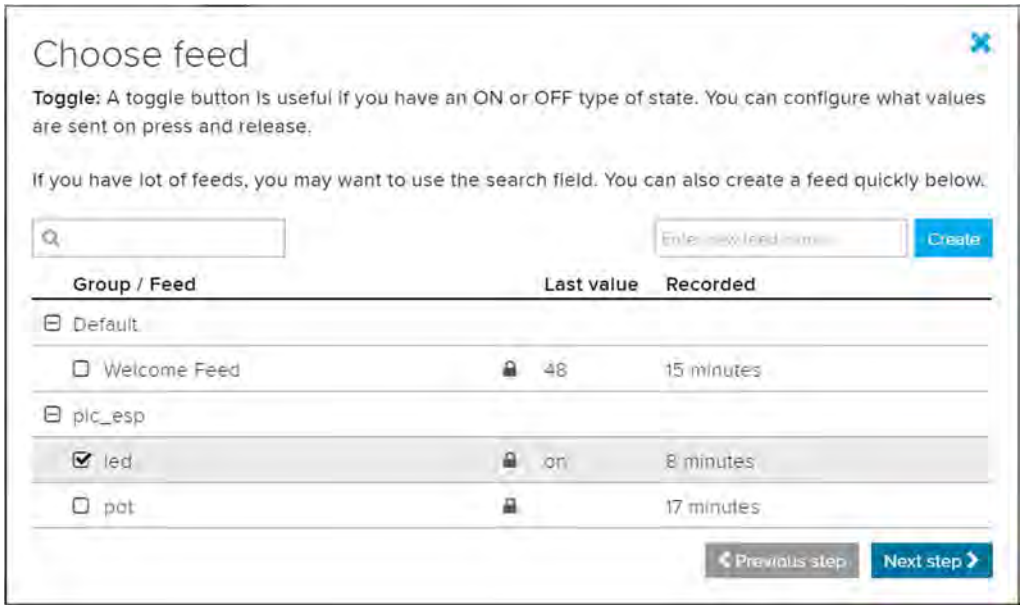


Figura P24-C.4 *Selección del feed del Toogle.*

- Nombre del tema de publicación de valores leídos

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P24-C.11.

Resultados

Se accede al *dashboard* creado en esta práctica, en cuanto el módulo se conecte a la plataforma se podrá observar cómo se van graficando los valores obtenidos de la entrada analógica del PIC (ver figura P24-C.8), para comprobar el correcto funcionamiento se debe variar la señal que está leyendo el PIC, por lo que se debe modificar la posición del eje del potenciómetro. Para encender o apagar el LED se debe interactuar con el *toogle,*

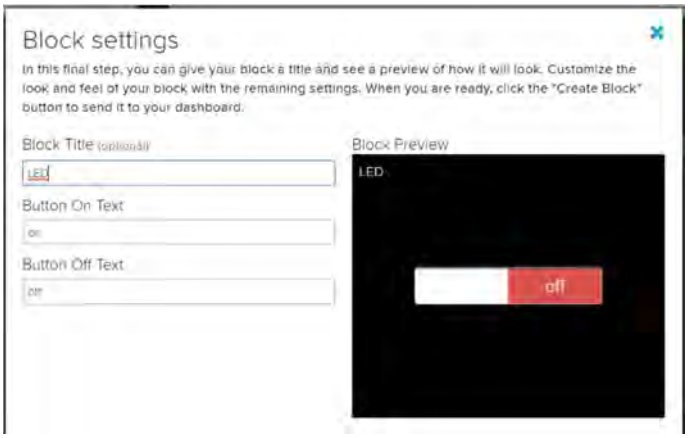


Figura P24-C.6 *Configuración del Toogle.*

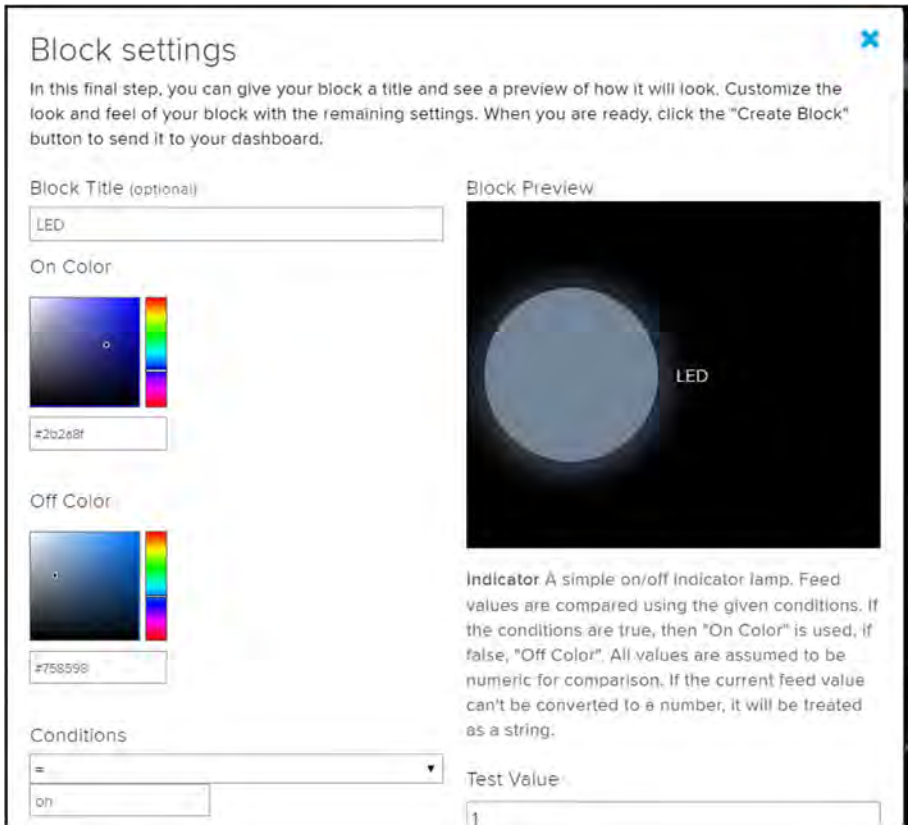


Figura P24-C.5 *Configuración del Indicator.*

al encender LED el indicador del LED también se encenderá (ver figura P24-C.9).

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/37M1Wo8

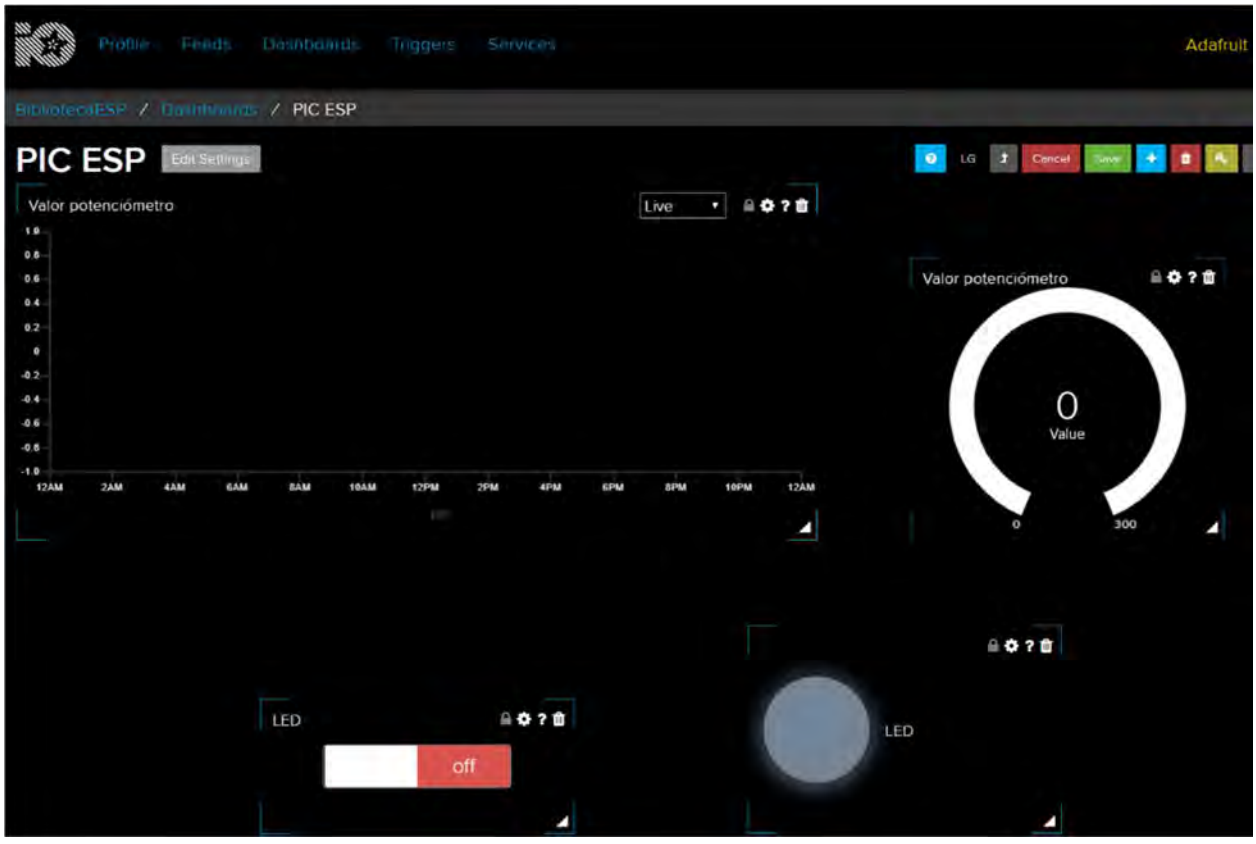


Figura P24-C.7 *Dashboard usado en la práctica.*

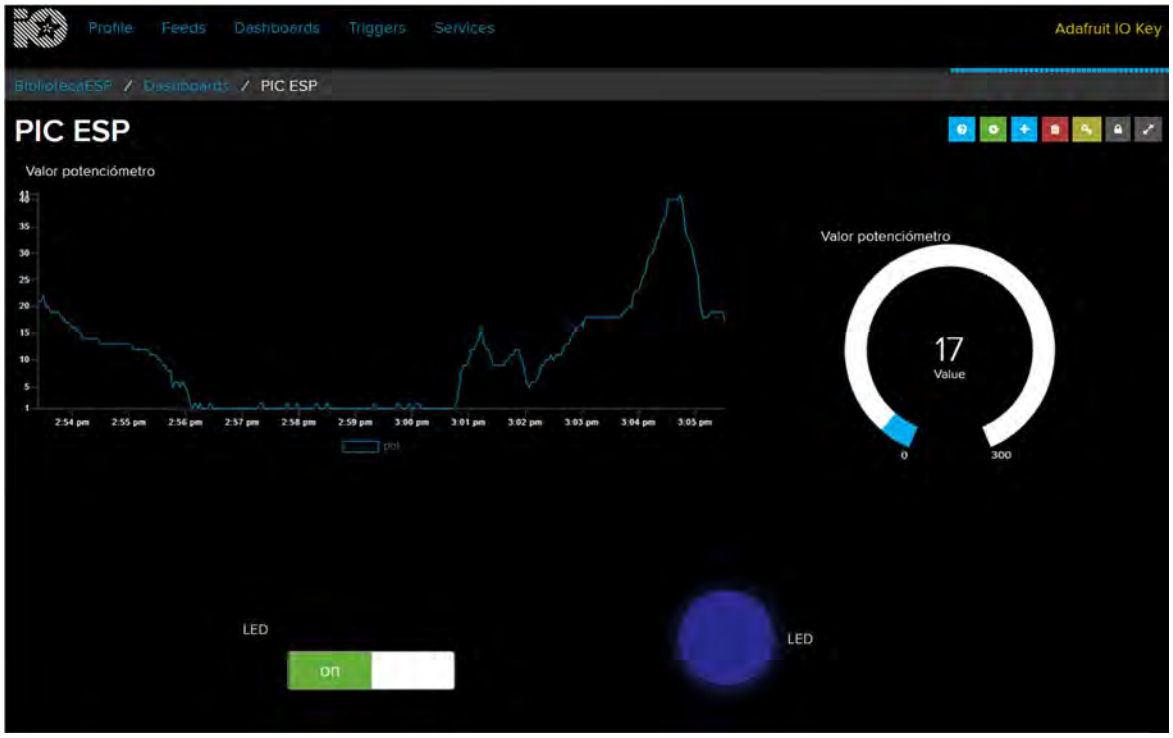


Figura P24-C.9 *Dashboard al encender LED.*

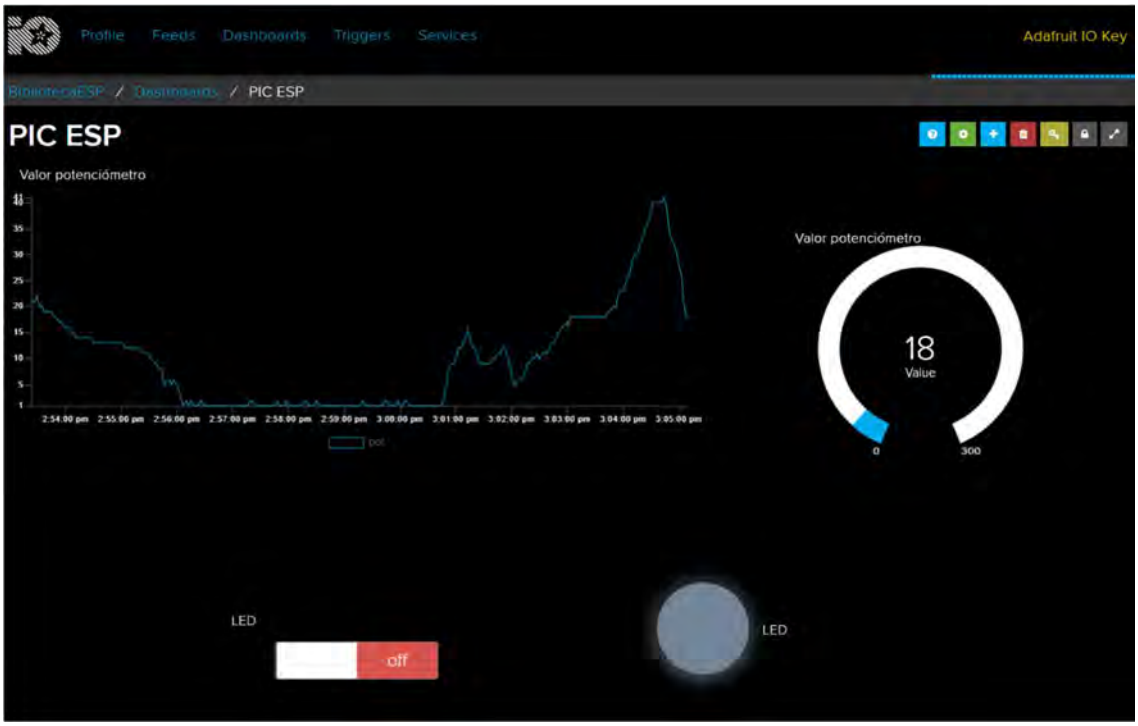


Figura P24-C.8 *Dashboard en funcionamiento.*

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

#define SSID "nombre del AP"
#define password "contraseña del AP"
#define io_user "nombre de usuario"
#define io_key "clave"
#define subscribe_topic "{nombre de usuario}/feeds/pic-esp.led"
#define publish_topic "{nombre de usuario}/feeds/pic-esp.pot"

char buffer_mqtt[40]={0};
char led_buffer[4]={0};
char str_valor[]="0000";
long valor=0;
void main()
{
    setup_adc(adc_clock_div_64);
    delay_us(10);
    setup_adc_ports(sAN10);
    /*Establece al PIN B7 como salida y al resto del puerto B
    como entradas*/
    set_tris_b(0b01111111);
    //Establece en cero todo el puerto B
    output_b(0);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Establece buffer de mensajes PUBLISH*/
    set_buffer_mqtt(buffer_mqtt,sizeof(buffer_mqtt));
    /*Se crea el cliente MQTT (se conecta al servidor de la plataforma)
    utilizando el puerto SSL y el canal 3*/
    create_mqtt("io.adafruit.com",8883,3,1);
    /*El cliente se conecta a una cuenta de Adafruit.Se establece
    el ID de cliente MQTT como 1001 y un keepalive de 300s*/
    connect_mqtt("1001",io_user,io_key,300);
    //Se subscribe al tema indicado con QoS0
    subscribe_mqtt(subscribe_topic,0);
    while(1)
    {
        //Obtención de valor
        set_adc_channel(10);
        valor=(read_adc()>>1);
        //Valor leído es colocado en el string que se envía
        sprintf(str_valor,"%lu",valor);
        //Atiende la recepción de mensajes PUBLISH
        refresh_mqtt();
        /*Extrae la carga útil del mensaje PUBLISH para el tema
        que controla el estado del LED*/
        get_payload_mqtt(subscribe_topic,led_buffer,sizeof(led_buffer));
        //Enciende LED
        if(led_buffer[0]=='o' && led_buffer[1]=='n')
            output_bit(PIN_B7,1);
        //Apaga LED
        else if(led_buffer[0]=='o' && led_buffer[1]=='f')
            output_bit(PIN_B7,0);
        /*Se publica el valor leído para el tema seleccionado*/
        publishQoS0_mqtt(publish_topic,str_valor);
        delay_ms(2600);
    }
}
```

Figura P24-C.10 *Código de la práctica 24-C.*

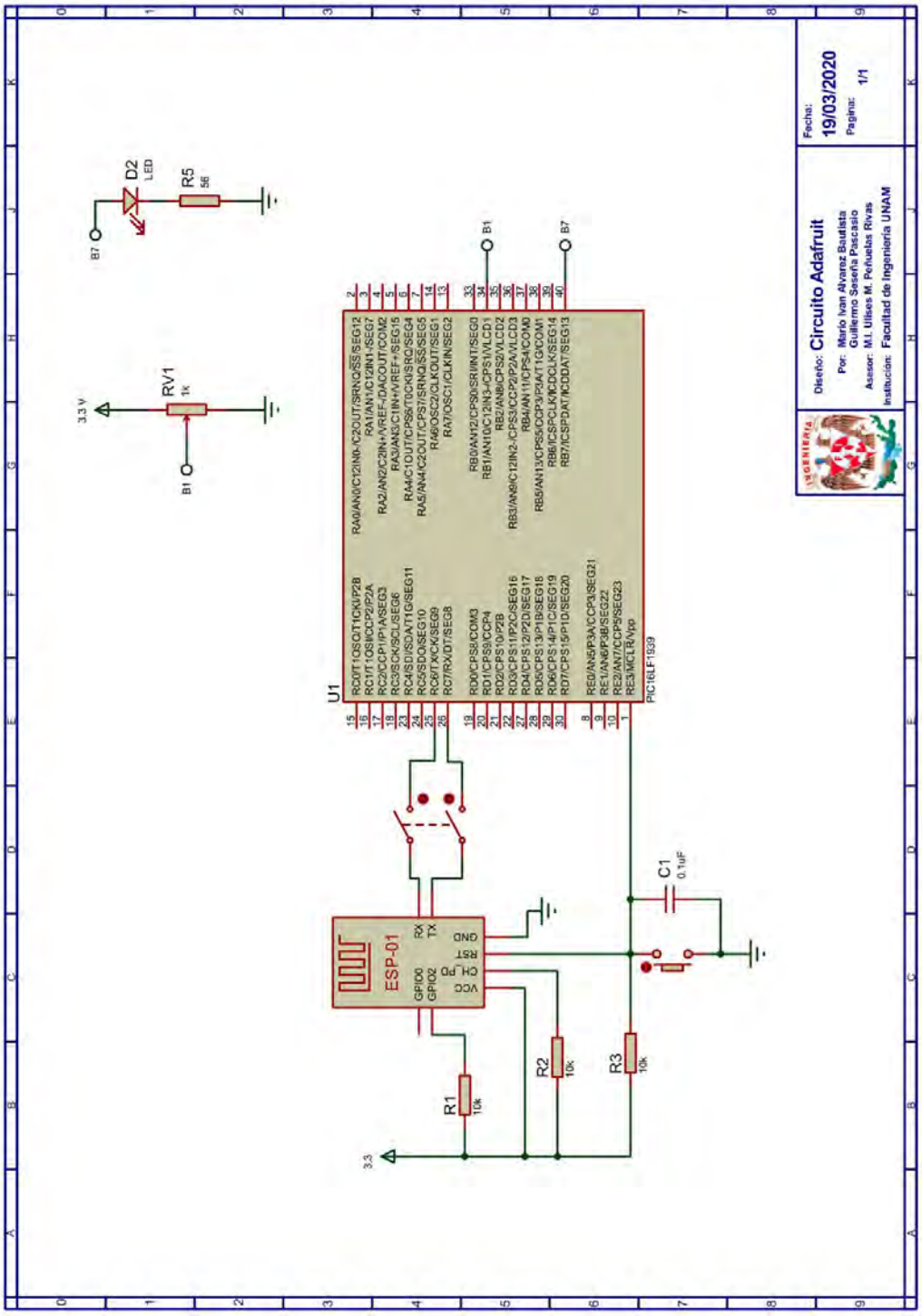


Figura P24-C.11 *Circuito de la práctica 25-C.*

Referencias

1. *Adafruit*. Disponible en:https://io.adafruit.com/.

Práctica 25-A IFTTT: asistente de Google

Introducción

IFTTT es una base de servicios web que ofrece automatización de tareas a través de la vinculación de distintos servicios *online* y redes sociales. Este sirve como intermediario de importantes servicios como Facebook, Twitter, Uber, Gmail, Asistente de Google, Amazon Alexa y muchos más, permitiendo crear un sin fin de aplicaciones [1].

Algunas de las aplicaciones más populares creadas con IFTTT son: sincronizar automáticamente la foto de perfil de distintas redes sociales, crear alarmas en dispositivos Android o iOS en función del clima e incluso utilizar los asistentes de Google y Amazon para controlar dispositivos domóticos.

IFTTT son las siglas en ingles de *IF This Then That* que se puede traducir como “si esto ocurre, entonces haz esto”, bajo este esquema se basan las aplicaciones creadas en IFTTT, las cuales se llaman *applets* o también conocidas como recetas.

Una *applet* está constituida por dos componentes un activador (*trigger*) y una acción (*action*), el activador espera a que ocurra el evento establecido para desencadenar la acción programada. Las *applets* puede ser creadas desde dispositivos móviles a través de la aplicación de IFTTT o también desde algún un navegador web.

Los módulos ESP pueden ser partícipes de aplicaciones IFTTT, fungiendo principalmente como clientes/servidores HTTP o como clientes MQTT.

Para más información sobre IFTTT consultar:

https://help.ifttt.com/hc/en-us/articles/115010325748-What-is-IFTTT -

Objetivo

Utilizar el servicio IFTTT para controlar el estado de un LED mediante comandos de voz del Asistente de Google.

Firmware de comandos AT

Se puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_MQTT_V311

Materiales

- 1 computadora
- 1 dispositivo móvil Android/iOS
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 1 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 LED de 2.2V y 20 *mA*
- 1 resistor de 56 Ω

Descripción

Existen varias formas de controlar al PIC-módulo desde IFTTT, una de ellas es estableciendo al módulo como un servidor web, sin embargo, para que IFTTT pueda acceder a dicho servidor es necesario efectuar un mapeo de puertos. Además, esto implica que cuando la dirección IP pública de la subred a la que esté conectada el módulo cambie, la *applet* creada para este propósito debe ser modificada.

En vez de utilizar al módulo como un servidor, en esta práctica se utilizará como un cliente MQTT, ya que la creación de la *applet* es independiente de la conexión del cliente MQTT, además por las características del protocolo, el módulo obtendrá información para controlar el LED sin que éste pregunte por nueva información.

Para esta práctica se crean dos *applets*, una para encender el LED y otra para apagarlo, por lo que es necesario contar con una cuenta de IFTTT, una de Google y una de Adafruit IO. Ambas *applets* utilizan como activador a un comando de voz del Asistente de Google, mientras que, como acción ambas envían información a la plataforma Adafruit, la cual son mensajes para encender o apagar el LED. El módulo accede a esta información como un cliente MQTT de la plataforma Adafruit IO, es importante señalar que esta práctica está basada en el control del LED de la práctica 24-A, por ello se utiliza el mismo *feed* y parte del código de mencionada práctica.

La creación de las *applets* para esta práctica es ilustrada desde un navegador web. La primer *applet* que se creará es para encender el LED. Se accede a la

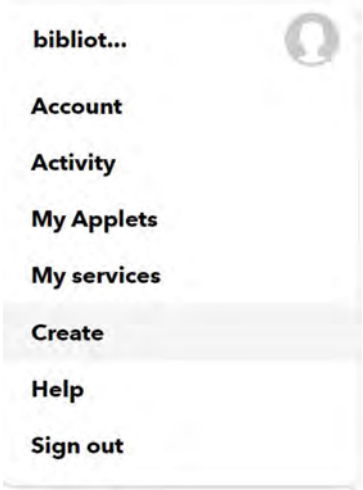


Figura P25-A.1 *Selección de opción Create.*

página de IFTTT, se inicia sesión y se selecciona la opción de "*create"* (ver figura P25-A.1), al realizarlo esto se muestra "If+This Then That" (ver figura P25-A.2), se oprime "+ *This*" para agregar el servicio que funciona como activador, en este caso se selecciona "*Google Assitant"*, al agregarlo, Google solicita ingresar a una cuenta y conceder los permisos para utilizar dicho servicio.

Figura P25-A.2 *Página para crear una applet.*

Posteriormente, se selecciona el tipo de activador, el cual corresponde a "*Say a simple phrase"*. En la configuración del activador se coloca que la frase de activación es "Encender LED", se selecciona el idioma español y como respuesta se indica "Encendiendo LED" (ver figura P25-A.3).

Después de configurar el activador, se selecciona el servicio de Adafruit para la acción, por lo que se tiene que ingresar una cuenta de dicha plataforma. El tipo de acción que se selecciona es "*Send data to Adafruit IO"*. En la creación de la acción se coloca como *feed name* el nombre del tema "pic_esp-led" y como dato a guardar es establece "*on"* (ver

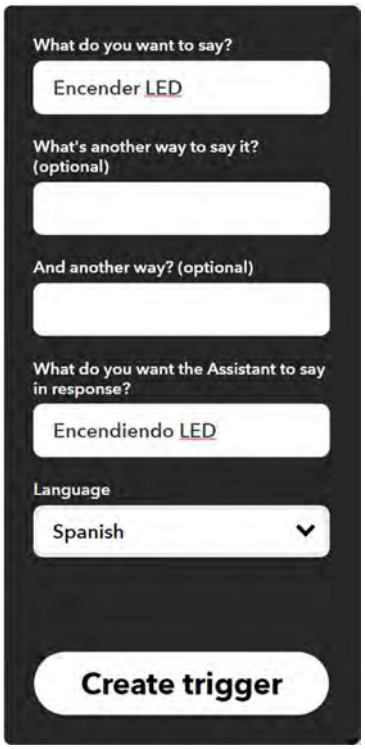


Figura P25-A.3 *Creación del activador del Asiste de Google.*

Figura P25-A.4 *Creación de la acción de Adafruit.*

figura P25-A.4) de este modo la *applet* funciona como publicador que sería lo equivalente al control del LED de la práctica 24-A.

La segunda *applet* se creará de la misma forma, solo que se cambia la frase del activador y la respuesta del asistente por "Apagar LED" y "Apagando LED" respectivamente, además se cambia el dato que se envía a Adafruit por "*off*".

Código

El código que se deberá ejecutar el microcontrolador PIC es el mostrado en la figura P25-A.6, en el que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet
- Nombre de usuario y clave de acceso de Adafruit IO
- Nombre del tema para controlar estado del LED (nombre_de_usuario/feeds/pic-esp.led)

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P25-A.7.

Resultados

Desde algún dispositivo móvil que tenga vinculada la cuenta de Google usada en la creación de las *applets,* se accederá a la aplicación del Asistente de Google. Primero se emitirá el comando voz "Encender LED", la aplicación responderá "Encendiendo LED" y eventualmente se encenderá el LED. Posteriormente se emitirá la frase "Apagar LED", responderá "Apagando LED" y se ejecutará dicha acción. En la figura P25-A.5 se muestra una interacción de los comandos utilizados con la aplicación de Asistente de Google.

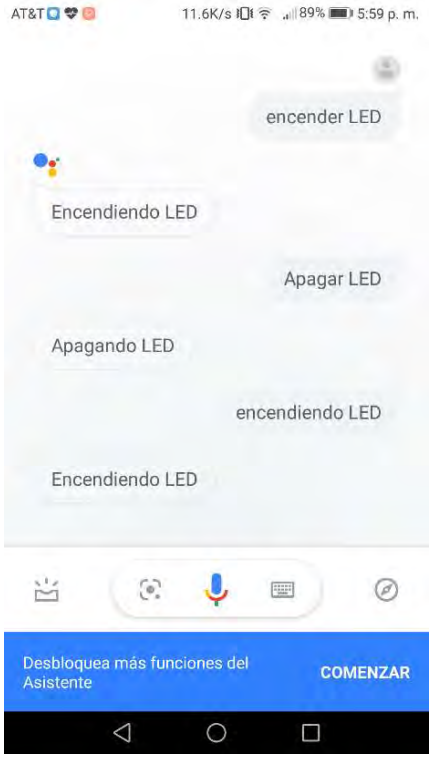


Figura P25-A.5 *Ejecución de comandos de voz en la aplicación del Asistente de Google.*

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3etlRe4

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define io_user "nombre de usuario"
#define io_key "clave"
#define topic "nombre_de_usuario/feeds/pic-esp.led"

char buffer_mqtt[40]={0};
char get_message[4]={0};
char mensaje[]="0000";
long valor=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    /*Establece buffer de mensajes PUBLISH*/
    set_buffer_mqtt(buffer_mqtt,sizeof(buffer_mqtt));
    //Se crea el cliente MQTT (se conecta al servidor de la plataforma)
    create_mqtt("io.adafruit.com",1883,3);
    /*El cliente se conecta a una cuenta de Adafruit.Se establece
    el ID de cliente MQTT como 1001 y un keepalive de 300s*/
    connect_mqtt("1001",io_user,io_key,300);
    //Se subscribe al tema indicado con QoS0
    subscribe_mqtt(topic,0);
    while(1)
    {
        //Atiende la recepción de mensajes PUBLISH
        refresh_mqtt();
        /*Extrae la carga útil del mensaje PUBLISH para el tema
        que controla el estado del LED*/
        get_payload_mqtt(topic,get_message,sizeof(get_message));
        //Enciende LED
        if(get_message[0]=='o' && get_message[1]=='n')
            output_bit(PIN_B7,1);
        //Apaga LED
        else if(get_message[0]=='o' && get_message[1]=='f')
            output_bit(PIN_B7,0);

    }
}
```

Figura P25-A.6 *Código de la práctica 25-A.*

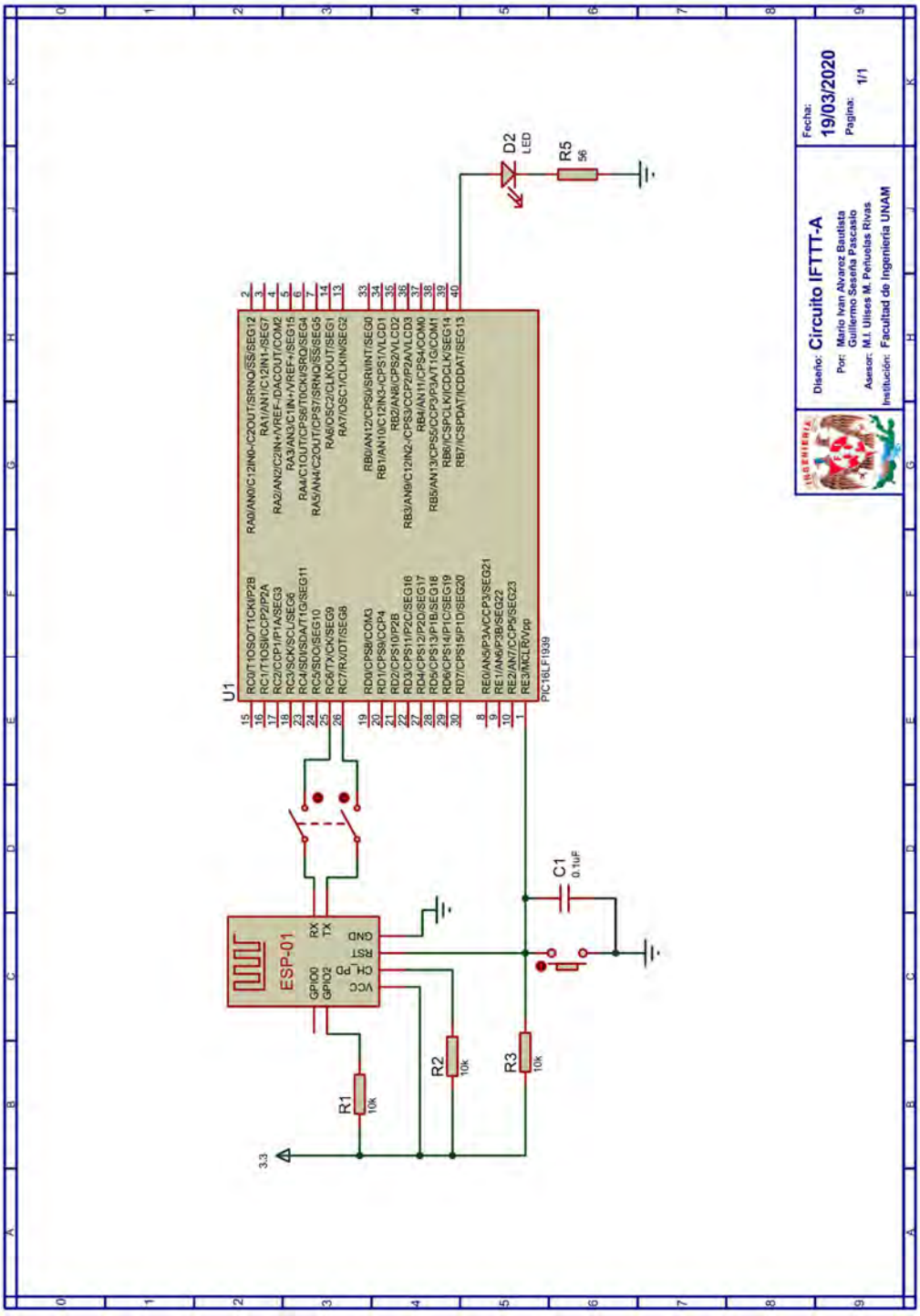


Figura P25-A.7 *Circuito de la práctica 25-A.*

Referencia

1. *IFTTT*. Disponible en:https://ifttt.com/.

Práctica 25-B IFTTT: publicar un Tuit

Introducción

IFTTT es una base de servicios web que ofrece automatización de tareas a través de la vinculación de distintos servicios *online* y redes sociales. Este sirve como intermediario de importantes servicios como Facebook, Twitter, Uber, Gmail, Asistente de Google, Amazon Alexa y muchos más, permitiendo crear un sin fin de aplicaciones [1].

Algunas de las aplicaciones más populares creadas con IFTTT son: sincronizar automáticamente la foto de perfil de distintas redes sociales, crear alarmas en dispositivos Android o iOS en función del clima e incluso utilizar los asistentes de Google y Amazon para controlar dispositivos domóticos.

IFTTT son las siglas en ingles de *IF This Then That* que se puede traducir como “si esto ocurre, entonces haz esto”, bajo este esquema se basan las aplicaciones creadas en IFTTT, las cuales se llaman *applets* o también conocidas como recetas. Una *applet* está constituida por dos componentes un activador (*trigger*) y una acción (*action*), el activador espera a que ocurra el evento establecido para desencadenar la acción programada.

Los módulos ESP pueden ser partícipes de aplicaciones IFTTT, fungiendo principalmente como clientes/servidores HTTP o como clientes MQTT.

Objetivo

Utilizar el servicio IFTTT para publicar un tuit que contenga el valor leído de una de las entradas analógicas del microcontrolador PIC.

Firmware de comandos AT

Se puede utilizar cualquiera de los tres *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION y E_MQTT_V311

Materiales

- 1 computadora
- 1 dispositivo móvil Android/iOS
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 4 resistores de 10 kΩ
- 2 botón pulsador n.o.
- 1 condensador de 0.1 μF
- 1 potenciómetro de 10 kΩ

Descripción

Para esta práctica el PIC-módulo funcionará como un activador al presionar un botón. Cada vez que se active dicho *trigger*, se publicará un tuit con el valor de lectura de una de las entradas analógicas del microcontrolador PIC.

En esta práctica, el módulo puede interactuar con IFTTT al ser establecido como un cliente de Webhook (un servicio propio de IFTTT) que envía peticiones HTTP para desencadenar acciones.

Para crear la *applet* de la práctica, será necesario contar con una cuenta en IFTTT y una en Twitter. Lo primero que se realiza en la creación de la *applet* será seleccionar como activador a "*Receive a web request"* del servicio Webhook, por lo que se crea un evento al cual se nombra como "Tuit" (ver figura P25-B.1), posteriormente de configurar el activador, se seleccionará el servicio Twitter y como acción se elige "*Post a tweet"*, en la configuración de la acción se escribirá como texto del tuit "El valor de la entrada analógica es:" y se agrega un ingrediente (ver figura P25-B.2), el cual corresponde al valor leído por el microcontrolador PIC.

Ya con la *applet* finalizada se deberá ingresar a https://ifttt.com/maker_webhooks y seleccionar la opción de documentación, en esta página se muestra la clave que se ha otorgado para ingresar al servicio web. Además, como se muestra en la figura P25-B.3, en esta página se pueden realizar pruebas de la *applet* creada, en este caso se muestra que se debe ejecutar una petición POST al recurso del activador del evento tuit con la clave que proporciona Webhook

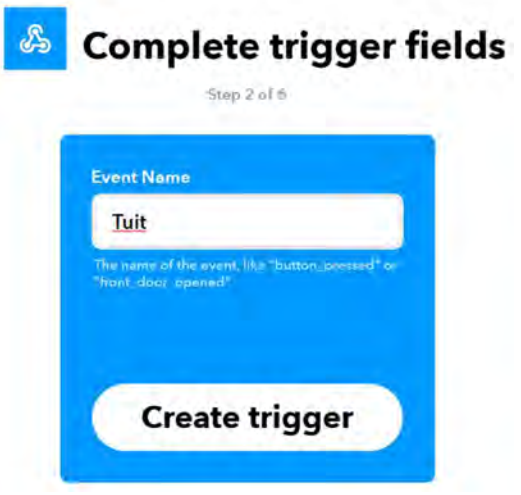


Figura P25-B.1 *Creación del activador con Webhook.*

Figura P25-B.2 *Creación de la acción para publicar un tuit.*

(trigger/Tuit/with/key/...), y el contenido de la petición debe estar en formato JSON, en el que valor del objeto *Value 1* corresponde a la lectura del microcontrolador PIC.

Código

El código que deberá ejecutar el microcontrolador PIC es el mostrado en la figura P25-B.5, en que se deben ingresar los siguientes parámetros:

- Nombre y contraseña de un AP con acceso a internet
- URI el directorio del activador de la *applet*

Circuito

El circuito para llevar a cabo esta práctica se muestra en la figura P25-B.6.

Resultados

Cada vez que se presione el botón en el microcontrolador PIC, se publicará un tuit que indica el valor leído de una de sus entradas analógicas, el cual varía en función de la variación de la posición del eje de un potenciómetro (ver figura P25-B.4).

Figura P25-B.3 *Documentación de Webhook, en la que se muestra como activar la applet.*

En el siguiente enlace está la carpeta que contiene el video del funcionamiento y el código de la práctica:

https://bit.ly/3hMVHoL

Figura P25-B.4 *Publicación de tuis del valor de la entrada analógica del PIC.*

```
#include <16lf1939.h>
#device ADC=10
#include "esp.h"

/*Parámetros a modificar*/
#define SSID "nombre del AP"
#define password "contraseña del AP"
#define uri "/trigger/Tuit/with/key/{clave}"
#define server "maker.ifttt.com"
#define port 443

char str_valor[4];
long codigo_estado_http=0;
char json[]="{\"value1\":\"0000\"}";
long valor=0;

void main()
{
    setup_adc(adc_clock_div_64);
    delay_us(10);
    setup_adc_ports(sAN10);
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("Estacion",SSID,password);
    while(1)
    {
        //Obtención de valor
        set_adc_channel(10);
        valor=(read_adc()>>1);
        //Valor leído es colocado en un string
        sprintf(str_valor,"%04lu",valor);
        //Al presionar botón envía petición HTTP
        if(1==input_state(PIN_B7))
        {
            //Coloca el valor obtenido en el string que se envía
            json[11]=str_valor[0];
            json[12]=str_valor[1];
            json[13]=str_valor[2];
            json[14]=str_valor[3];
            /* En el canal de comunicación 0, se establece un cliente SSL que
            ejecuta una petición POST (3) con contenido json*/
            request_http_client(0, 1,server,3,uri,port, json);
            while(1==input_state(PIN_B7))
            {
                //Se mantiene hasta que se suelte
            }
        }
    }
}
```

Figura P25-B.5 *Código de la práctica 25-B.*

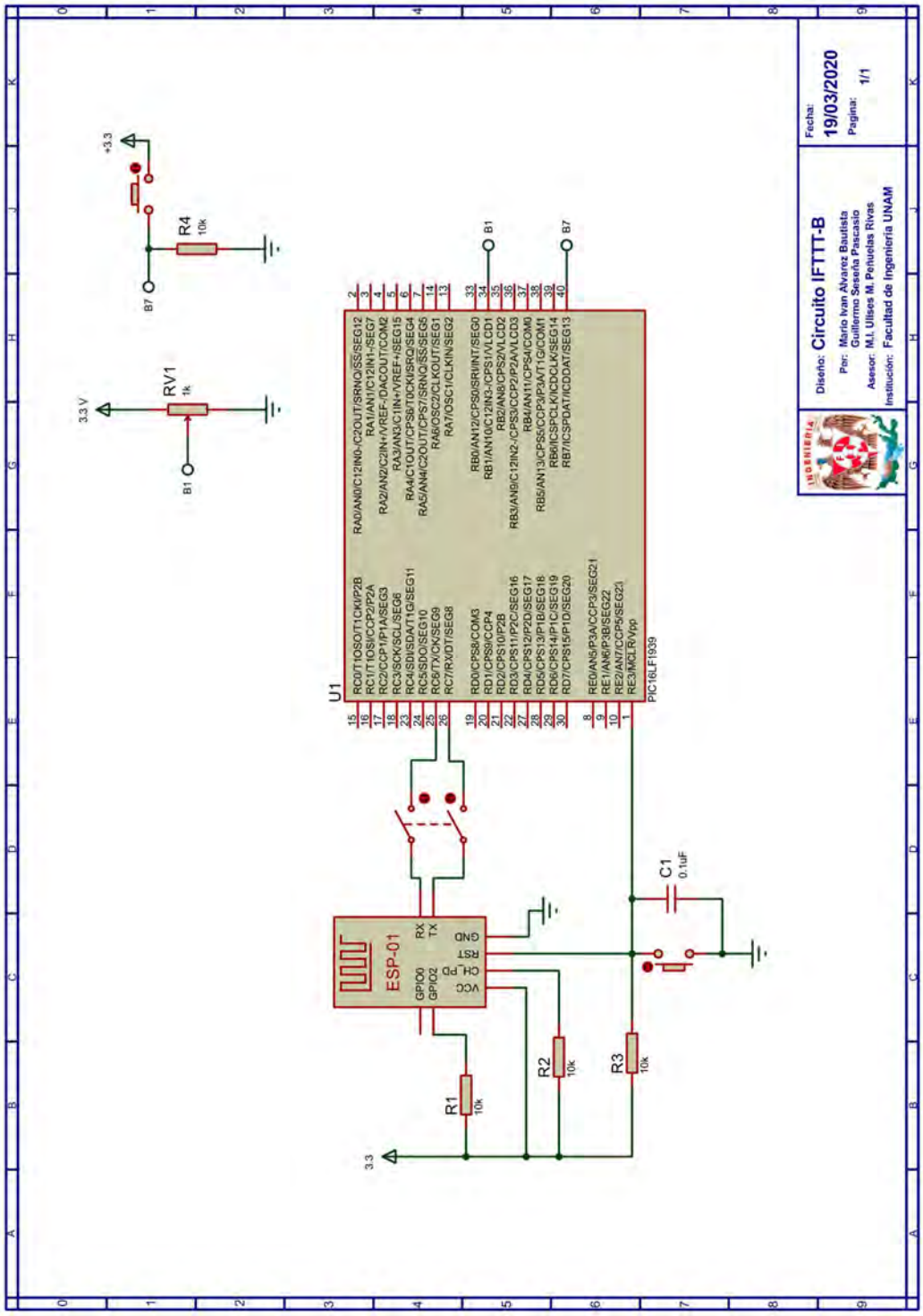


Figura P25-B.6 *Circuito de la práctica 25-B.*

Referencias

1. *IFTTT*. Disponible en:https://ifttt.com/.

Práctica 26 Cliente especial

Introducción

El denominado cliente especial, es un único cliente TCP o SSL que puede ser creado en los módulos que incorporen el SoC ESP8266 para conectarse a un servidor TCP o SSL. Este cliente tiene la peculiaridad de ser capaz de aplicar un filtro a todos los mensajes que se reciben de un servidor o al primero que se recibe tras haber enviado un mensaje al servidor. La finalidad de los filtros es reducir la información que llega al microcontrolador PIC, eliminando parte de la información que no es requerida y, con ello, reducir el número de veces que es llamada la interrupción por recepción de datos RDA para su captura.

Es por esto, que los filtros deben ser desarrollados para aplicaciones particulares, ya que para crearlos es necesario conocer previamente la estructura del mensaje que provendrá desde el servidor. Los filtros además de reducir la información pueden ser creados para segmentar, analizar o almacenar el mensaje o parte de éste en el módulo WiFi, libera recursos en el microcontrolador PIC.

Los filtros deben ser creados en el archivo *user_client.c* que forma parte de los archivos para la modificación del *firmware* de comandos AT. En este caso, en el archivo están desarrollados dos filtros como ejemplo del proceso de creación.

El primer filtro denominado HTTP, es utilizado para únicamente reenviar al microcontrolador el código de estado de una respuesta HTTP. Esta parte correspondería únicamente a 200 OK de la respuesta HTTP que se muestra en la figura P26.1. El segundo filtro denominado HTML, es utilizado para reenviar solamente al microcontrolador el contenido del cuerpo de una respuesta HTTP que esté entre las etiquetas <html> y </html>, incluyendo las propias etiquetas, y excluyendo el resto de la respuesta HTTP.

```
HTTP/1.1 200 OK
Connection: close
Date: Sat, 07 Jul 2007 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 6 May 2007 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

(datos datos datos datos datos ...)
```

Figura P26.1 *Ejemplo de respuesta HTTP.*

La biblioteca ESP permite realizar las siguientes acciones:

- Crear un único cliente especial y éste puede ser TCP o SSL.
- Establecer un *buffer* único para este cliente para almacenar los mensajes que se reciban desde el servidor.

- Comparar el *buffer* con algún string.
- Enviar *strings* alojados en la ROM del PIC (máximo 1000 bytes).
- Enviar *strings* e información binaria (en bytes) alojada en la RAM del PIC (máximo 1000 bytes)
- Cerrar la conexión del cliente.
- Actualmente cuenta únicamente con dos filtros, los cuales pueden ser aplicados a todos los mensajes que se reciban desde el servidor o al primero que se reciba tras el envío de un mensaje desde el cliente al servidor.

Objetivo

Mostrar la utilidad del cliente especial y el impacto para futuras mejoras de la biblioteca.

***Firmware* de comandos AT**

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3 o en el SDK NONOS 2.2.1, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

- STATION y E_SSL_TCP_UDP_CLIENT_UNICON para la primera actividad
- STATION y E_SPECIAL_CLIENT para la segunda y tercera actividad

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 1 adaptador USB-TTL
- 4 resistores de 10 kΩ
- 2 botón pulsador n.o.
- 1 condensador de 0.1 µF

Descripción

En todas las actividades se utilizará un módulo WiFi que deberá estar conectado a un punto de acceso con conexión a internet. Estas actividades son:

- Conexión con cliente TCP
- Conexión con cliente especial TCP con un filtro al primer mensaje
- Conexión con cliente especial TCP con filtro a todos los mensajes

I Conexión con cliente TCP

En esta actividad el módulo creará un cliente TCP que se conectará a iot.espressif.cn en el puerto 80 y enviará una petición GET para obtener el código HTML cuando se presione un botón. Se

deberá usar un adaptador serial para observar en una terminal toda la información que recibe el uC PIC del módulo WiFi.

II Conexión con cliente especial TCP con un filtro al primer mensaje

Se repetirá nuevamente el proceso anterior, pero creando un cliente especial TCP con filtro HTTP aplicado al primer mensaje que se recibe tras la petición.

III Conexión con cliente especial TCP con filtro a todos los mensajes

Se repetirá nuevamente el proceso anterior, pero aplicando el filtro HTTP a todos los mensajes que se reciben.

Las respuestas obtenidas en las tres actividades deben compararse para comprender el impacto del cliente especial. La petición GET para todas las actividades es la siguiente:

GET /index.html HTTP/1.1\r\n
Host: iot.espressif.cn\r\n
Connection: Close\r\n\r\n

Código

El código para cuando el módulo se utiliza como cliente TCP se muestra en la figura P26.7, el código del cliente especial con filtro al primer mensaje se muestra en la figura P26.8 y el código del cliente especial con filtro a todos los mensajes se muestra en la figura P26.9.

En todas las actividades se deberá ingresar nombre y contraseña de un AP con acceso a internet.

```
chksum 0xd5
load 0x3ffe8000, len 3900, room 0
tail 12
chksum 0xf1
ho 0 tail 12 room 4
load 0x3ffe8f40, len 17688, room 12
tail 12
chksum 0x12
csum 0x12
[illegible]
[illegible]
ready
ATE0

OK

OK

OK
WIFI CONNECTED
WIFI GOT IP

OK
```

Figura P26.2 *Respuestas de los comandos AT ejecutados.*

Circuito

El circuito de esta práctica se muestra en la figura P26.10.

Resultados

Como el adaptador USB-TTL estará conectado a la línea de comunicación que RX del uC PIC al ejecutar el código del microcontrolador, se observarán los mensajes de la figura P26.2. Una vez observados todos ellos, se podrá presionar el botón para realizar la petición.

Cuando se envía la petición desde un cliente TCP, la respuesta del servidor es tan grande que el uC PIC recibe tres mensajes como respuesta (ver figura P26.3 y figura P26.4) de 1452, 1452 y 1401 bytes respectivamente, donde el primero contiene la cabecera y parte del cuerpo de la respuesta. Esto provoca que, en total, se ingrese a la interrupción por recepción de datos 4335 veces, 4305 por la información del mensaje de respuesta y 30 debido a los bytes que añade el módulo WiFi como parte de su identificador.

```
+IPD,1452:HTTP/1.1 200 OK␍␊
Server: nginx/1.10.2␍␊
Date: Mon, 09 Mar 2020 19:01:29 GMT␍␊
Content-Type: text/html␍␊
Content-Length: 4071␍␊
Last-Modified: Wed, 18 Jan 2017 11:07:18 GMT␍␊
Connection: close␍␊
ETag: "587f4c66-fe7"␍␊
Accept-Ranges: bytes␍␊
␍␊
```

Figura P26.3 *Cliente TCP, respuesta recibida en la actividad 1.*

La biblioteca está obligada a entrar esta cantidad de veces a la interrupción, a pesar de que sólo sea de interés una parte de la respuesta, para evitar que se confunda parte de la respuesta con un comando AT o evitar mandar una secuencia de comandos antes de tiempo.

Cuando se envía la petición desde un cliente especial TCP con un filtro HTTP que se aplica únicamente al primer mensaje que se recibe tras un envío, el uC PIC reciben como respuesta

```
<!DOCTYPE html>␍␊<html lang="zh-Hans" ng-app="iotApp">␍␊  <head>␍␊    <meta charset="utf-8">␍␊    <meta property="qc:admins" content="2531277324653025
53316636" />␍␊    <meta http-equiv="X-UA-Compatible" content="IE=edge">␍␊    <meta name="viewport" content="width=device-width, initial-scale=1">␍␊
   <meta name="description" content="Internet Of Things @Espressif">␍␊    <meta name="author" content="espressif">␍␊    <link rel="icon" href="http
s://www.espressif.com/sites/all/themes/espressif/favicon.ico" type="image/x-icon">␍␊␍␊    <title>{{ title }} ¤¤ Iot ¤¤ Espressif</title>␍␊␍␊    <link
 href="/static/css/bootstrap.min.css" rel="stylesheet">␍␊    <link href="/static/css/app.css" rel="stylesheet">␍␊␍␊    <script src="/static/js/angul
ar.min.js"></script>␍␊    <script src="/static/js/angular-resource.min.js"></script>␍␊    <script src="/static/js/angular-route.min.js"></script>␍␊
   <script src="/static/js/angular-cookies.min.js"></script>␍␊    <script src="/static/js/angular-sanitize.min.js"></script>␍␊    <script src="/sta
tic/js/ngStorage.min.js"></script>␍␊    <script src="/static/js/jquery.min.js"></script>␍␊    <script src="/static/js/bootstrap.min.js"></script>␍␊
   <script src="/static/js/moment.min.js"></script>␍␊    <␍␊␍␊
+IPD,1452:cript src="/static/js/moment-zh-cn.js"></script>␍␊    <script src="/static/js/moment-zh-tw.js"></script>␍␊    <script src="/static/js/qrc
ode.min.js"></script>␍␊␍␊    <script src="/static/js/highcharts.js"></script>␍␊    <script src="/static/js/exporting.js"></script>␍␊␍␊    <script src=
"/static/js/const.js"></script>␍␊    <script src="/static/js/common.js"></script>␍␊    <script src="/static/js/app.js"></script>␍␊    <script src="/
static/js/controllers.js"></script>␍␊    <script src="/static/js/directives.js"></script>␍␊    <!--script src="/static/js/angular-animate.min.js">
</script>␍␊    <script src="/static/js/angular-loader.min.js"></script>␍␊    <script src="/static/js/angular-touch.min.js"></script-->␍␊  </head>␍␊␍␊
 <body>␍␊    <div id="header-wrapper" ng-controller="userCtrl">␍␊      <div id="header" class="container">␍␊        <ul class="nav nav-pills pull-ri
ght">␍␊          <li ng-if="user != null && user.unread_message_count > 0"><a href="#/user/messages/">{{'message'|xlat}} <span class="label label-
danger"> {{user.unread_message_count}} </span></a></li>␍␊          <li ng-if="user != null"><a href="#/devices/">{{'device-develop'|xlat}}</a></li
>␍␊          <li ng-if="user != null"><a href="#/products/">{{'product-manage'|xlat}}</a></li>␍␊          <li class="dropdown">␍␊            <a href
="javascript:" class="dropdown-toggle" data-toggle="dropdown"><span class="glyphicon glyphicon-th"></span> {{'start'|xlat}} <b class="caret"></b>
</a>␍␊          ␍␊
+IPD,1401:     <ul class="dropdown-menu">␍␊              <li><a href="#/api/">{{'api-docs'|xlat}}</a></li>␍␊              <li class="divider"></li>
␍␊              <li><a href="#/help/">{{'help'|xlat}}</a></li>␍␊            </ul>␍␊          </li>␍␊          <li class="dropdown" ng-if="user != nul
l">␍␊            <a href="javascript:" class="dropdown-toggle" data-toggle="dropdown"><span class="glyphicon glyphicon-user"></span> {{ user.usern
ame }} <b class="caret"></b></a>␍␊            <ul class="dropdown-menu">␍␊              <li><a href="#/user/devices/">{{'my-devices'|xlat}}</a></li
>␍␊              <li><a href="#/user/messages/">{{'my-messages'|xlat}} <span ng-if="user.unread_message_count > 0" class="label label-danger"> {{u
ser.unread_message_count}} </span></a></li>␍␊              <li><a href="#/settings/">{{'settings'|xlat}}</a></li>␍␊              <li class="divider
"></li>␍␊              <li><a ng-click="logout()" href="#/logout/">{{'logout'|xlat}}</a></li>␍␊            </ul>␍␊          </li>␍␊          <li ng-i
f="user == null"><a href="#/join/">{{'join'|xlat}}</a></li>␍␊          <li ng-if="user == null"><a href="#/login/">{{'login'|xlat}}</a></li>␍␊
   </ul>␍␊        <h2 class="logo"><a href="#/"><strong>Iot</strong>¤¤Espressif</a></h2>␍␊      </div>␍␊    </div>␍␊␍␊    <div id="main-wrapper">␍␊
  <div id="main" class="container">␍␊        <div ng-view></div>␍␊      </div>␍␊    </div>␍␊  </body>␍␊</html>␍␊CLOSED␍␊
```

Figura P26.4 *Cliente TCP, respuesta recibida en la actividad 1.*

nuevamente tres mensajes de 6, 1452 y 1401 bytes respectivamente (ver figura P26.6). Esto provoca que en total se entre a la interrupción por recepción de datos 2889 veces, 2859 por la información del mensaje de respuesta y 30 veces debido a los bytes que añade el módulo WiFi como parte de su identificador.

Finalmente, cuando se envía la petición desde un cliente especial TCP con un filtro HTTP que se aplica a todos los mensajes que se recibe, se recibirá un solo mensaje de respuesta de 6 bytes (ver figura P26.5). Esto provoca que en total se entre a la interrupción por recepción de datos 16 veces, 6 veces por la información del mensaje y 10 debido a los bytes que añade el módulo WiFi como parte de su identificador.

Figura P26.5 *Cliente especial TCP, respuesta con filtro aplicado a todos los mensajes que se recibe en la actividad 3*

En el siguiente enlace está la carpeta que contiene el código y video del funcionamiento de esta práctica:

https://bit.ly/37NEpUa

Recomendación

Crear un filtro especial para facilitar el intercambio de información con una plataforma IoT.

Figura P26.6 *Cliente especial TCP, respuesta con filtro HTTP aplicado al primer mensaje que se recibe en la actividad 2.*

```
#include <16lf1939.h>
#include "esp.h"

const char request[]={
"GET /index.html HTTP/1.1\r\nHost: iot.espressif.cn\r\nConnection: Close\r\n\r\n"};

char buffer[20];
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","INFINITUM0101","123456789");
    //Se establce buffer para almacenar mensajes
    set_buffer_client(buffer,sizeof(buffer));
    /*Se crea un cliente TCP*/
    fprintf(PIC,"ready\r\n");
    while(1)
    {
        if(1==input_state(PIN_A1))
        {
            while (1==input_state(PIN_A1))
            {
                // Espera hasta que se deje de presionar el botón
            }
            create_tcp_pclient("iot.espressif.cn",80);
            // Se envía la petición al servidor
            send_client_tcp_const(request);
        }
    }
}
```

Figura P26.7 *Código de la práctica 26, actividad 1, cliente TCP.*

```
#include <16lf1939.h>
#include "esp.h"

const char request[]={
"GET /index.html HTTP/1.1\r\nHost: iot.espressif.cn\r\nConnection: Close\r\n\r\n"};

char buffer[20];
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","INFINITUM0101","123456789");
    //Se establce buffer para almacenar mensajes
    set_buffer_special_client(buffer,sizeof(buffer));
    fprintf(PIC,"ready\r\n");
    while(1)
    {
        if(1==input_state(PIN_A1))
        {
            while (1==input_state(PIN_A1))
            {
                // Espera hasta que se deje de presionar el botón
            }
            create_special_pclient(NONE,TCP_CLIENT,"iot.espressif.cn",80);
            // Se envía la petición al servidor
            send_special_client_const(HTTP_FILTER,request);
        }
    }
}
```

Figura P26.8 *Código de la práctica 26, actividad 2, cliente especial TCP con filtro aplicado al primer mensaje que se recibe.*

```
#include <16lf1939.h>
#include "esp.h"

const char request[]={
"GET /index.html HTTP/1.1\r\nHost: iot.espressif.cn\r\nConnection: Close\r\n\r\n"};

char buffer[20];
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Conexión al punto de acceso indicado
    begin_station("ESP","INFINITUM8052","oEcazdx3sW");
    //Se establce buffer para almacenar mensajes
    set_buffer_special_client(buffer,sizeof(buffer));
    fprintf(PIC,"ready\r\n");
    while(1)
    {

        if(1==input_state(PIN_A2))
        {
            while(1==input_state(PIN_A2))
            {
                // Espera hasta que se deje de presionar el botón
            }
            create_special_pclient(HTTP_FILTER,TCP_CLIENT,"iot.espressif.cn",80);
            // Se envía la petición al servidor
            send_special_client_const(NONE,request);
        }
    }
}
```

Figura P26.9 *Código de la práctica 26, actividad 3, cliente especial TCP con filtro HTTP aplicado a todos los mensajes que se recibe.*

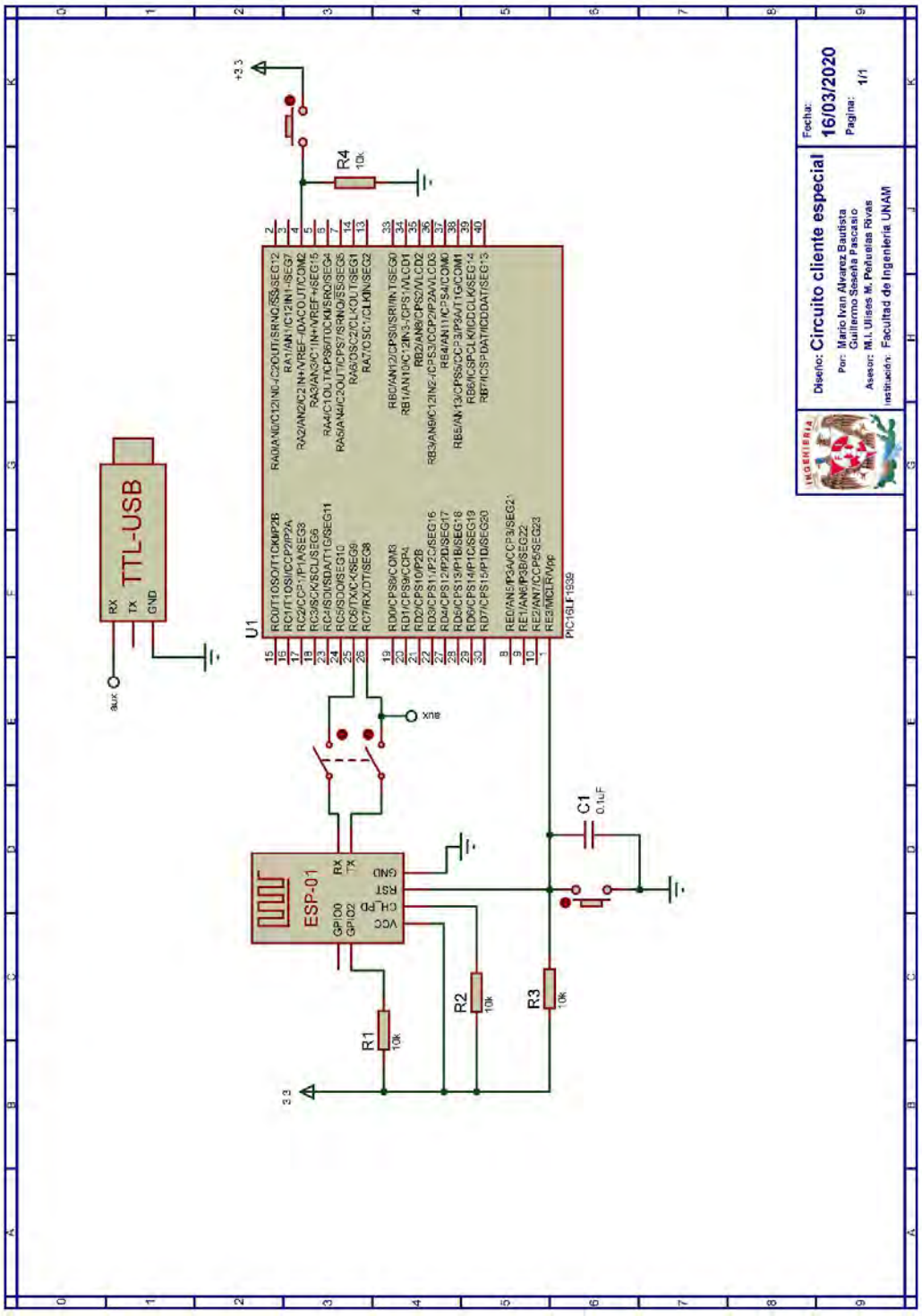


Figura P26.10 *Circuito de la práctica 26.*

Referencias

Para una descripción de las funciones que se utilizan en los achivos donde se crea el filtro, se recomienda consultar:

- Espressif Systems, ESP8266 Non-OS SDK API Reference Version 3.0.1. 2019.

Práctica 27-A *Upgrade* PIC: evento de actualización por *hardware*

Introducción

La actualización del programa de un microcontrolador PIC de manera inalámbrica permite cambiar el programa que el microcontrolador ejecuta sin tener que conectar a éste directamente a una computadora para realizar el proceso, lo cual resulta de utilidad cuando no se tiene un fácil acceso al microcontrolador. En este caso, la actualización está diseñada para llevarse a cabo a nivel local a través de una red WLAN, la computadora donde se ejecuta el programa que realizará la actualización debe incorporar WiFi.

Para la actualización inalámbrica se hicieron modificaciones en el *firmware* de comandos AT para el módulo WiFi y en el programa de licencia libre *Tiny Multi Bootloader v0.11.0,* que es utilizado para actualizar el programa de un microcontrolador PIC de manera alámbrica.

Las modificaciones realizadas en el *firmware* del módulo WiFi contemplaron la creación de una función especial, la cual al ser llamada configurará al módulo para crear un punto de acceso que permita la conexión de una sola estación y creará un servidor TCP en el puerto 100 con la dirección IP 192.168.4.0.

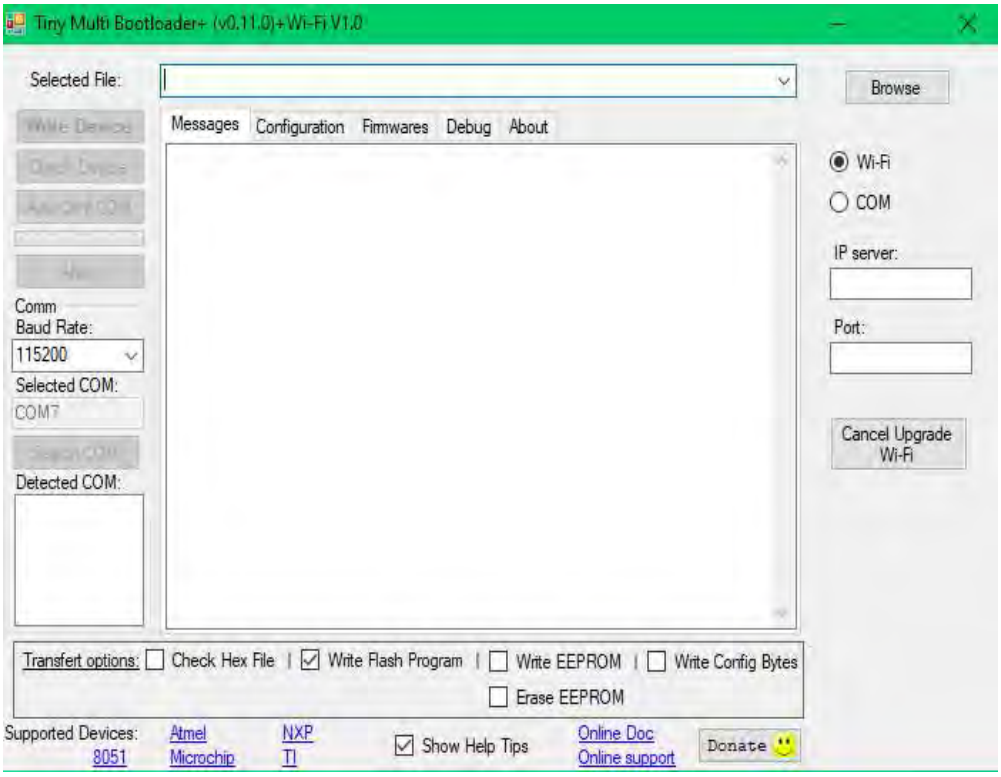


Figura P27-A.1 *Interfaz Tiny Multi Bootloader WiFi.*

Este servidor está programado para ejecutar una secuencia de pasos, que permitan realizar la actualización del programa del microcontrolador.

Mientras que ningún cliente se conecte al servidor para iniciar el proceso de actualización, el módulo WiFi mantendrá, mediante un GPIO de éste que está configurado internamente en *pull-up* (GPIO 0 en SDK NONOS y ESP IDF ESP8266, mientras que ESP IDF 32 será el GPIO 18), en estado de reinicio al microcontrolador PIC. Este proceso que realiza el módulo se conocerá como modo actualización.

Las modificaciones realizadas en el programa *Tiny Multi Bootloader v0.11.0,* contemplan la creación de un cliente TCP para que éste se conecte con el servidor creado por el módulo WiFi. Para esto, el programa debe ejecutarse en una computadora que esté conectada al punto de acceso creado en el módulo. También se realizaron modificaciones en la interfaz del programa (ver figura P27-A.1) para permitir:

- Seleccionar si la actualización se realizará de manera inalámbrica o alámbrica.

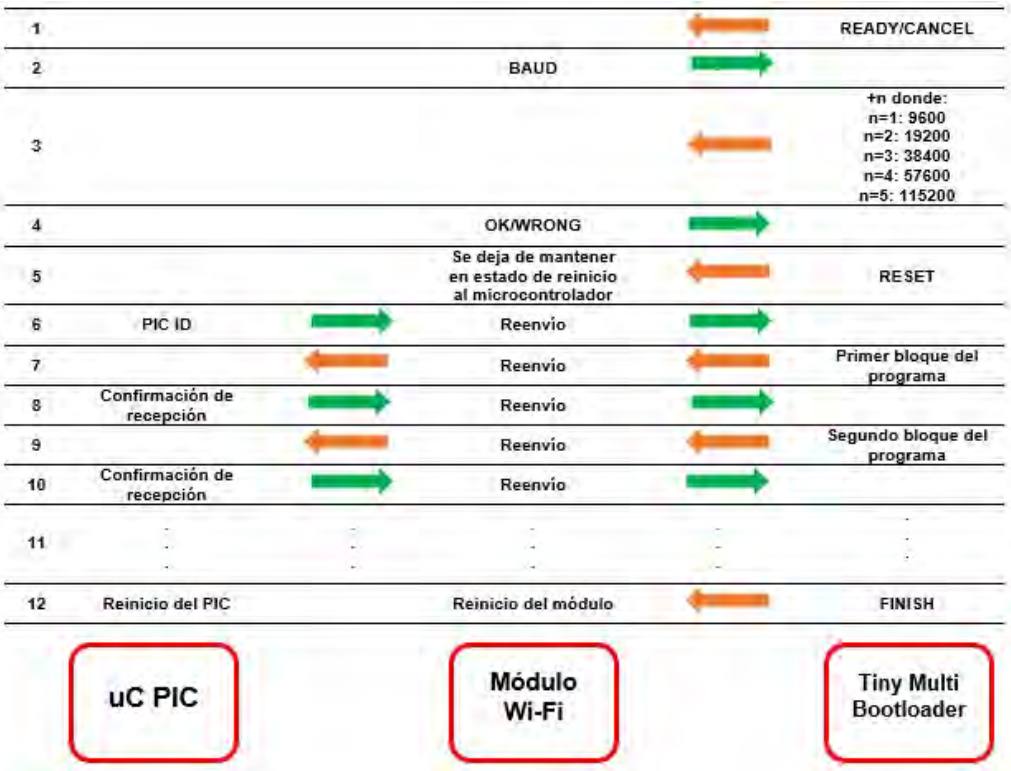


Figura P27-A.2 *Proceso de comunicación entre uC PIC, Módulo WiFi y Tiny Multi Bootloader.*

- Establecer la dirección IP y puerto del servidor creado por el módulo WiFi.
- Establecer el *baud rate* de la comunicación entre cinco velocidades distintas, el *timeout* de cada lectura, el tiempo entre cada envío de la señal de reinicio del microcontrolador y el número total de éstas. Estos parámetros también son configurados en un proceso alámbrico en la pestaña *Configuration.*

Si se cancela el proceso de actualización, tanto el módulo WiFi como el microcontrolador PIC se reiniciarán.

Una vez que el cliente TCP del programa *Tiny Multi Bootloader* se conecte al servidor del módulo, se efectúa un proceso de comunicación entre el microcontrolador PIC, el módulo WiFi y el programa, que se ilustra en la figura P27-A.2

Como se observa en dicha imagen, el encargado de realizar todas las tareas de actualización del programa del microcontrolador es el módulo WiFi, razón por la cual no se requiere modificar el *bootloader* del microcontrolador.

La velocidad de transmisión entre el módulo WiFi y el *Tiny Multi Bootloader* se puede ver afectada por diversos factores como: la distancia a la que se encuentre la computadora donde se ejecuta el programa *Tiny Multi Bootloader*; del módulo WiFi, si existe una línea de vista directa o existen objetos entre éstos, si existen otras redes presentes en la misma área y el tipo de antena del módulo WiFi empleado.

Para realizar la actualización inalámbrica de un programa del uC PIC por evento de *hardware,* el módulo debe presentar un nivel bajo (GND) en el GPIO UPG (GPIO 2 en SDK NONOS y ESP IDF ESP8266, mientras que ESP IDF ESP32 será el GPIO 19) tras 850 *ms* de ser ser reiniciado. Sin embargo, cuando el GPIO UPG corresponde al GPIO 2 se requiere que éste presente un nivel alto (VCC) durante el reinicio para que el módulo encienda de manera correcta y posteriormente una señal en bajo para disparar el evento, por lo que se debe utilizar un ciruito adicional, independiente del que genera la señal, para efectuar el cambio (HtoL).

Objetivo

Mostrar como cambiar el programa que está ejecutando un microcontrolador PIC de manera inalámbrica por *hardware.*

Habilitación de aplicaciones

No requiere la habilitación de ninguna aplicación.

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT,

por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Materiales

Opción 1:

- 1 computadora
- 1 microcontrolador
- 1 módulo WiFi
- 1 temporizador 555
- 3 optoacopladores 4N25
- 1 botón pulsador n.o.
- 1 interruptor DPDT
- 5 resistores de 10kΩ
- 2 resistor de 220Ω
- 1 resistor de 190Ω
- 1 resistor de 20kΩ
- 1 resistor de 500Ω
- 2 resistores de 3.3kΩ
- 1 resistor de 1.5kΩ
- 1 resistor de 1.277 kΩ (usar 1.2 kΩ + 82Ω)
- 1 condensador de 10μF
- 1 condensador de 47μF
- 1 diodo 1n4004
- 1 amplificador operacional LM393N

Opción 2:

- 1computadora
- 2 microcontroladores
- 1 módulo WiFi
- 5 resistores de 10kΩ
- 1 resistor de 220Ω
- 1 botón pulsador n.o.
- 1 interruptor DPDT
- 1 optoacoplador 4N25

Para ambos circuitos:

- 1 resistor de 56 Ω.
- 1 LED de 2.2V y 20 mA.

Descripción

De manera alámbrica, se grabará un programa inicial al microcontrolador PIC que mantendrá encendido un LED. Por medio de un botón, se establecerá al microcontrolador PIC en modo actualización (evento por *hardware*). El programa se actualizará inalámbricamente con un nuevo programa que hará parpadear al LED en intervalos de un segundo.

Código

El código que deberá ejecutar el microcontrolador PIC previo a la actualización se muestra en la figura P27-A.4, mientras que el código que deberá ejecutar posterior a la

actualización se muestra en figura P27-A.6.

Circuito

Para realizar la actualización inalámbrica de un programa del uC PIC por evento de *hardware,* el módulo debe ser reiniciado y tras aproximadamente 850 *ms* del encendido, existe un periodo de 100 *ms* en el que se debe recibir una señal en bajo (GND) que indique si el módulo debe entrar en el modo actualización (la señal se lee cada 10 *ms*).

Para lograr lo anterior, el *firmware de* comandos AT fue modificado para utilizar un GPIO como entrada, en la cual llegará un cambio en señal digital de VCC (3.3 *V*) a GND para indicar la actualización. Mientras que otro GPIO es utilizado como salida para realizar el reinicio del uC PIC, el cual inicialmente tiene una salida en GND.

La primera opción para llevar a cabo la actualización involucra el uso de un circuito para generar la señal de actualización. El circuito propuesto para este fin cuenta con un botón en configuración *pull-up* y un interruptor DPDT. El botón es utilizado para reiniciar tanto el módulo WiFi como el uC PIC o para iniciar el proceso de actualización, el interruptor es el que establece cual será el funcionamiento del botón. Posteriormente se encuentra un temporizador 555 en modo monoestable cuyo pulso de salida debe tener una duración mínima de 878 *ms* para llevar a cabo el proceso de actualización, en este caso el pulso del circuito tendrá una duración de:

$$T_H = 1.1 * R_1 * C_1$$

$$T_H = 1.1 * 20k\Omega * 47\mu F = 1.034\ s$$

A la salida del temporizador se encuentra un diodo en polarización directa, que provoca una caída de 0.6 *V*, para evitar que el circuito RC que se encuentra después, se descargue a través de esta salida. El circuito RC junto con el amplificador operacional en modo comparación que se encuentra después de éste tienen la finalidad de retrasar la señal al menos 30 *ms*, tiempo suficiente para mantener el nivel de voltaje requerido en el GPIO2 (VCC) en la versión SDK NONOS, y para que el módulo pueda encender de manera correcta tras su reinicio. Por cuestiones de diseño, se optó que el tiempo de retraso de la señal fuera la constante de tiempo Ƭ del circuito RC, por lo que los valores de los componentes de dicho circuito son:

$$Ƭ = R * C = 3.3k\Omega * 10\mu F = 0.033\ ms$$

Como al cabo de un tiempo Ƭ (constante de tiempo) el condensador alcanza un 63.2 % de la tensión máxima a la que se alimenta, el voltaje tras la constante de tiempo será de 2.78 *V*.

$$V_c = (V_{555} - V_D)(63.2\%)$$

$$V_c = (V_{555} - V_D)(63.2\%)$$

$$V_c = 4.4 * 0.632 = 2.78\,V$$

Para evitar que el condensador se siga cargando hasta alcanzar la tensión máxima, se agregó un resistor que formará un divisor de voltaje con el resistor de carga del condensador y estará en paralelo con el condensador, lo que limitará la tensión máxima que podrá alcanzar el condensador:

$$V_{C\_max} = V_{in} * \frac{R_2}{R_2 + R_1}$$

$$V_{c\_max} = 4.4V * \frac{6.8k\Omega}{3.3k\Omega + 6.8k\Omega} = 2.96\,V$$

A su vez, será a través de este nuevo resistor la descarga del condensador en un tiempo aproximado de tres constantes de tiempo, ya que el condensador no se carga completamente:

$$T_{des} = 3 * R * C$$

$$T_{des} = 3 * 6.8k\Omega * 10\mu F = 0.204s$$

El nivel de voltaje del condensador irá a la terminal no inversora del comparador, mientras que en la terminal inversora estará conectada a un divisor de voltaje, donde el voltaje de referencia será:

$$V_{inver} = 5V * \frac{1.5k\Omega}{1.5k\Omega + 1.277k\Omega} = 2.70V$$

Finalmente, la señal de salida del amplificador operacional irá hacia un optoacoplador. Como la salida del amplificador operacional sufre una caída de 2 *V* respeto al voltaje de alimentación cuando se supera el voltaje de referencia, se requiere emplear un resistor. Considerando que el LED del optoacoplador trabaja a 1.1 *V* y 10 *mA*, se tiene que el resistor debe ser:

$$R = \frac{1.9V}{10\,mA} = 190\Omega$$

Se utilizan otros dos optoacopladores, uno de ellos para realizar el reinicio del módulo WiFi a partir de la pulsación del botón, mientras que el otro es utilizado para realizar el reinicio del microcontrolador a través de su pin VPP mediante un GPIO del módulo WiFi. El circuito total se muestra en la figura P27-A.7.

La segunda opción para generar la señal de actualización es utilizar un segundo microcontrolador que lleve a cabo las mismas tareas que realiza el circuito. El cual tendría que incorporar el código de la figura P27-A.5 y cuyo circuito se muestra en la figura P27-A.8.

En ambas opciones el GPIO0 del módulo WiFi ya está configurado como una salida para establecer al microcontrolador en estado de reinicio mediante su pin VPP. Este GPIO desde el arranque presenta una salida en alto y

pasa a estado bajo cuando se requiere poner en estado de reinicio al microcontrolador y al igual que el GPIO2, requiere de presentar un estado alto (VCC) para que el módulo pueda arrancar de manera correcta tras su reinicio (de ahí su configuración en el optoacoplador). Se utiliza junto con un optoacoplador para suministrar la corriente necesaria al pin VPP desde la fuente de alimentación.

Resultados

Al colocar el interruptor DPDT en posición para que el botón inicie la actualización del programa del microcontrolador PIC y haber presionado este último, se deberá crear un punto de acceso llamado UPGRADE PIC (ver figura P27-A.3) cuya contraseña por defecto es ESP8266, al cual se deberá conectar la computadora que ejecutará el programa *Tiny Multi Bootloader.*

Una vez conectado al punto de acceso, se deberá seleccionar en el *Tiny Multi Bootloader,* la opción WiFi e ingresar la dirección IP 192.168.4.0, el puerto 100 y seguir el mismo procedimiento como si fuera una actualización alámbrica.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2YelUF9

Figura P27-A.3 *Aparición del punto de acceso para realizar la actualización.*

El tiempo que tardó en realizar la actualización de manera inalámbrica fue de aproximadamente 9.25 segundos, mientras que si ésta se realizara de manera alámbrica sería de aproximadamente 4.84 segundos.

Nota

- Con la finalidad de hacer compatible esta herramienta con todos los módulos WiFi-basados en el SDK NONOS, se usan los GPIO 0 y 2, sin embargo, éstos deben presentar cierto nivel de voltaje durante el arranque del módulo. Si se cambian estos

GPIO en el archivo del *firmware* el circuito se puede simplificar, ya que no se requiere retrasar la señal.

- Para los módulos que utilizan el *firmware* de comandos AT basado en ESP-IDF, el *read time out* debe ser de al menos 400 *ms*, éste se encuentra dentro de *COM and WiFi parameters* en la pestaña de *Configurations.*
- Para los módulos que utilizan el *firmware* de comandos AT basado en SDK-NONOS, el *read time out* debe ser de al menos 200 ms, éste se encuentra dentro de *COM and WiFi parameters* en la pestaña de *Configurations.*
- El programa Tiny modificado se puede descargar en: https://bit.ly/2BsLtJt
- La contraseña del AP es "ESP8266EX" y sólo puede ser cambiada en los archivos del *firmware* del módulo.

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    while(1)
    {
        //Enciende el LED
        output_bit(PIN_A1,TRUE);
        delay_ms(1000);
        //Apaga el LED
        output_bit(PIN_A1,FALSE);
        delay_ms(1000);
    }
}
```

Figura P27-A.6 *Código de la práctica 27-A, posterior a la actualización.*

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    //Enciende un LED
    output_bit(PIN_A1,TRUE);
    while(1)
    {
    }
}
```

Figura P27-A.4 *Código de la práctica 27-A, previo a la actualización.*

```
#include<16lf1939.h>
#fuses  NOWDT,INTRC_IO, NOLVP
#use delay(CLOCK=32 MHz)

void main()
{
    /*Pin que genera la señal de salida, en estado ALTO por defecto*/
    output_bit(PIN_C4,TRUE);
    while(1)
    {
        if(input_state(PIN_A7)==0)
        {
            while (input_state(PIN_A7)==0)
            {
                //Espera a que se deje de presionar el botón
            }
            /*Tiempo de retraso de señal, en estado ALTO para que el módulo
            arranque de manera correcta*/
            delay_ms(33);
            /*Señal, en estado BAJO para que el módulo entre en modo actualización*/
            output_bit(PIN_C4,FALSE);
            delay_ms(1100);
            /*Regresa al estado ALTO por defecto*/
            output_bit(PIN_C4,TRUE);
        }
    }
}
```

Figura P27-A.5 *Código del uC PIC simulando el circuito.*

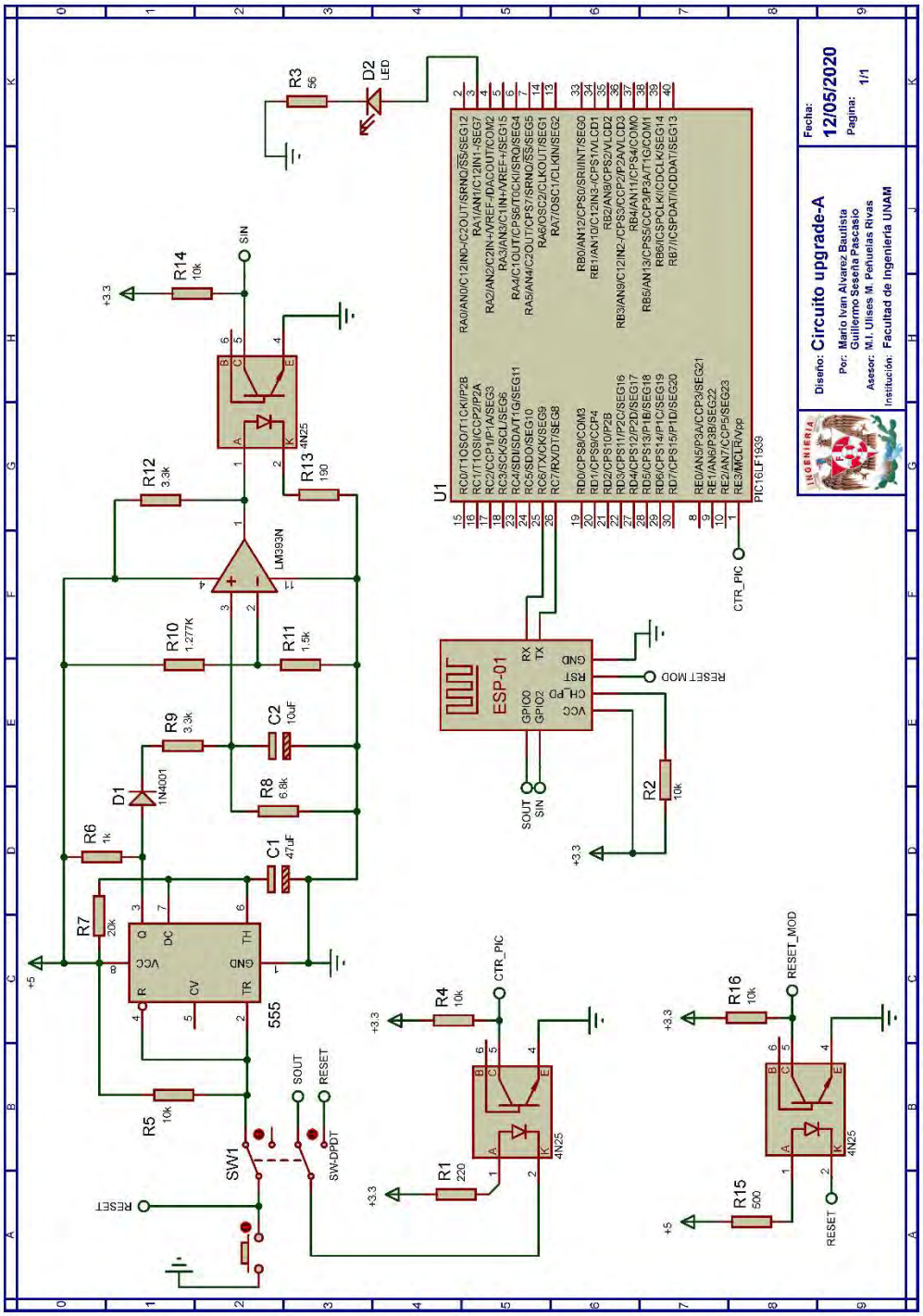


Figura P27-A.7 *Circuito de la práctica 27-A, opción 1.*

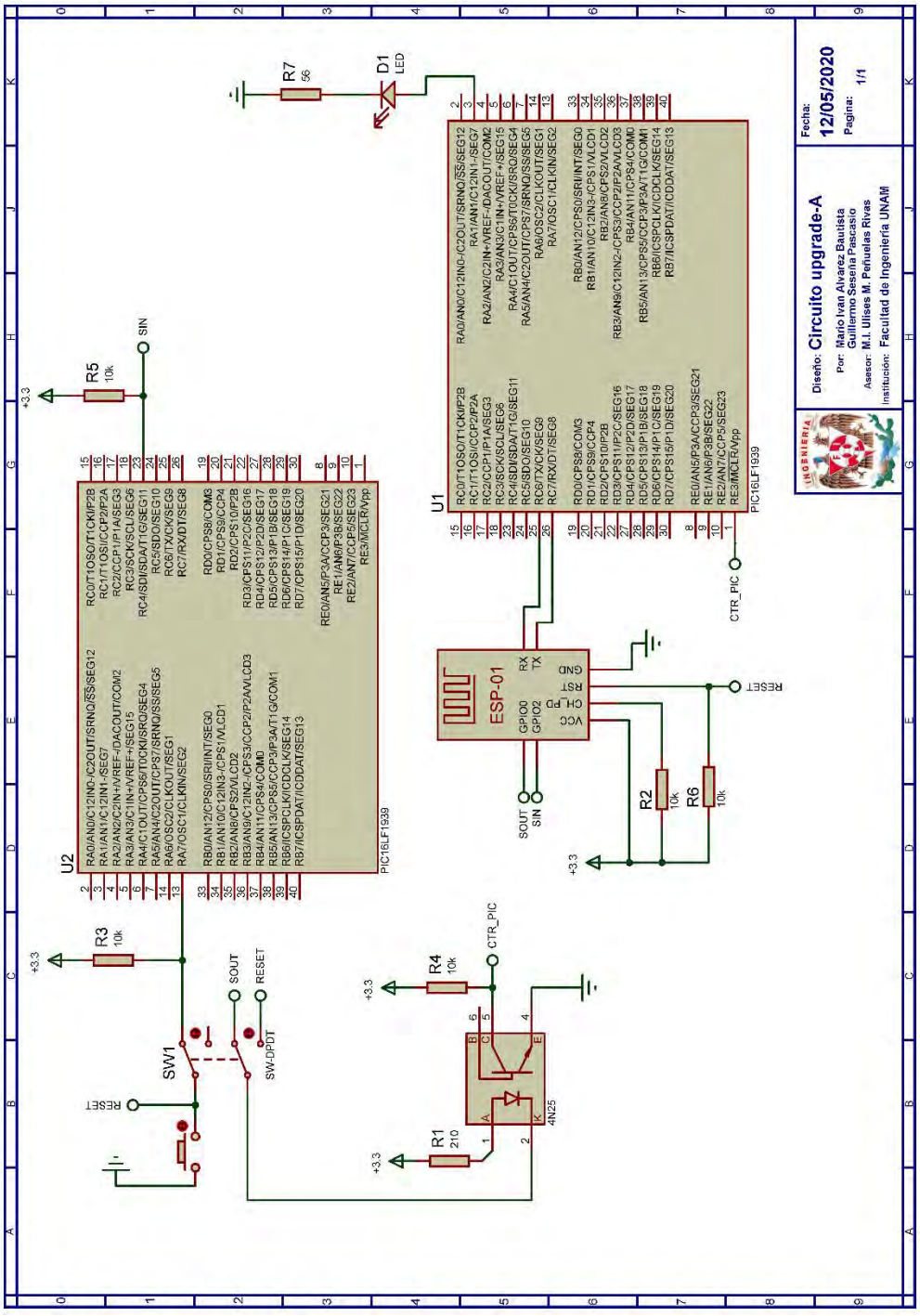


Figura P27-A.8 *Circuito de la práctica 27-A, opción 2.*

Referencias

Para obtener más detalles sobre el programa *Tiny Multi Bootloader* (que no incluye las modificaciones realizadas) se recomienda consultar:

- https://sourceforge.net/projects/tinypicbootload/support

Mientras que para obtener información sobre las funciones utilizadas para crear el servidor de actualización se recomienda consultar:

- Espressif Systems, ESP8266 Non-OS SDK API Reference Version 3.0.1. 2019.
- Espressif Systems. API Reference.[Citado 2020 Marzo]; Disponible en: https://docs.espressif.com/projects/esp-idf/en/v3.3/api-reference/index.html .

Práctica 27-B *Upgrade* PIC: evento de actualización por *software*

Introducción

La actualización del programa de un microcontrolador PIC de manera inalámbrica permite cambiar el programa que el microcontrolador ejecuta sin tener que conectar a éste directamente a una computadora para realizar el proceso, lo cual resulta de utilidad cuando no se tiene un fácil acceso al microcontrolador. En este caso, la actualización está diseñada para llevarse a cabo a nivel local a través de una red WLAN, la computadora donde se ejecuta el programa que realizará la actualización debe incorporar WiFi.

Para la actualización inalámbrica se hicieron modificaciones en el *firmware* de comandos AT para el módulo WiFi y en el programa de licencia libre *Tiny Multi Bootloader v0.11.0,* que es utilizado para actualizar el programa de un microcontrolador PIC de manera alámbrica.

Las modificaciones realizadas en el *firmware* del módulo WiFi contemplaron la creación de una función especial, la cual al ser llamada configurará al módulo para crear un punto de acceso que permita la conexión de una sola estación y creará un servidor TCP en el puerto 100 con la dirección IP 192.168.4.0.

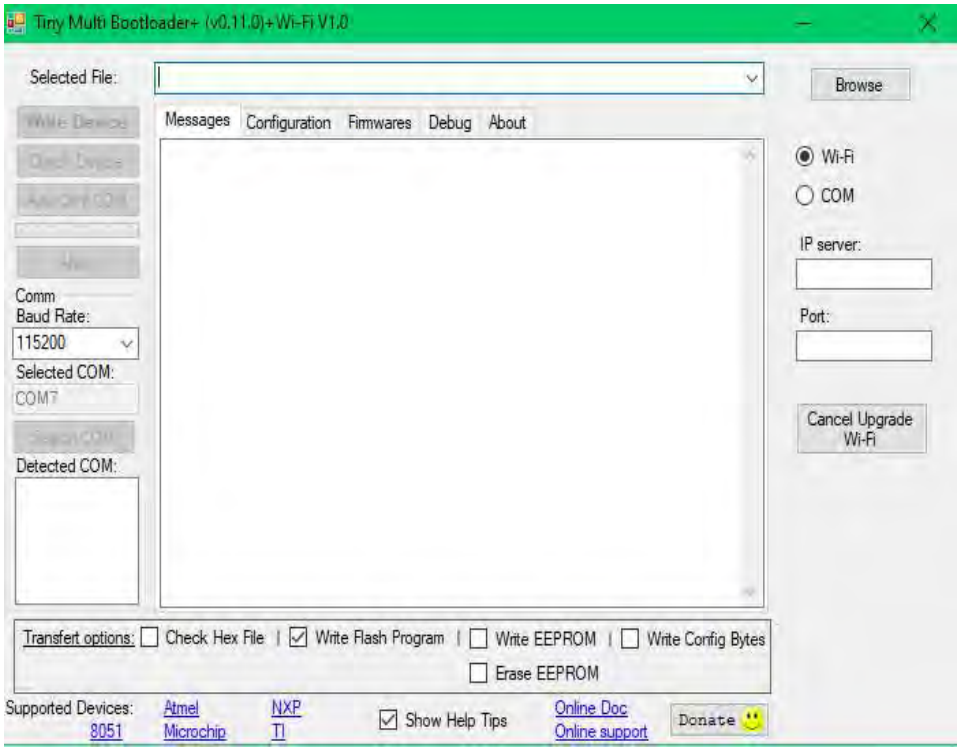


Figura P27-B.1 *Interfaz Tiny Multi Bootloader WiFi.*

Este servidor está programado para ejecutar una secuencia de pasos, que permitan realizar la actualización del programa del microcontrolador.

Mientras que ningún cliente se conecte al servidor para iniciar el proceso de actualización, el módulo WiFi mantendrá, mediante un GPIO que está configurado internamente en *pull-up* (GPIO 0 en SDK NONOS y ESP IDF ESP8266, mientras que ESP IDF 32 será el GPIO 18), en estado de reinicio al microcontrolador PIC. Este proceso que realiza el módulo se conocerá como modo actualización.

Las modificaciones realizadas en el programa *Tiny Multi Bootloader v0.11.0,* contemplan la creación de un cliente TCP para que éste pueda conectarse con el servidor creado por el módulo WiFi. Para esto, el programa debe ejecutarse en una computadora que previamente esté conectada al punto de acceso del módulo. También se realizaron modificaciones en la interfaz del programa (ver figura P27-B.1) para permitir:

- Seleccionar si la actualización se realizará de manera inalámbrica o alámbrica.

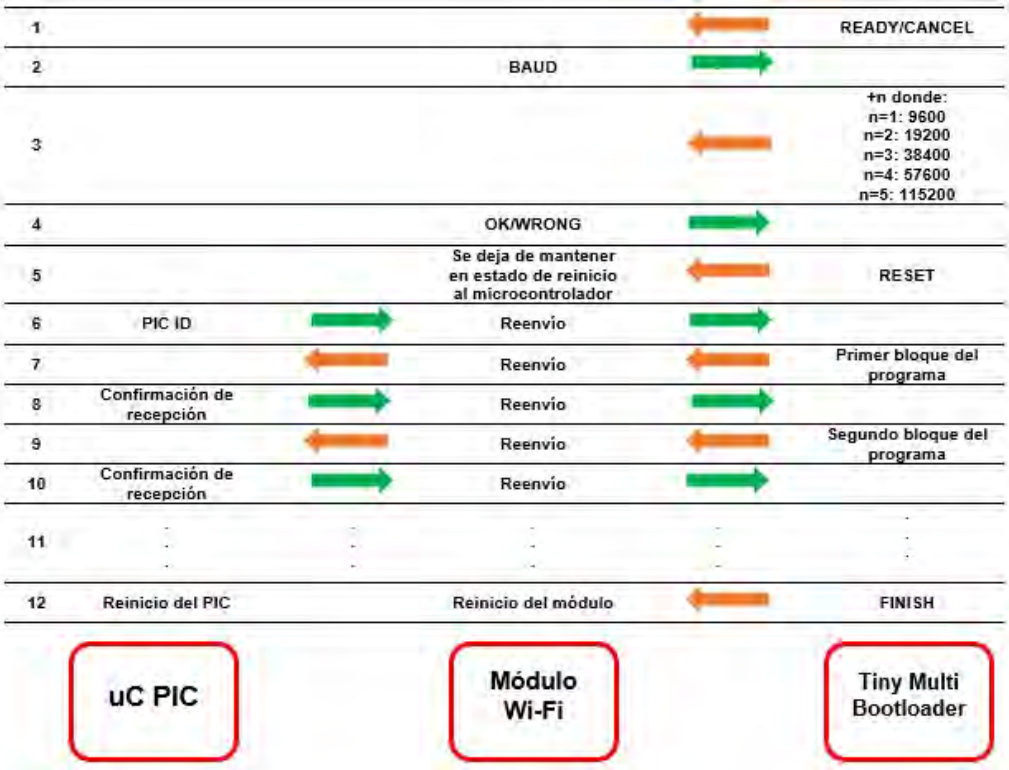


Figura P27-B.2 *Proceso de comunicación entre uC PIC, Módulo WiFi y Tiny Multi Bootloader.*

- Establecer la dirección IP y puerto del servidor creado por el módulo WiFi.
- Establecer el *baud rate* de la comunicación entre cinco velocidades distintas, el *timeout* de cada lectura, el tiempo entre cada envío de la señal de reinicio del microcontrolador y el número total de éstas. Los parámetros también son configurados en un proceso alámbrico en la pestaña *Configuration.*

Si se cancela el proceso de actualización, tanto el módulo WiFi como el microcontrolador PIC se reiniciarán.

Una vez que el cliente TCP del programa *Tiny Multi Bootloader* se conecte al servidor del módulo, se efectúa un proceso de comunicación entre el microcontrolador PIC, el módulo WiFi y el programa, que se ilustra en la figura P27-B.2.

Como se observa en dicha imagen, el encargado de realizar todas las tareas para realizar la actualización del programa del microcontrolador es el módulo WiFi, razón por la cual no se requiere modificar el *bootloader* del microcontrolador.

La velocidad de transmisión entre el módulo WiFi y el *Tiny Multi Bootloader* se puede ver afectada por diversos factores como la distancia a la que se encuentre la computadora donde se ejecuta el programa *Tiny Multi Bootloader,* del módulo WiFi, si existe una línea de vista directa o existen objetos entre éstos, si existen otras redes presentes en la misma área y el tipo de antena del módulo WiFi empleado.

Para realizar la actualización inalámbrica de un programa del uC PIC por evento de *software,* se debe conectar el GPIO UPG (GPIO 2 en SDK NONOS y ESP IDF ESP8266, mientras que ESP IDF ESP32 será el GPIO 19) del módulo a través de una resistor a VCC.

Objetivo

Mostrar como cambiar el programa que está ejecutando un microcontrolador PIC de manera inalámbrica por *software.*

***Firmware* de comandos AT**

El módulo WiFi puede utilizar cualquiera de los cuatro *firmware* de comandos AT, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-01, para lo cual únicamente SDK_V2 debe ser TRUE en esp.h.
- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-01.

Habilitación de aplicaciones

STATION, E_SOFT_UPGRADE_PIC y E_SSL_TCP_UDP_CLIENT_UNICON

Materiales

- 1 computadora
- 1 microcontrolador
- 1 módulo WiFi
- 4 resistores de 10kΩ
- 1 resistor de 1kΩ
- 2 resistor de 220Ω
- 1 botón pulsador n.o.
- 2 optoacoplador 4N25

Descripción

De manera alámbrica, se grabará un programa inicial al microcontrolador PIC, el cual mantendrá encendido un LED. Además, creará un servidor TCP que recibirá mensajes desde una computadora que haya creado un cliente TCP mediante el programa *packet sender.* Estos mensajes serán impresos en una terminal.

Al recibir el mensaje “UPG”, el microcontrolador se pondrá en modo actualización, momento en el cual se cambiará el programa del microcontrolador PIC de manera inalámbrica por uno que haga parpadear un LED en intervalos de un segundo con el programa *Tiny multi bootloader*.

Código

El código que deberá ejecutar el microcontrolador PIC previo a la actualización se muestra en la figura P27-B.6, en el que se deben ingresar nombre y contraseña de un AP.

Mientras que el código que deberá ejecutar posterior a la actualización se muestra en figura P27-B.5.

Circuito

El circuito para esta práctica se muestra en la figura P27-B.7. Este circuito cuenta con dos optoacopladores, el primero de ellos es utilizado para reiniciar el uC PIC cuando se presiona el botón que reinicia el módulo WiFi, mientras que el segundo es utilizado para reiniciar el uC PIC mediante el GPIO0 del módulo WiFi cuando el módulo entra en modo actualización.

Resultados

Se deberá ejecutar el código del uC PIC, una vez que en la terminal parezca la dirección IP que le fue asignada al módulo WiFi y que corresponderá a la del servidor TCP creado en el módulo, se abrirá el programa *packet sender,* se creará un cliente TCP y se comenzará a enviar mensajes al servidor, los cuales deben aparecer en la terminal (ver figura P27-B.3).

El módulo al recibir el mensaje UPG, cerrará la conexión y se reiniciará, posteriormente se deberá crear un punto de acceso llamado UPGRADE PIC (ver figura P27-B.4) cuya contraseña por defecto es ESP8266EX, al cual se

deberá conectar la computadora que ejecutará el programa *Tiny Multi Bootloader.*

Una vez conectada la computadora al punto de acceso, se deberá seleccionar en el *Tiny Multi Bootloader,* la opción WiFi e ingresar la dirección IP 192.168.4.0, el puerto 100 y seguir el mismo procedimiento como si fuera una actualización alámbrica.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/37K8MKY

Figura P27-B.4 *Aparición del punto de acceso que permite la actualización.*

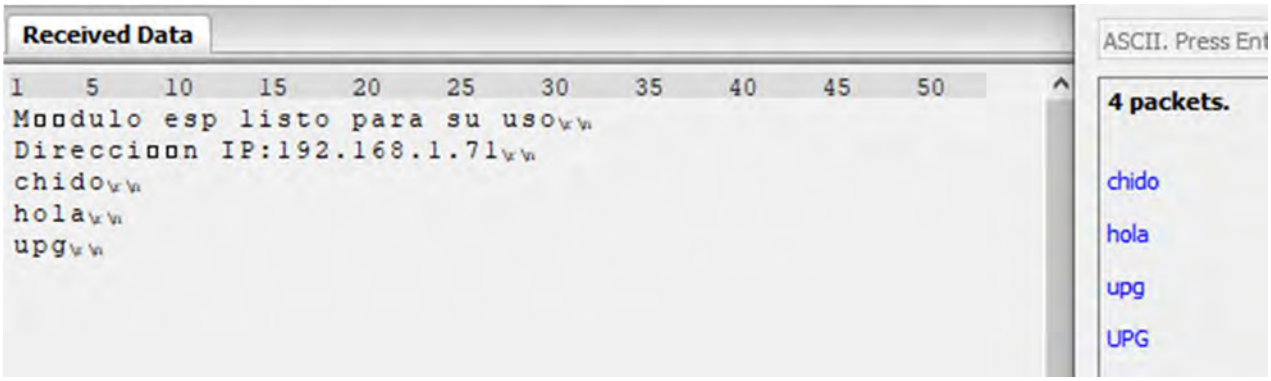


Figura P27-B.3 *Recepción de mensajes en el servidor TCP.*

Nota

- Para los módulos que utilizan el *firmware* de comandos AT basado en ESP-IDF, el *read time out* debe ser de al menos 200 ms, éste se encuentra dentro de *COM and WiFi parameters* en la pestaña de *Configurations.*
- Para los módulos que utilizan el *firmware* de comandos AT basado en SDK-NONOS, el *read time out* debe ser de al menos 200 ms, éste se encuentra dentro de *COM and WiFi parameters* en la pestaña de *Configurations.*
- El programa Tiny modificado se puede descargar en: https://bit.ly/2BsLtJt
- En el archivo de cabecera de la biblioteca esp.h está la definición PIC_PASSWORD correspondiente a la contraseña que tomará el punto de acceso cuando se realiza la actualización del programa del PIC de manera inalámbrica por un evento de *software*. Por defecto es “ESP8266EX”.

```
#include <16lf1939.h>
#include "esp.h"

void main()
{
    while(1)
    {
        //Enciende el LED
        output_bit(PIN_A1,TRUE);
        delay_ms(1000);
        //Apaga el LED
        output_bit(PIN_A1,FALSE);
        delay_ms(1000);
    }
}
```

Figura P27-B.5 *Código de la práctica 27-B, posterior a la actualización.*

```
#include <16lf1939.h>
#include "esp.h"

extern _bool new_message_server1;
char server_buffer[20]={0};

void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Estado inicial del LED
    output_bit(PIN_A1,TRUE);
    //Conexión al punto de acceso indicado
    begin_station("ESP","SSID","123456789");
    /* obtención de la dirección IP asignada al módulo y que
    será la dirección IP del servidor */
    get_ip_station(server_buffer, sizeof(server_buffer));
    fprintf(PIC,"Dirección IP:%s\r\n",server_buffer);
    // Se establece el buffer  para  recibir mensajes
    set_buffer_server1(server_buffer,sizeof(server_buffer));
    // Se crea el servidor TCP en el puerto 80 con un timeout de 300
    create_server1(80,2,300);
    while(1)
    {

        //¿Se recibió el mensaje UPG?
        if(1==compare_S1buffer("UPG",1))
        {
            start_upgrade_pic();
        }
        if(new_message_server1==1)
        {
            delay_ms(100);
            fprintf(PIC,"%s\r\n",server_buffer);
            new_message_server1=0;
        }
    }
}
```

Figura P27-B.6 *Código de la práctica 27-B, previo a la actualización.*

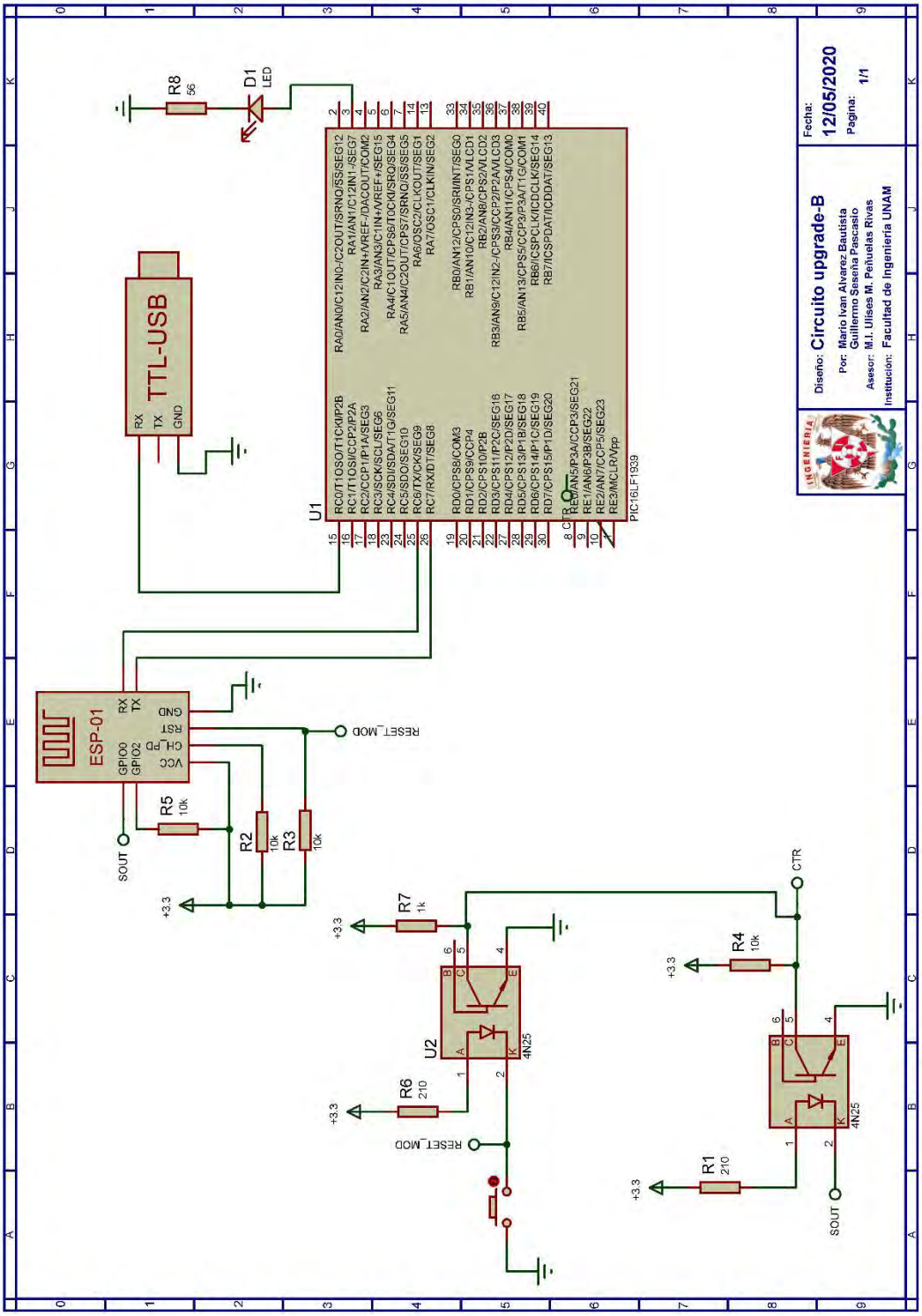


Figura P27-B.7 *Circuito de la práctica 27-B.*

Referencias

Para obtener más detalles sobre el programa *Tiny Multi Bootloader* (que no incluye las modificaciones realizadas) se recomienda consultar:

- https://sourceforge.net/projects/tinypicbootload/support

Mientras que para obtener información sobre las funciones utilizadas para crear el servidor de actualización se recomienda consultar:

- Espressif Systems, ESP8266 Non-OS SDK API Reference Version 3.0.1. 2019.
- Espressif Systems. API Reference.[Citado 2020 Marzo]; Disponible en: https://docs.espressif.com/projects/esp-idf/en/v3.3/api-reference/index.html .

Práctica 28 GPIO

Introducción

Se conoce como pines de propósito general (*general purpose Input/Output*-GPIO) a los pines de un microcontrolador que pueden ser configurados como entradas o salidas, utilizados con actuadores o sensores, o con algún otro componente que requiera comunicarse con el microcontrolador [1]. El SoC ESP8266 cuenta con 17 GPIOs, mientras que los SoC ESP32 cuentan con 34 GPIOs, los cuales pueden ser configurados internamente como *pull-up*, *pull-down* o alta impedancia. El número GPIOs disponibles para el usuario puede variar, ya que algunos de estos GPIOs pueden estar reservados para ejecutar otras funciones como SDIO, UART o SPI modificando los registros adecuados [2, 3] y dependen de que el *hardware* del módulo brinde acceso a éstos.

En general, en los módulos WiFi y tarjetas basadas en éstos, vienen marcados los GPIOs que el usuario puede utilizar.

La biblioteca ESP permite realizar las siguientes acciones:

- Utilizar los GPIO de los módulos WiFi.
- Configurar cada GPIO como entrada o salida.
- Configurar el GPIO internamente como *pull-up* o *pull down* (Dependiendo del SoC).

Objetivo

Mostrar cómo utilizar los GPIOs incorporados en el módulo WiFi.

Firmware de comandos AT

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3 o en ESP-IDF, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP32-DevKitC

Habilitación de aplicaciones

STATION o ACCESS_POINT y E_GPIO deben ser TRUE en esp.h

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 3 resistores de 10 kΩ
- 2 botones pulsadores n.o.
- 1 condensador de 0.1 μF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 *mA*

Descripción

Se controlará el estado de un LED mediante un botón pulsador. Cuando el botón pulsador esté presionado, el LED estará encendido y cuando éste se libere, el LED se apagará. Tanto el LED como el botón pulsador estarán conectados a 2 GPIO del módulo WiFi que estarán configurados como salida y entrada respectivamente.

Código

El código de esta práctica se muestra en la figura P28.1.

Circuito

El circuito de esta práctica se muestra en la figura P28.2

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC, el LED estará encendido inicialmente puesto que el botón está en configuración *pull-down.* Al mantener presionado el botón, el LED deberá mantenerse apagado, puesto que a la entrada el GPIO asociado al botón está presente un voltaje VCC.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2BjZ3z5

Notas

- En la versión del *firmware* de comandos AT basado en el SDK NONOS 3.0.0 se ha bloqueado el uso de los GPIO 0 y 2, mientras que en la versión basada en ESP-IDF se ha bloqueado el uso de los GPIO 18 y 19.

```
#include <16lf1939.h>
#include "esp.h"

uint8 estado=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    //Configuración de los GPIO
    gpio_mode(12,0); // GPIO 12 como entrada
    gpio_mode(25,1); // GPIO 25 como salida
    /*Se indica que se terminó de configurar los GPIO
    que se utilizarán en esta práctica*/
    gpio_ready();
    while(1)
    {
        // Lectura del estado en el  GPIO 12
        estado=gpio_read(12);
        // ¿VCC en GPIO 12?
        if(estado==1)
        {
            gpio_write(25,0); // Se paga LED
        }
        else
        {
            gpio_write(25,1); // Se enciende LED
        }
    }
}
```

Figura P28.1 *Código de la práctica 28.*

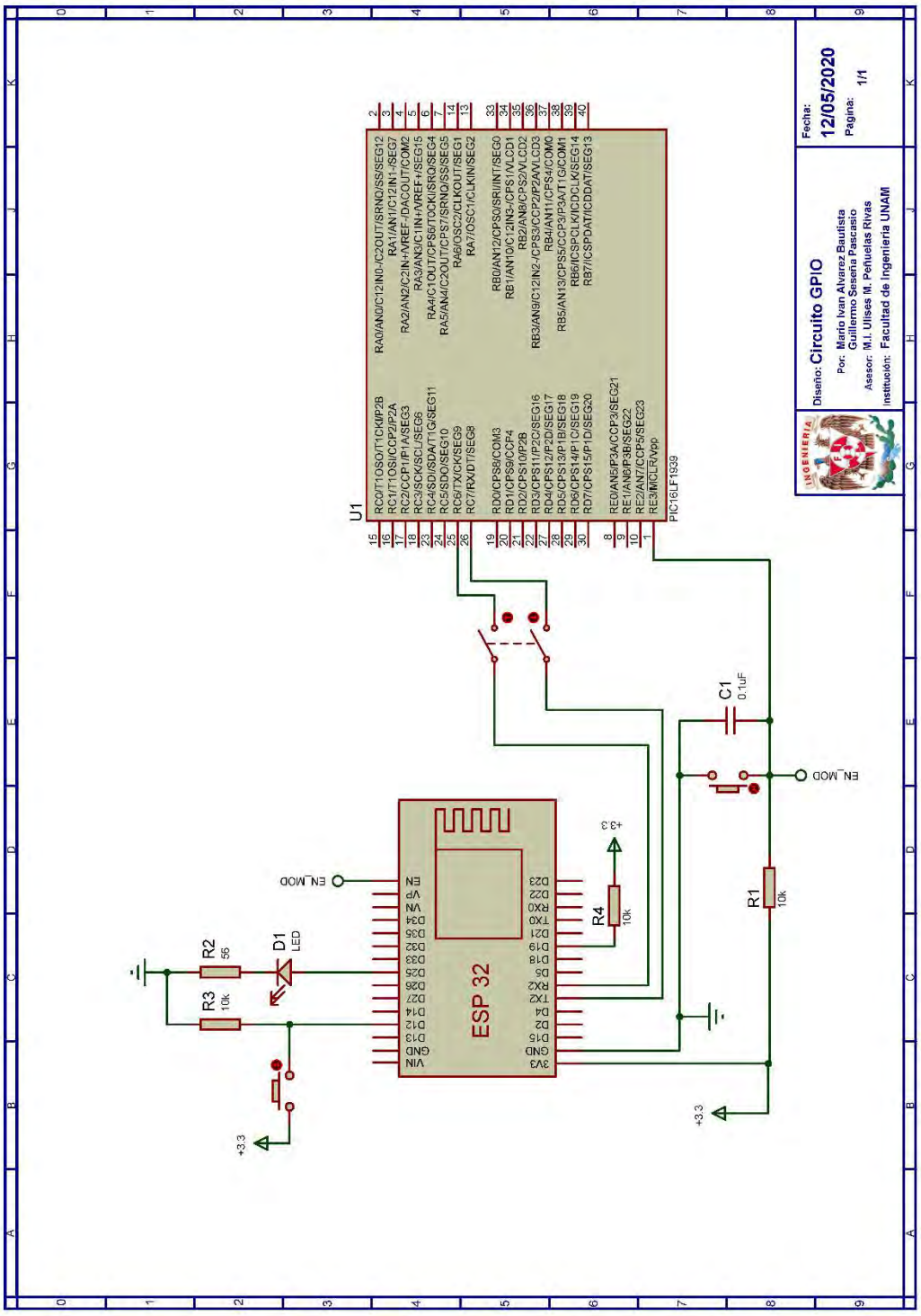


Figura P28.2 *Circuito de la práctica 28.*

Práctica propuesta

Utilizar los GPIO del módulo WiFi para controlar un *display* de 7 segmentos.

Referencias

1. Artero, Ó.T., *Arduino: Curso práctico de fomación*. 2013, España: RC Libros.
2. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.
3. Espresssif Systems, *ESP32 Series DataSheet Version 3.2*. 2019.

Práctica 29 PWM

Introducción

La modulación por ancho de pulsos (*Pulse Width Modulation*, PWM) es una técnica para dividir una señal en una cadena de pulsos de frecuencia constante cuyas alturas se relacionan con la magnitud del voltaje de la señal de entrada. Al variar el ciclo de trabajo de estos pulsos, es decir, la fracción de cada ciclo en la cual el voltaje es alto (ver figura P29.1), se puede controlar el valor promedio del voltaje resultante [1, 2]. Un ciclo de trabajo pequeño resultará en un menor voltaje promedio de salida, mientras que un ciclo de trabajo grande resultará en un mayor voltaje promedio de salida [2].

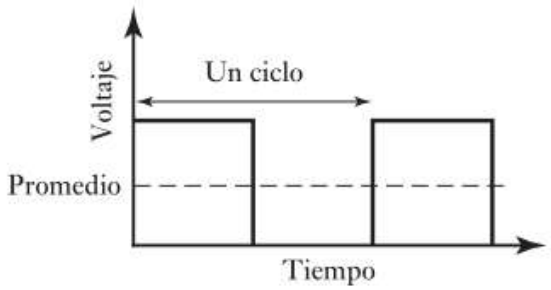


Figura P29.1 *Señal PWM[1].*

Los módulos WiFi que implementan el SoC ESP8266 y utilizan la versión SDK NONOS cuentan con cuatro pines PWM (ver tabla P29.1), los cuales compartirán la misma frecuencia pero pueden tener diferente ciclo de trabajo[3, 4]. Esta frecuencia debe estar entre 100 *Hz* y 1 *kHz* (100 *μs* y 10000 *μs*)[3].

Tabla P29.1 *Definiciones de PWM[3].*

GPIO	Canal PWM por defecto
GPIO 12	PWM0
GPIO 15	PWM1
GPIO 14	PWM2
GPIO 4	PWM3

La resolución mínima que se puede alcanzar es de 45 *ns* a una tasa de actualización de 1 *kHz*, mientras que el ciclo de trabajo mínimo con el que se puede trabajar es de 1/22222 y se puede con una resolución de hasta 14 bits a una tasa de actualización de 1 *kHz* [3, 4]. El valor máximo que el ciclo de trabajo puede está definido por:

$$duty_{max} = period * \frac{1000}{45}$$

donde el periodo debe estar especificado en microsegundos.

En los módulos WiFi ESP32 hay dos APIs para PWM, en este caso la biblioteca ESP utiliza PWM basado en la API *LED Control*, en la cual se hace uso de los ochos canales que operan en modo *low speed* que permiten al controlador cambiar el ciclo de trabajo PWM por *software.* La frecuencia y la resolución del ciclo de trabajo son interdependientes. Cuanto mayor sea la frecuencia PWM, una menor resolución del ciclo de trabajo estará disponible y viceversa. Los valores del ciclo de trabajo van desde cero hasta el determinado por:

$$duty\ values = 2^{duty\ resolution} - 1$$

La frecuencia máxima disponible que se puede alcanzar es de 40 MHz con una resolución del ciclo de trabajo de 1 bit. [5].

La biblioteca ESP permite realizar principalmente las siguientes acciones en ambos SoC:

- Configurar los canales PWM y habilitar su respectivo GPIO.
- Actualizar el valor del periodo o frecuencia conforme a la versión del *firmware* de comandos AT.
- Actualizar el valor del ciclo de trabajo de un canal en específico.

Objetivo

Mostrar cómo utilizar el PWM incorporado en el módulo WiFi.

***Firmware* de comandos AT**

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3 o ESP-IDF, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S y únicamente SDK_V3 debe ser TRUE en esp.h.

Habilitación de aplicaciones

STATION o ACCESS_POINT, y E_PWM

Materiales

- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 interruptor DPST
- 4 resistores de 10 kΩ
- 3 botones pulsadores n.o.
- 1 condensador de 0.1 µF
- 1 resistor de 56 Ω
- 1 LED de 2.2V y 20 *mA*

Descripción

Se controlará la intensidad de luz de un LED, utilizando el módulo PWM que tiene incorporado el módulo WiFi a través del GPIO 14. Para esto se utilizarán dos botones, uno que incrementará la intensidad de luz, mientras que el otro la reducirá. La frecuencia de la señal PWM debe ser de 222 *Hz* (4500 *µs*), por lo que el ciclo de trabajo máximo será de aproximadamente 100,000. El valor inicial del ciclo de trabajo debe ser de 50,000 y los incrementos como decrementos deben ser de 10,000.

Código

El código de esta práctica se muestra en la figura P29.2, mientras que en la figura P29.3 se muestra un código de ejemplo para la versión IDF.

Circuito

El circuito de esta práctica se muestra en la figura P29.4.

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC, el LED deberá estar encendido a la mitad de su intensidad máxima. Al presionar los botones asociados al microcontrolador se deberá observar como el LED varía su intensidad luminosa, lo cual comprobará que los cambios del ciclo de trabajo se están llevando a cabo correctamente.

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/2APPs2Q

Notas

- En el código del programa se debe declarar que se utiliza un solo canal PWM para esta práctica.
- IDF es exclusivo para ESP32 en esta aplicación.

Práctica propuesta

Utilizar el PWM del módulo WiFi para controlar un LED RGB y mostrar al menos 5 colores diferentes.

```
#include <16lf1939.h>
#include "esp.h"

uint32 duty=50000; //Ciclo de trabajo inicial
void main()
{
  //Inicialización del módulo Wi-Fi
  begin_esp();
  /*Se establece el orden de los canales PWM.Como el GPIO 14 será
  el único que se utilizará, éste debe ir primero(canal PWM0) */
  setup_gpio_pwm(14,15,12,4);
  /*Inicia el módulo PWM con  un periodo de 4500 us, como sólo se utilizará  un
  GPIO (canal PWM0), sólo se especifica el ciclo de trabajo para este canal,
  si fueran dos canales sería:    begin_pwm(4500,duty1,duty2);
  */
  begin_pwm(4500,duty);
  while (1)
  {
    // ¿Se ha presionado únicamente el botón de incrementos?
    if(1==input_state(PIN_D0) && 0==input_state(PIN_D7) && duty <100000)
    {
      while(1==input_state(PIN_D0))
      {
        //Espera que se deje de presionar el botón
      }
      duty=duty+10000; // Se realiza el incremento del ciclo de trabajo
      set_duty_pwm(duty,0); // Se estable el nuevo ciclo de trabajo
    }
    // ¿Se ha presionado únicamente el botón de decrementos?
    if(1==input_state(PIN_D7) && 0==input_state(PIN_D0) && duty>0)
    {
      while(1==input_state(PIN_D7))
      {
        //Espera que se deje de presionar el botón
      }
      duty=duty-10000; // Se realiza el decremento del ciclo de trabajo
      set_duty_pwm(duty,0);// Se estable el nuevo ciclo de trabajo
    }
  }
}
```

Figura P29.2 *Código de la práctica 29.*

```c
#include <16lf1939.h>
#include "esp.h"

uint32 duty=511; //Ciclo de trabajo inicial
void main()
{
  //Inicialización del módulo Wi-Fi
  begin_esp();
  /*Se configura el temporizador cero de los 4 temporizadores
  pwm con una resolución de 10 bits  y una frecuencia de 222 Hz*/
  setup_timer_pwm(0,10,222);
  /* Se configura el canal 0 PWM para utilizar el GPIO14, el timer 0
  y un duty de 511 (50%) */
  begin_pwm(0,14,0,duty);
  while (1)
  {
    // ¿Se ha presionado  el botón de incrementos?
    if(1==input_state(PIN_D0) && duty <1000)
    {
      while(1==input_state(PIN_D0))
      {
        //Espera que se deje de presionar el botón
      }
      duty=duty+100;
      // Se establece el nuevo ciclo de trabajo
      set_duty_pwm(duty,0);
    }
    // ¿Se ha presionado el botón de decrementos?
    if(1==input_state(PIN_D7) && duty>0)
    {
      while(1==input_state(PIN_D7))
      {
        //Espera que se deje de presionar el botón
      }
      duty=duty-100;
      // Se establece el nuevo ciclo de trabajo
      set_duty_pwm(duty,0);
    }
  }
}
```

Figura P29.3 *Ejemplo de práctica 29 en IDF ESP32*

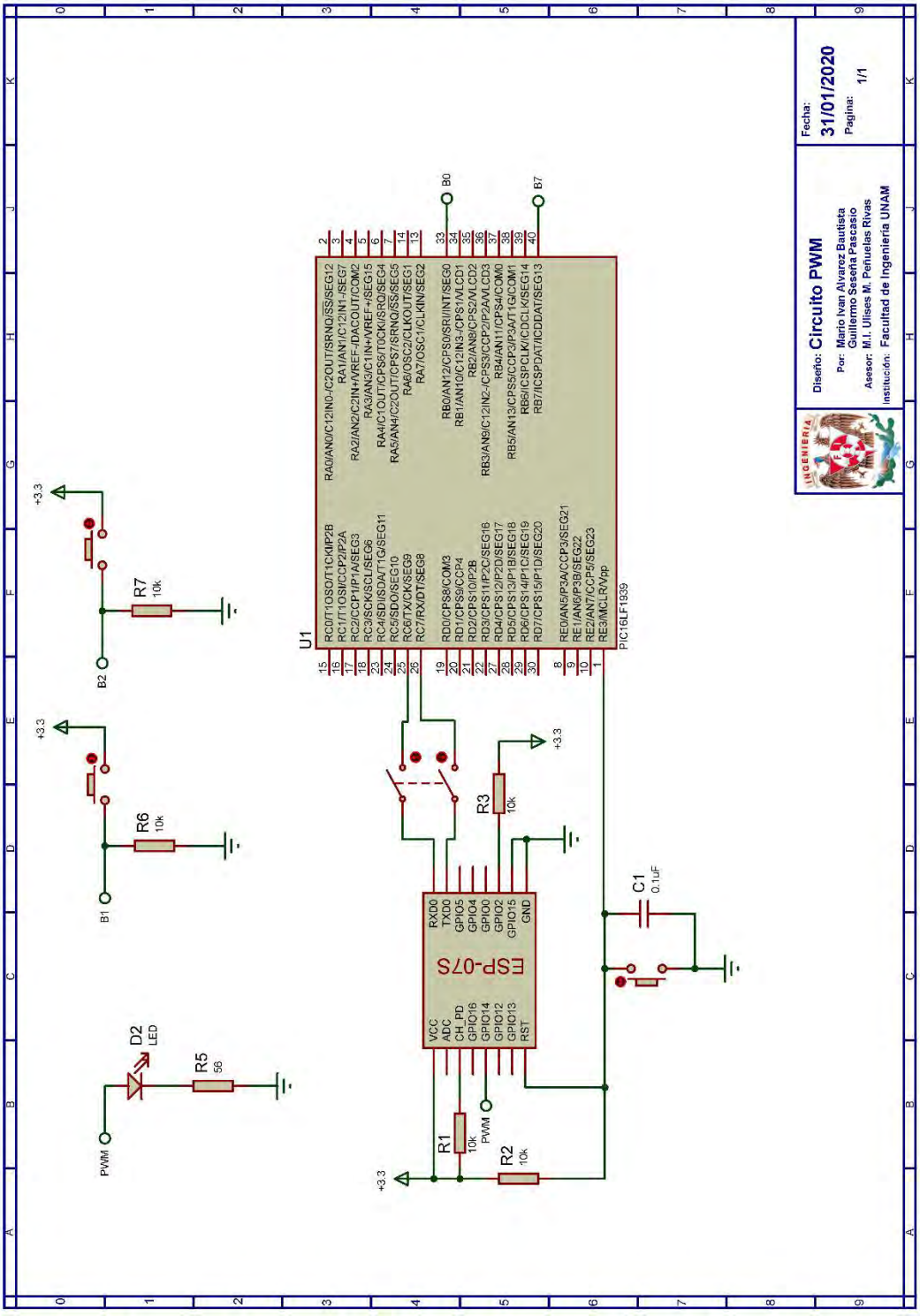


Figura P29.4 *Circuito de la práctica 29.*

Referencias

1. Alciatore, D.G. and M.B. Histand, *Introduction to Mechatronics and Measurement Systems*. 4ta ed. 2012: McGraw-Hill.
2. Artero, Ó.T., *Arduino: Curso práctico de fomación*. 2013, España: RC Libros.
3. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.
4. Espressif Systems, *ESP8266 Technical Reference v1.4*. 2019.
5. Espressif Systems. *Get Started*. ESP-IDF Programming Guide [Citado 2020 Marzo]; Disponible en:https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html.

Práctica 30 Convertidor analógico a digital

Introducción

El convertidor analógico a digital (ADC) toma una señal analógica y la convierte en una palabra binaria que representa el nivel de la señal de entrada [1]. En este caso, los módulos WiFi que implementan el SoC ESP8266 cuentan con un convertidor analógico-digital de aproximaciones sucesivas de 10 bits de precisión, por lo que el mayor número binario que puede alcanzar es 1024 y tiene una resolución de $1/1024\,V$ [2, 3].

El rango de voltaje de entrada es de 0 a $1.0\,V$, por lo que se debe utilizar un circuito externo para evitar exceder el voltaje máximo de entrada [4]. Este circuito externo puede ser un divisor de voltaje (ver figura P30.1), cuyos valores son determinados por:

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$

donde V_{out} corresponde a ADC de la figura P30.1, que estará conectado al pin ADC del módulo WiFi, mientras que V_{in} corresponde a ADC EXTERNO de la misma figura y que estará conectada a la fuente de la señal analógica.

Algunas de las tarjetas de desarrollo que están construidas a partir de los módulos WiFi que incorporan el SoC ESP8266 ya cuentan con un divisor interno para el

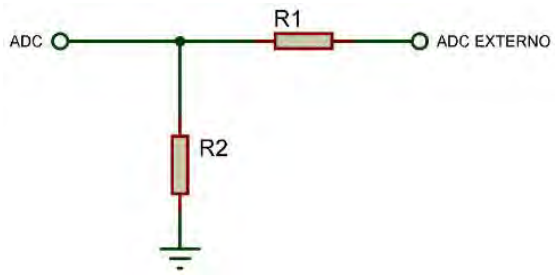


Figura P30.1 *Divisor de voltaje.*

ADC, por lo que el rango de voltaje de entrada puede variar.

Los SoC ESP32 integran dos convertidores analógico a digital de aproximaciones sucesivas de hasta 12 bits de precisión. En este caso, sólo se hace uso de uno de ellos, el ADC1 por lo que se tiene acceso a 8 canales correspondientes a los GPIO 32-39. La escala predeterminada completa de voltaje de ADC es 1.1 *V*. Para leer voltajes más altos (hasta el voltaje máximo del pin, generalmente 3.3 *V*) se requiere configurar la atenuación (> 0 dB) de señal para ese canal de ADC [5] (ver tabla P30.1) donde el máximo voltaje es limitado por VDD_A.

Tabla P30.1 *Voltajes ADC*

Atenuación VDD_A=3.3V		**Rangos recomendados para mayor precisión**
0 *dB*	1.1 *V*	100 y 950 *mV*
2.5 *dB*	1.5 *V*	100 y 1250 *mV*
6 *dB*	2.2 *V*	150 y 1750 *mV*
11 *dB*	3.9 *V*	150 y 2450 *mV*

La biblioteca ESP permite realizar las siguientes acciones:

- Configurar los canales ADC en ESP-IDF
- Realizar una lectura ADC

Objetivo

Mostrar cómo utilizar el ADC incorporado en el módulo WiFi.

***Firmware* de comandos AT**

El módulo WiFi empleado debe tener el *firmware* de comandos AT basado en el SDK NONOS 3.0.3 o ESP-IDF, por lo que se puede emplear cualquiera de los siguientes módulos:

- ESP-07S, para lo cual únicamente SDK_V3 debe ser TRUE en esp.h.
- ESP32-DevKitC, para lo cual únicamente IDF_ESP32 (o IDF_ESP8266 para el ESP8266) debe ser TRUE en esp.h.

En esta práctica se empleará el módulo ESP-07S y únicamente SDK_V3 debe ser TRUE en esp.h.

Habilitación de aplicaciones

STATION o ACCESS_POINT y E_ADC

Material

- 1 potenciómetro de 10kΩ
- 1 resistor de 510Ω
- 1 resistor de 220Ω
- 3 resistores de 10kΩ
- 1 microcontrolador PIC
- 1 módulo WiFi
- 1 adaptador serial USB-TTL

Descripción

Se realizarán lecturas analógicas de un potenciómetro cada segundo, utilizando el ADC que tiene incorporado el módulo WiFi. Los valores obtenidos serán impresos en una terminal.

Código

El código de esta práctica se muestra en la figura P30.3, mientas que en la figura P30.4 se muestra un ejemplo del código para IDF.

Circuito

El circuito de esta práctica se muestra en la figura P30.5.

Para determinar los valores de los resistores del divisor de voltaje se utilizó la siguiente ecuación:

$$\mathrm{V_{out}} = \mathrm{V_{in}} * \frac{\mathrm{R_2}}{\mathrm{R_1} + \mathrm{R_2}}$$

considerando que Vin es 3.3 V y que Vout es 1 V. Para R2 se propuso el valor de 220 Ω por lo que para determinar R1 se tiene:

$$R_1 = \frac{V_{in} * R_2}{V_{out}} - R_2 = \frac{3.3 * 220}{1} - 220 = 506\Omega$$

Resultados

Al iniciar la ejecución del programa del microcontrolador PIC, en la terminal comenzarán a aparecer mensajes cada segundo con el valor obtenido del potenciómetro (ver figura P30.2). Se variará el valor del potenciómetro y se deberá observar cómo cambian los valores que son impresos en la terminal.

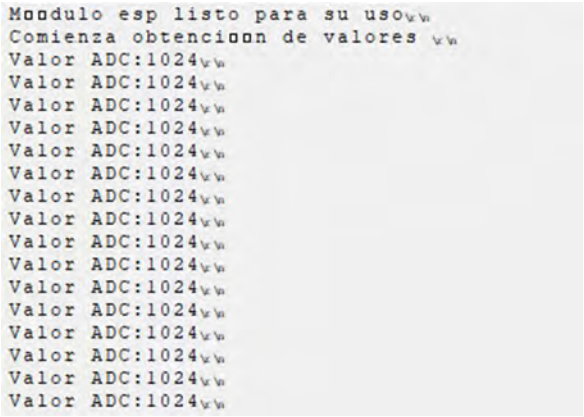


Figura P30.2 *Impresión de valores obtenidos.*

En el siguiente enlace está la carpeta que contiene el código y el video del funcionamiento de la práctica:

https://bit.ly/3diZirp

Notas

- Las conexiones del adaptador USB_TTL están basadas en los pines por defecto del segundo serial por software creado para el microcontrolador PIC. Estos pines pueden ser cambiados en el archivo de cabecera esp.h
- Se recomienda utilizar el programa HTerm para simular la terminal.
- IDF es exclusivo para ESP32 en esta aplicación.

Práctica propuesta

Realizar lecturas analógicas de un potenciómetro utilizando el ADC que tiene incorporado el módulo WiFi. Estos valores deben ser utilizados para cambiar la intensidad de luz de un LED.

```c
#include <16lf1939.h>
#include "esp.h"

uint16 valor_adc=0;
uint8 err=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    up_to_level2();
    fprintf(PIC,"Comienza obtención de valores \r\n");
    while(1)
    {
        // Realiza lectura ADC
        err=read_mod_adc(&valor_adc);
        //¿Se realizó correctamente la lectura ADC?
        if(err==0)
        {
            fprintf(PIC,"Valor ADC:%lu\r\n",valor_adc);
            delay_ms(500);
        }
    }
}
```

Figura P30.3 *Código de la práctica 30 para SDK NONOS.*

```c
#include <16lf1939.h>
#include "esp.h"

uint16 valor_adc=0;
uint8 err=0;
void main()
{
    //Inicialización del módulo Wi-Fi
    begin_esp();
    up_to_level2();
    // Se configura el canal a utilizar y el nivel de ate
    config_channel_adc(0,0);
    fprintf(PIC,"Comienza obtención de valores \r\n");
    while(1)
    {
        // Realiza lectura ADC
        err=read_mod_adc(0,&valor_adc);
        //¿Se realizó correctamente la lectura ADC?
        if(err==0)
        {
            fprintf(PIC,"Valor ADC:%lu\r\n",valor_adc);
            delay_ms(500);
        }
    }
}
```

Figura P30.4 *Ejemplo de código de la práctica 30 para IDF.*

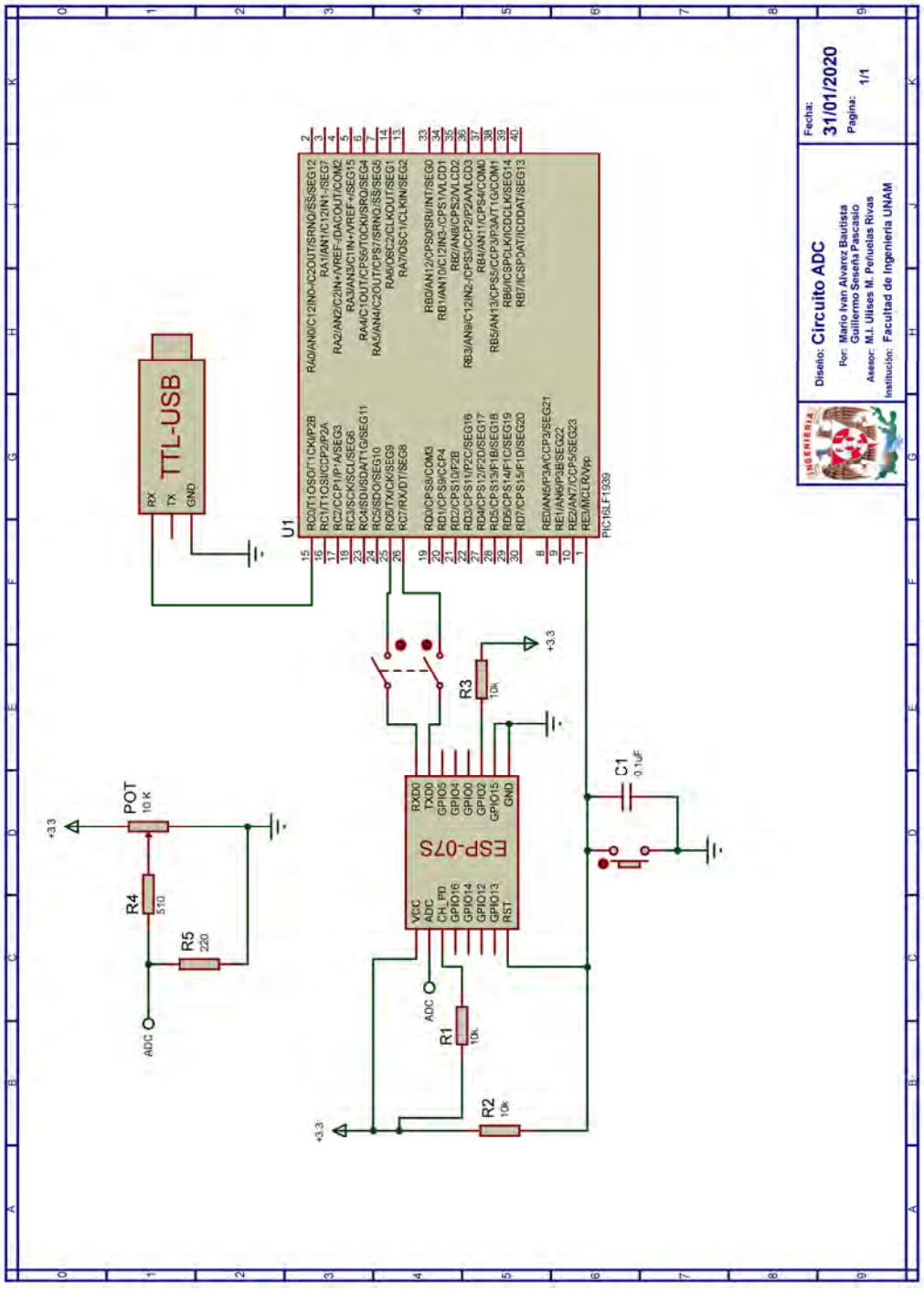


Figura P30.1 *Circuito de la práctica 30*

Referencias

1. Bolton, W., *Mecatrónica: sistemas de control electronico en la ingenieria mecánica y eléctrica*. 5 ed. 2013: Alfaomega.
2. Espressif Systems, *ESP8266 AT Instruction Set Version 3.0.2*. 2019.
3. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.
4. AI-Thinker Inc, *ESP-01/07/12 Series Modules User's Manual Version 1.1*. 2017.
5. Espressif Systems. *Get Started*. ESP-IDF Programming Guide [Citado 2020 Marzo]; Disponible en:https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html.

Capítulo 6

Conclusiones y trabajo a futuro

6.1 Conclusiones

El presente trabajo consistió en el desarrollo de una biblioteca que permitiera emplear un microcontrolador PIC junto con un módulo WiFi como una opción, caracterizada por su flexibilidad respecto a *hardware,* para el desarrollo de prototipos mecatrónicos que se adapten a las principales características que imponen las nuevas tendencias, principalmente el IoT. La elección de la tecnología WiFi se debió a su impacto socioeconómico y su participación en las etapas tempranas del *internet of Things*.

La biblioteca ESP creada en este trabajo satisface el punto anterior, ya que permite que un microcontrolador PIC, pueda acceder a una red para comunicarse de manera inalámbrica con otros dispositivos y así satisfacer necesidades de movilidad o colocación en zonas de difícil acceso. Además, brinda la capacidad de conectarse a internet para acceder a servicios web o consultar, almacenar y/o procesar información con plataformas IoT. Todo esto lo realiza utilizando los protocolos más comunes en internet que fueron parcialmente desarrollados en la biblioteca.

Con las modificaciones realizadas en los *firmware* de comandos AT, se incrementó el número de aplicaciones a las que podría acceder con biblioteca ESP, sin embargo, esto implica que el usuario debe actualizar el *firmware* del módulo por alguna de las versiones modificadas antes de utilizarla por primera vez. La elección de la versión del *firmware* está basada principalmente en el SoC, el módulo y el tamaño de memoria de éste. Además, impacta en el número de aplicaciones a las que el usuario tendrá acceso. Sin embargo, el

impacto más significativo es la cantidad de los periféricos del módulo a los que se tendría acceso, que depende del *hardware* del módulo seleccionado.

Aunque en este trabajo se proporcionan los archivos para realizar la actualización del *firmware* para ciertos tamaños de memoria del módulo, si el usuario cuenta con uno cuya memoria es de tamaño diferente, se verá obligado a realizar la compilación de los archivos del *firmware*. Creemos que, lo anterior podría desmotivar el uso de esta alternativa a usuarios con poca experiencia. Por esto, en este trabajo se brinda la documentación para realizar los procedimientos para apoyar a los desarrolladores. Así mismo, se brinda la información que permitirá agregar nuevas aplicaciones a la biblioteca, logrando así que la biblioteca se mantenga actualizada y se aprovechen los nuevos desarrollos en el *firmware* del módulo.

El diseño de la biblioteca ESP como un módulo de compilación separada contrasta con las colecciones de objetos *"precompilados"* que comúnmente se ofrecen en otras. Además, derivado de la estructura de ésta, el usuario debe modificar el archivo fuente y el archivo de cabecera para seleccionar un microcontrolador PIC, configurar los periféricos que utiliza la biblioteca y configurar el funcionamiento de ésta, lo cual es poco usual. Otro aspecto relevante es que la biblioteca fue diseñada como un solo módulo para aprovechar mejor los recursos PIC, que permite emplear aplicaciones simultáneas en un solo proyecto. Estas aplicaciones son habilitadas desde el archivo de cabecera, que difiere de otras bibliotecas que por cada aplicación dispone de un archivo de cabecera que sirve para incluir los respectivos objetos al proyecto. Finalmente, derivado del compilador para el cual se diseñó la biblioteca ESP, la portabilidad de ésta a otros compiladores se ve comprometida.

El manual de prácticas desarrollado para la biblioteca ESP aunado a la documentación de cada función, facilita su utilización e introduce a los usuarios a los principales conceptos que se manejan en redes e internet. El manual puede ser empleado como material didáctico para la asignatura Circuitos Digitales como educación basada en el desarrollo de proyectos, reduciendo significativamente el tiempo de aprendizaje que un estudiante tendría que invertir para el desarrollo de un prototipo que involucre estos conceptos.

Partiendo del uso de un microcontrolador, el conjunto propuesto servirá para el desarrollo de aplicaciones simples, es decir, que ejecuten principalmente una sola tarea; sin embargo, con la posibilidad de comunicarse con plataformas IoT, con lo que es posible que el uC PIC sea capaz de generar grandes cantidades de información que serán enviadas a éstas para su almacenamiento y análisis, acciones que resultaban casi imposibles debido a las limitaciones del microcontrolador.

El hecho de utilizar un uC PIC como maestro de un dispositivo de mayor capacidad computacional como lo es un módulo Wi-Fi puede parecer incrongruente; sin embargo, la demanda de recursos y nivel de procesamiento que requiere la admistración de las tareas de red, hace que la arquitectura distribuida utilizada en este trabajo, en el que el módulo se encarga únicamente de tareas de comunicación, resulte adecuado para algunas aplicaciones en donde la interrupción prolongada de tareas no es deseada. No obstante, la lógica empleada en la biblioteca reduce la posibilidad de ejecutar tareas completamente simultáneas, esta característica es mejor aprovechada cuando se diseñan funciones para acciones particulares desde el *firmware* del módulo, tal como se ilustra en RSSI y *tracking,* en donde el módulo únicamente notifica al uC PIC del resultado, permitiendo que este último realice otras acciones durante la ejecución de las tareas del módulo.

Si bien la biblioteca ESP brinda flexibilidad de *hardware* para utilización tanto de diferentes microcontroladores como de módulos WiFi y, por tanto, es más probable que se pueda seleccionar un conjunto que se adapte a las necesidades del prototipo que se quiera desarrollar y que puedan ser reutilizados; se requiere que éstos presenten ciertas características para que puedan trabajar juntos, las cuales no podrán ser utilizadas por el usuario. En el caso del microcontrolador PIC, se concluyó que para usar la biblioteca ESP y pueda ser maestro de un módulo WIFi, éste debe contar con un módulo EUSART con interrupción por la recepción de datos RS232 (interrupción RDA), un temporizador de 8 o 16 bits con interrupción por desbordamiento y un tamaño, al menos, de 6 registros de pila. Las características de memoria dependerán de las del prototipo. En el caso del módulo WiFi, éste deberá incorporar un SoC ESP8266 o algún SoC ESP32, ser compatible con alguna de las versiones del *firmware* de comandos AT SDK NONOS o ESP-IDF que proporciona Espressif Systems y contar con una memoria flash de al menos 8 Mbits.

6.2 Trabajo a futuro

En este trabajo no se desarrollaron todas las aplicaciones que son posibles con las herramientas y APIs del módulo, ni tampoco algunas de las modificaciones realizadas en el *firmware* de comandos AT basado en el SDK NONOS, se desarrollaron en la versión ESP-IDF. Tomando en cuenta lo anterior, en este trabajo se describió la estructura que permitiría hacer dichas modificaciones y agregar nuevas aplicaciones a la biblioteca ESP, logrando así que ésta pueda seguir siendo desarrollada por terceros y, eventualmente, no exista una diferencia más allá de la originada por las características propias de cada versión del *firmware*. Estas modificaciones se pueden realizar en dos formas, el primero es creando nuevas aplicaciones a partir de las herramientas con las que ya se cuenta, enfocando el desarrollo del código hacia el uC PIC, mientras que la segunda forma

consiste en desarrollar código tanto en el uC PIC como en el módulo WiFi, ésto último mediante la creación de nuevos comandos AT y aprovechando la actualizaciones y lanzamientos de nuevas APIs en el *firmware* de *Espressif Systems*.

Aquí se mencionan algunos de los múltiples caminos en los que esta biblioteca puede ser enriquecida:

- El primero de ellos corresponde a la interfaz de comunicación utilizada para que el módulo WiFi y el microcontrolador PIC se comuniquen. El *firmware* de comandos AT permite la utilización de otras interfaces, como lo es I2C, por lo cual se podrían realizar las modificaciones correspondientes tanto en los archivos del *firmware* como en los de la biblioteca para habilitar al usuario la opción de seleccionar la interfaz de comunicación que éste desee.

- Considerando la forma en que se añaden las modificaciones al *firmware* de comandos AT y que los módulos también presentan una limitación respecto a la memoria, se puede crear *firmware* personalizado, es decir, que contengan únicamente las aplicaciones que el usuario necesite y aprovechar la memoria no utilizada para almacenar información proveniente de otros dispositivos, reteniéndola en el módulo y notificando al uC de ésta.

- Se pueden crear más comandos AT, especialmente para la versión ESP-IDF, para aprovechar todas las características que el SoC ESP8266 y los ESP32 ofrecen. Esto implica también desarrollar código en el microcontrolador PIC.

- Desarrollar en la biblioteca ESP funciones que utilicen los comandos AT ya existentes para el uso de *bluetooth*, tecnología que ya intengran los ESP32. Esto permitiría al conjunto propuesto comunicarse con otro tipo de dispositivos.

- Solicitar a Espressif Systems los archivos del firmware de comandos AT para modificar el nombre de éstos y, por tanto, reducir la cantidad de memoria que se requiere para su almacenamiento, lo cual podría abrir la posibilidad de que más microcontroladores PIC sean compatibles.

- Desarrollar funciones que permitan acceder a los modos de ahorro de energía del módulo (*sleep mode*) para periodos sin actividad con el objetivo de aumentar el tiempo de funcionamiento de la batería que alimente al conjunto.
- Los protocolos de aplicación parciamente adaptados en la biblioteca ESP pueden ser actualizados a futuras versiones o incluir más especificaciones de éstos.
- Implementar los protocolos de aplicación parcialmente adaptados en los firmwares NONOS e IDF para liberar recursos del uC PIC.

- Desarrollar más aplicaciones a partir de las herramientas que ya se proporcionan, entre las que podrían destacar:

    - Uso de una conexión SSL en *servidor upgrade* (de actualización).
    - Transmisión de audio o imágenes mediante paquetes UDP.
    - Ejecutar la aplicación *Servidor upgrade* (de actualización) a través de internet y ya no de manera remota. Para esto se tendría que conectár a un AP con acceso a internet, en vez de crear uno como actualmente lo hace, lo cual podría requerir un mapeo de puertos en el *router* involucrado.

Referencias

1. Tanenbaum, A.S. and D. Wetherall, *Redes de computadoras*. Quinta ed. 2012: Pearson.
2. Forouzan, B.A., *Data Communications and Networking*. 4th Edition ed. 2007: McGraw-Hill.
3. Molisch, A.F., *Wireless communications*. 2011: John Wiley & Sons.
4. Goldsmith, A., *Wireless communications*. 2005: Cambridge University Press.
5. Salazar, J., *Redes inalámbricas.* 2016.
6. Cisco Visual Networking Index, *Global Mobile Data Traffic Forecast Update, 2016–2021* W. Paper, Editor. 2017.
7. Internet Society, *La internet de las cosas. Una breve reseña.* Internet Society, 2015.
8. Sisini, E., et al., *Industrial Internet of Things: Challenges, Opportunities, and Directions.* IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, 2018. **10**(10).
9. Autorité de Régulation des Communications Électroniques et des Postes, et al., *Preparing for the Internet of Things Revolution: Mapping out the challenges.* 2016(1).
10. Jeschke, S., et al., *Industrial Internet of Things. Cybermanufacturing Systems*. Wireless Technology. 2017: Springer.
11. Anitha Varghese and D. Tandur *Wireless requirements and challenges in Industry 4.0*. 2014.
12. Wollschlaeger M, Sauter T, and Jasperneite J, *The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0.* IEEE Industrial Electronics Magazine, 2017. **11**(1).
13. Frotzscher, A., et al., *Requirements and current solutions of wireless communication in industrial automation*. 2014, 2014 IEEE International Conference on Communications Workshops (ICC): Sydney, NSW, Australia.
14. Bolton, W., *Mecatrónica: sistemas de control electronico en la ingenieria mecánica y eléctrica*. 5 ed. 2013: Alfaomega.
15. Alciatore, D.G. and M.B. Histand, *Introduction to Mechatronics and Measurement Systems*. 4ta ed. 2012: McGraw-Hill.

16. Kuru, K. and H. Yetgin, *Transformation to Advanced Mechatronics Systems Within New Industrial Revolution: A Novel Framework in Automation of Everything (AoE).* IEEE Access, 2019. **7**.
17. Rojas, J.H.C. *La Cuarta Revolución Industrial o Industria 4.0 y su Impacto en la Educación Superior en Ingeniería en Latinoamérica y el Caribe*. International Multi-Conference for Engineering, Education, and Technology, 2017.
18. Bradley, D., et al., *The Internet of Things – The future or the end of mechatronics.* Elsevier Ltd, 2015.
19. Bright, G., N.S. Tlale, and M.C. Kumile, *Wireless Communication Technology for Modular Mechatronic Controllers* in *Mechatronics in Action: Case Studies in Mechatronics Applications and Education*. 2010, Springer.
20. Rahul, A., et al., *Near Field Communication (NFC) technology: A survey.* International Journal on Cybernetics & Informatics, 2015. **4**.
21. Lethaby, N., *Wireless connectivity for the Internet of Things: One size does not fit all.* Texas Instruments, 2017.
22. Olsson, J., *6LoWPAN demystified.* Texas Instruments, 2014.
23. Girubanandhini, M. and R. Kalaiprasath, *Wireless Zigbee network- Capacity calculation and secure data conveyance using indegree.* International Journal of Current Advanced Research, 2016. **5**(3).
24. Fuller, J.D. and B.W. Ramsey, *Rogue Z-wave controllers: A persistent attack channel* IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops), 2015.
25. International Telecommunication Union, *Short range narrow-band digital radiocommunication transceivers – PHY and MAC layer specifications* ITU Publications, 2012. **9959**.
26. Sigfox. *Welcome to Sigfox Build*. [Citado 2019 Diciembre]; Disponible en: https://build.sigfox.com/steps/sigfox.
27. Prieto, J., *Tecnología y desarrollo en dispositivos móviles: Introducción a los sistemas de comunicación inalámbricos.* 2011.
28. Wi-Fi Alliance, *Next generation Wi-Fi®: The future of connectivity.* Whitepaper, 2018.
29. Khan, J. and A. Khwaja, *Building Secure Wireless Networks with 802.11*. 2003, Indianapolis, Indiana: John Wiley & Sons.
30. Coleman, D.D. and D.D. Westcott, *CWNA® Certified Wireless Network Administrator Study Guide Exam CWNA-107*. Quinta ed. 2018, Indianapolis, Indiana: John Wiley & Sons.
31. Wi-Fi Alliance. *News & Events*. [Citado 2019 Diciembre]; Disponible en:https://www.wi-fi.org/news-events.
32. Oullet, E., et al., *Building a Cisco Wireless LAN*. 2002: Syngress Publishing Inc.
33. Interbrand. *Growing global recognition takes the right name Wi-Fi*. [Citado 2019 Agosto]; Disponible en:https://www.interbrand.com/work/catchier-than-ieee-802-11b-welcome-wi-fi/.
34. Wi-Fi Alliance, *Infographic 20 years of Wi-Fi*. 2019.
35. Wi-Fi Alliance. *Certification*. [Citado 2019 Diciembre]; Disponible en:https://www.wi-fi.org/certification.

36. Martinez, B., C. Cano, and X. Vilajosana, *A Square Peg in a Round Hole: The Complex Path for Wireless in the Manufacturing Industry*, in *IEEE Communications Magazine*. 2019. p. 109-115.
37. Candell, R. and M. Kashef, *Industrial Wireless: Problem Space, Success Considerations, Technologies, and Future Direction.* Resilience Week (RWS), 2017.
38. Wireless Broadband Alliance. *World Wi-Fi Day*. [Citado 2019 Diciembre]; Disponible en:https://wballiance.com/events-awards-3/world-wi-fi-day/.
39. Gobierno de la ciudad de México. *Agencia Digital de Innovación Pública* Centros, Conectividad e Infraestructura Tecnológica [Citado 2019 Diciembre]; Disponible en:https://adip.cdmx.gob.mx/centros/conectividad-e-infraestructura-tecnologica.
40. AT&T. *AT&T amplía conectividad y Wi-Fi gratuito en las líneas 8 y 9 del Metro de la CDMX*. 2019 [Citado 2019 Diciembre]; Disponible en:https://www.att.com.mx/newsroom/noticia/att-amplia-conectividad-wifi-gratuito-lineas-8-y-9-metro-cdmx.
41. Wireless Broadband Alliance. *Facts and Stats*. [Citado 2019 Diciembre]; Disponible en:https://worldwifiday.com/about-us/facts/.
42. AbdulAziz, H., N.N.M.K. Yusoff, and M.Z.B.M. Sapien, *MINI 11-Microcontroller Development Board for SCL Approach*, in *IEEE Student Conference on Research and Development (SCOReD)*. 2010: Putrajaya, Malaysia.
43. Fidai, A., et al., *Internet of Things (IoT) Instructional Devices in STEM Classrooms: Past, Present and Future Directions.* IEEE Frontiers in Education Conference (FIE), 2019.
44. Artero, Ó.T., *Arduino: Curso práctico de fomación*. 2013, España: RC Libros.
45. Arduino. *Arduino Products*. [Citado 2019 Diciembre]; Disponible en:https://www.arduino.cc/en/Main/Products.
46. Ibrahim, D., *Applying the ESP8266 processor in IoT applications*, in *Electronics World*. 2017. p. 14-17.
47. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2019.
48. Ceja, J., et al., *Módulo ESP8266 y sus aplicaciones en el internet de las cosas* in *Revista de Ingeniería Eléctrica*. 2017. p. 24-36.
49. AI-Thinker team *ESP-01 WiFi Module version 1.0*. 2015.
50. SparkFun Electronics. *SparkFun Electronics*. [Citado 2019 Diciembre]; Disponible en:https://www.sparkfun.com/.
51. Espressif Systems. *Modules*. 2019 [Citado 2019 Diciembre]; Disponible en:https://www.espressif.com/en/products/hardware/modules.
52. Wilderness Labs. *Which Netduino is right for your project?* [Citado 2019 Diciembre]; Disponible en:http://www.wildernesslabs.co/Netduino.
53. 330ohms. *Página de compra*. [Citado 2019 Diciembre]; Disponible en:https://www.330ohms.com/.
54. Adafruit. *Adafruit Shop*. [Citado 2019 Diciembre]; Disponible en:https://www.adafruit.com/.
55. Onion Corporation. *OMEGA2 +*. [Citado 2019 Diciembre]; Disponible en:https://onion.io/store/omega2p/.

56. UDOO. *UDOO NEO EXTENDED*. [Citado 2019 Diciembre]; Disponible en:https://shop.udoo.org/other/udoo-neo-extended1.html?___from_store=other&popup=no.
57. Espressif Systems. *ESP32-DevKitC V4 Getting Started Guide*. Get Started [Citado 2019 Diciembre]; Disponible en:https://docs.espressif.com/projects/esp-idf/en/latest/hw-reference/get-started-devkitc.html.
58. MEDIATEK labs. *LinkIt™ ONE*. [Citado 2019 Diciembre]; Disponible en:https://labs.mediatek.com/en/platform/linkit-one.
59. Espresssif Systems. *Development Boards*. [Citado 2019 Diciembre]; Disponible en:https://www.espressif.com/products/hardware/development-boards.
60. Ciuffoletti, A., *Design and implementation of a low cost modular sensor.* Second International Workshop on Smart Environments and Urban Networking, 2017.
61. Wilmshurst, T., *Designing Embedded Systems with PIC Microcontrollers Principles and applications*. 2007: Elsevier Ltd.
62. Mandado Pérez, E., *Microcontroladores PIC: Sistema Integrado para el Autoaprendizaje.* 2007.
63. Espressif Systems. *SoC ESP*. [Citado 2020 Enero]; Disponible en:https://www.espressif.com/en/products/hardware/socs.
64. Espressif IOT Team, *Datasheet ESP8266EX*, Espressif Systems, Editor. 2018.
65. Espresssif Systems, *ESP32 Series DataSheet Version 3.2*. 2019.
66. Espressif Systems, *ESP8266 SDK Getting Started Guide Version 3.2*. 2019.
67. Osorio, D.M.L. and A.d.R. Delgado, *DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMUNICACIÓN INALÁMBRICO PARA AUTOMATIZACIÓN Y MONITOREO*, in *Facultad de Ingeniería*. 2016, Universidad Nacional Autónoma de México: Ciudad Universitaria.
68. Espressif Systems, *ESP8266 Non-OS SDK API Reference Version 3.0.1*. 2019.
69. Espressif Systems. *SDK*. Resources [Citado 2020 Enero]; Disponible en:https://www.espressif.com/en/support/download/sdk.
70. Espressif Systems. *ESP-IDF Programming Guide*. Get Started,Build System and About [Citado 2020 Enero]; Disponible en:https://docs.espressif.com/projects/esp-idf/en/release-v3.0/index.html.
71. Espresssif Systems, *ESP8266 IDF AT Release Note*. 2019.
72. Espressif Systems. *esptool.py*. esptool [Citado 2020 Enero]; Disponible en:https://github.com/espressif/esptool.
73. AI-Thinker Inc, *ESP-01/07/12 Series Modules User's Manual Version 1.1*. 2017.
74. wujiangang. *SDK NONOS Releases*. 2018 [Citado 2020 Febrero]; Disponible en:https://github.com/espressif/ESP8266_NONOS_SDK/releases/.
75. Microchip Technology Inc. *Libraries*. MPLAB X IDE Microchip Developer Help [Citado 2020 Febrero]; Disponible en:http://microchipdeveloper.com/mplabx:libraries.
76. Pitts, R.I. *Modules, Separate Compilation, Make Files*. [Citado 2020 Febrero]; Disponible en:https://www.cs.bu.edu/teaching/c/separate-compilation/.
77. AI-Thinker, *ESP-01 802.11 b/g/n Wi-Fi Module V1.2*. 2017, Ai-Thinker Technology Co.

78. SparkFun Electronics. *Página de compra*. [Citado 2020 Marzo]; Disponible en:https://www.sparkfun.com/.
79. AI-Thinker, *ESP-07S DataSheet V1*. 2018.
80. Microchip Technology Inc, *PIC16(L)F1938/9 DataSheet*. 2011-2017.
81. Microchip Technology Inc. *PIC16F1939*. [Citado 2020 Marzo]; Disponible en:https://www.microchip.com/wwwproducts/en/PIC16F1939.

Apéndice A

Actualización del *firmware* del módulo WiFi

A.1 Herramientas

El programa utilizado para actualizar el *firmware* de los módulos que incorporen el SoC ESP8266 o el SoC ESP32 es *ESP Flash Download Tool,* el cual es desarrollado por de Espressif Systems para computadoras con sistema operativo Windows y puede ser descargado de manera gratuita, dentro de la sección *Flash Download Tools (ESP8266 & ESP32),* en:

http://espressif.com/en/support/download/other-tools

Este programa se encuentra dentro de una carpeta comprimida con extensión .zip, la cual contiene todos los archivos del programa para que éste trabaje de manera correcta. Una vez realizada la descarga y la descompresión de los archivos, basta con hacer doble clic sobre el archivo mostrado en la figura A.1 para iniciar el programa. En este caso, se usa la versión 3.6.7 del programa.

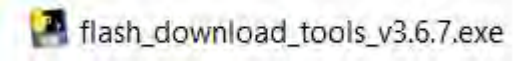


Figura A.1 *Ejecutable de programa de actualización.*

Así mismo, es necesario contar con un adaptador serial USB-TTL para comunicar al módulo y a la computadora donde se ejecuta el programa antes mencionado. Existen varios adaptadores USB-TTL que pueden ser utilizados, únicamente se tiene que tener en cuenta, no utilizar adaptadores de 5 *volts* o adaptadores RS-232 estándar [1]. En la figura A.2 se muestra un ejemplo de un convertidor con el que se puede actualizar el *firmware*.

A.2 Hardware

Para realizar la actualización del *firmware,* los módulos deben ser reiniciados al establecer ciertos niveles de voltaje en algunos de sus GPIOS durante el encendido del módulo, al realizar lo anterior el módulo entra en modo denominado *serial bootloader* [1]. A continuación, en la figura A.3 y figura A.4 se muestran los circuitos para este fin.

Figura A.2 *Ejemplo de adaptador serial USB-TTL.*

A.3 Software

Para realizar la actualización del *firmware* de un módulo WiFi se requieren los archivos binarios que se desea grabar, que se encuentran en el enlace:

https://bit.ly/2BsLtJt

Los archivos binarios del *firmware* modificado de las versiones SDK NONOS (en la carpeta *bin*) y ESP-IDF (en la carpeta *build*) generados para módulos con las siguientes características:

- ESP-IDF ESP32, configuración por defecto para un ESP32 de 4 *MBytes* de memoria, velocidad SPI 40 *MHz*, modo SPI DIO.
- ESP-IDF, configuración por defecto para un ESP8266 de 4, 8 y 16 *MBytes* de memoria, velocidad SPI 40 MHz, modo SPI DIO.
- SDK NONOS v3.0.3, para un módulo ESP8266 de 16 y 32 *Mbits* de memoria con un mapa de 1024 *KB* + 1024 *KB*, velocidad SPI 40 MHz, modo SPI QIO.

- SDK NONOS v2.1.0, para un módulo ESP32 de 8 ,16, y 32 *Mbits* de memoria con un mapa de 512 *KB* + 512 *KB*, velocidad SPI 40 MHz, modo SPI QIO.

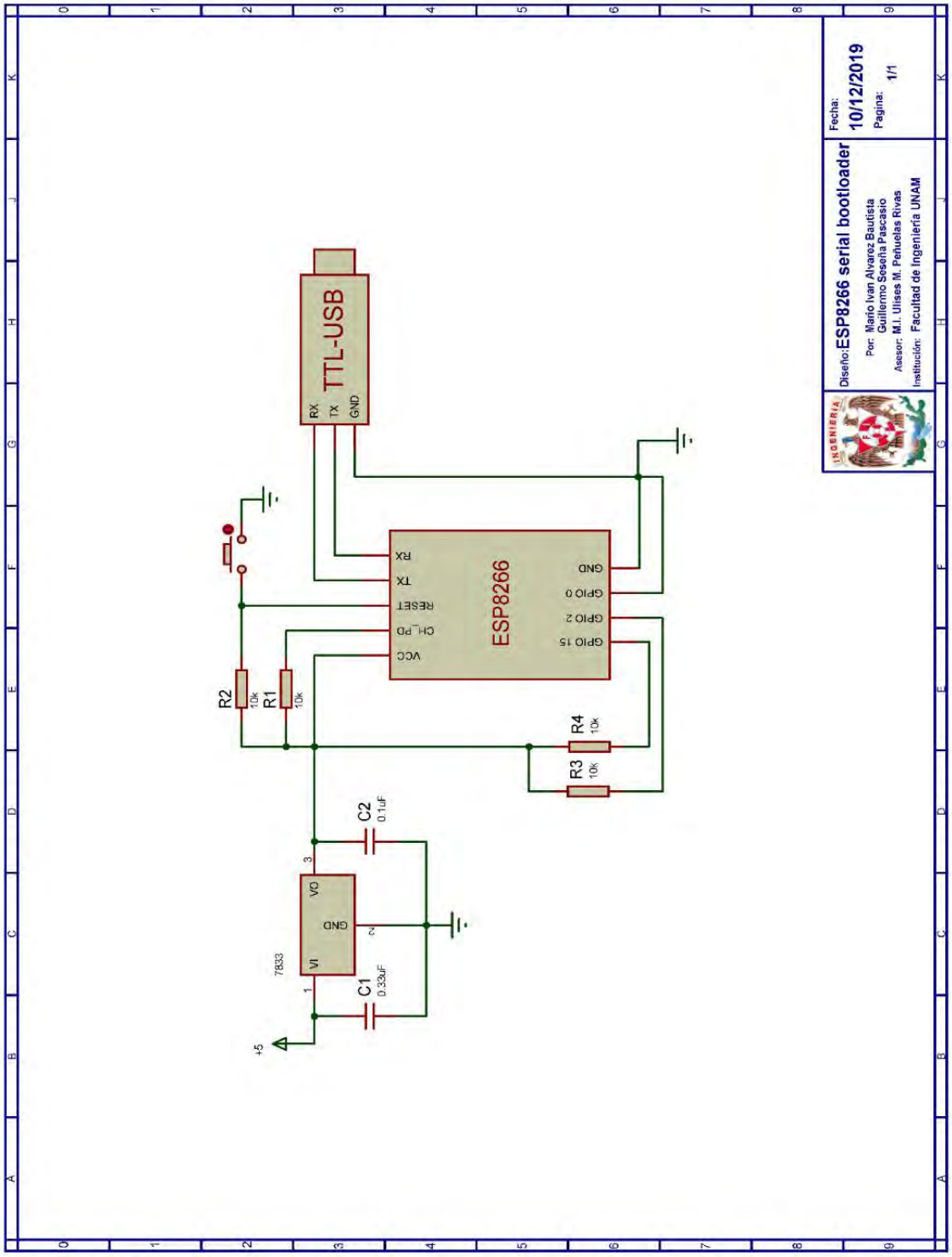


Figura A.3 *Circuito de actualización de firmware ESP8266 [1].*

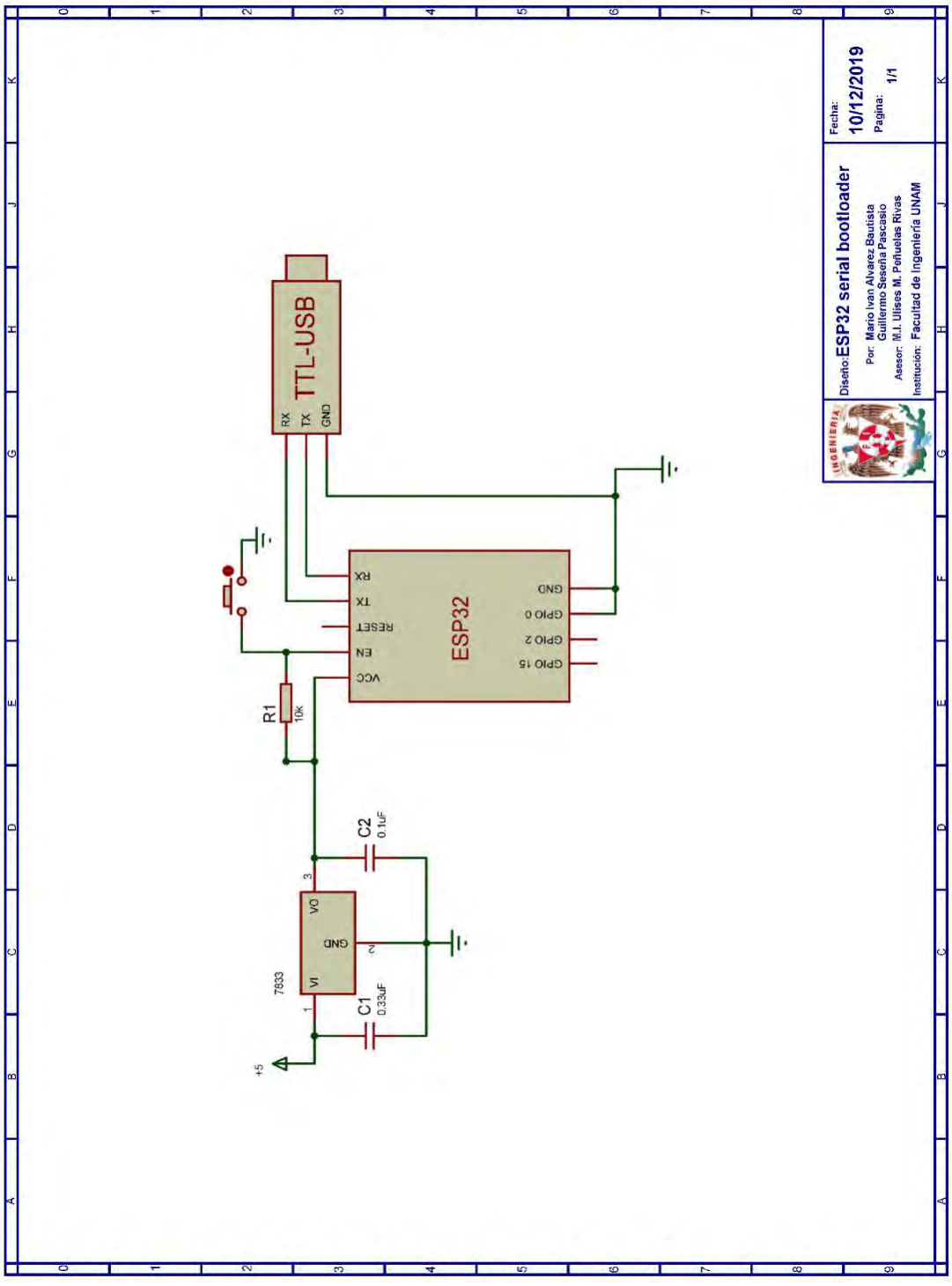


Figura A.4 *Circuito de actualización de firmware ESP32 [1].*

Para módulos con configuraciones diferentes, se recomienda seguir las instrucciones que se especifican en el Apéndice B para generar los archivos binarios que incluyan las modificaciones realizadas en este trabajo. A continuación, se explica el procedimiento a seguir para realizar la actualización.

A.3.1 Selección de SoC

Al ejecutar el programa de la figura A.1 se mostrará inmediatamente la interfaz de la figura A.5, en la cual se observará un menú de las diferentes herramientas para realizar la actualización de los SoC WiFi. En este caso en particular, sólo se ocupan las herramientas *ESP8266 Download Tool* y *ESP32 Download Tool.* Al hacer clic en uno de los recuadros que contienen los nombres de éstas, se mostrará una de las interfaces de la figura A.7 y figura A.6. Estas interfaces corresponden a la pestaña *SPIDownload* que será sobre la cual se trabajará. Tal y como se observa en la figura A.7 y la figura A.6, estas interfaces se diferencian únicamente por las configuraciones establecidas por defecto y por los valores disponibles en la sección *FLASH SIZE.*

Es por esto y porque el procedimiento para ambos SoC es el mismo, que se utilizará de aquí en adelante la interfaz de la figura A.6 para ilustrar el proceso de actualización del *firmware.*

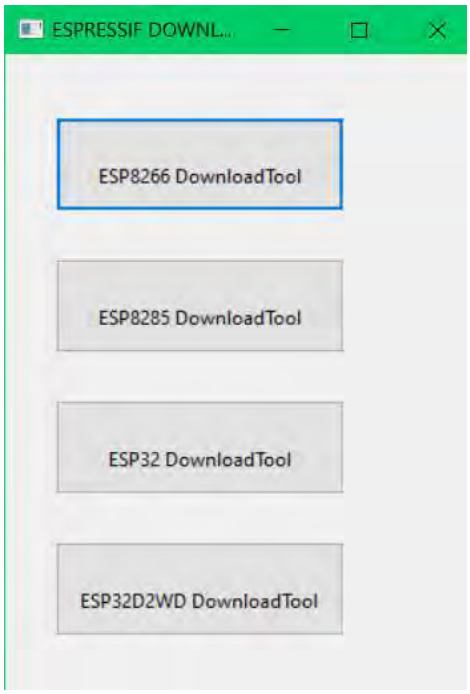


Figura A.5 *Interfaz de herramientas de actualización.*

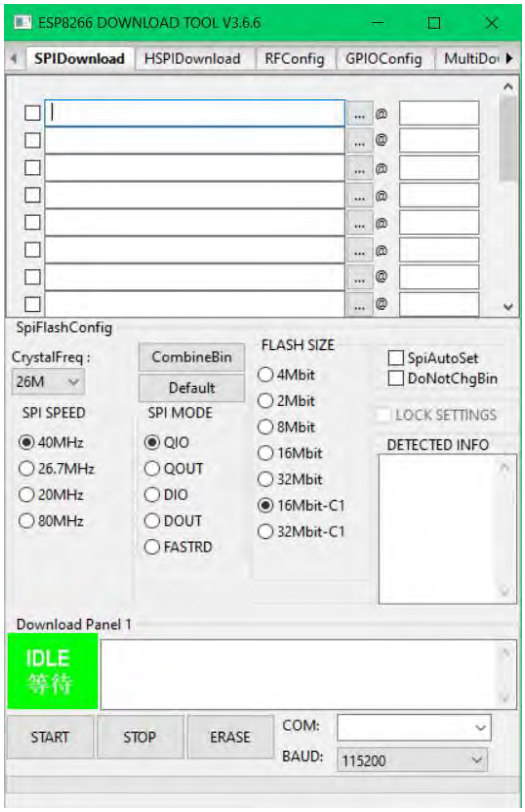


Figura A.6 *Interfaz de actualización ESP8266*

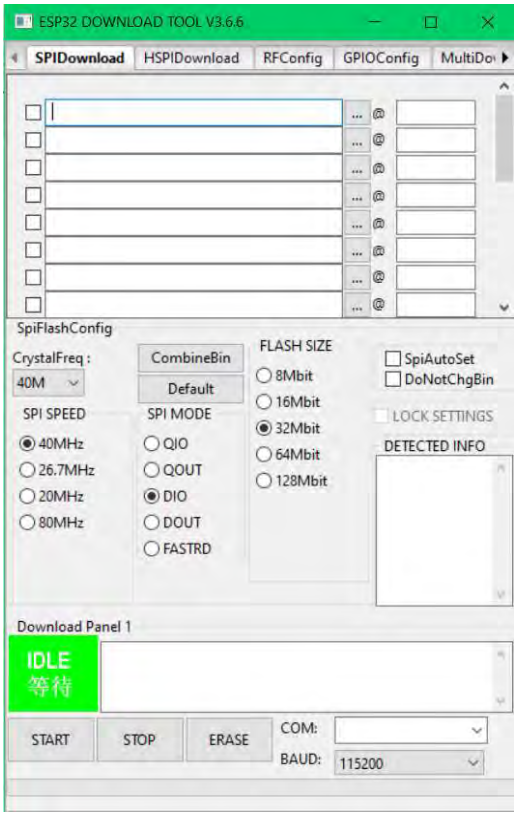


Figura A.7 *Interfaz de actualización ESP32*

A.3.2 Descripción de la interfaz

En la figura A.8 *Partes de la interfaz de actualización* se pueden observar las diferentes partes que componen a la interfaz de actualización. La zona marcada con el número 1 se conoce como el panel de configuración de rutas de descarga (*Download Path Config)*, éste es el panel donde se establecen las direcciones de memoria y se seleccionan los archivos binarios que serán cargados en la memoria flash del módulo WiFi. La zona marcada con el número 2, se conoce como panel de configuración SPI (*SPI Flash Config*), en el cual se establecen los parámetros para realizar la actualización (ver tabla A.1). La zona marcada con el número 3 se conoce como panel de descarga (*Download Panel*), en el cual están los controles para iniciar o detener la actualización, así como también muestra el estado actual de la actualización (ver tabla A.2).

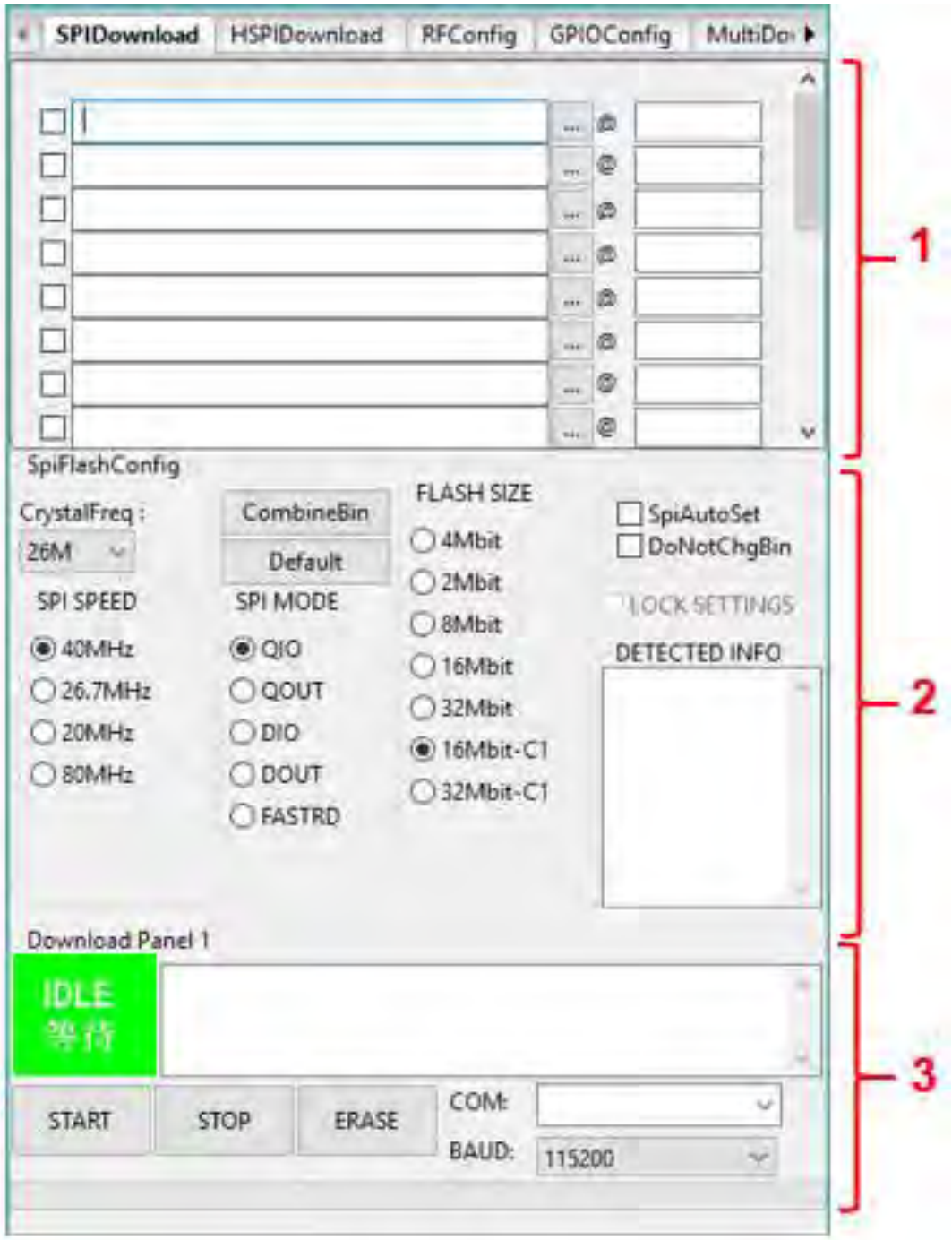


Figura A.8 *Partes de la interfaz de actualización*

Tabla A.1 *Elementos del panel de configuración SPI [2, 3].*

Elemento	Descripción
CrystalFreq	Selección de la frecuencia del cristal acorde al cristal oscilador usado en el módulo.
CmbineBin	Combina los archivos binarios seleccionados dentro de *target.bin* con la dirección 0x0000.
Default	Establece los valores por defecto para todas las áreas.
SPI SPEED	Selecciona la velocidad de reloj para la lectura/escritura de la memoria flash.
SPI MODE	Selecciona el modo de datos SPI para las operaciones de lectura/escritura de la memoria flash.
FLASH SIZE	Selecciona el modo de asignación de la memoria flash. Para diferentes tamaños del mapa de memoria, los parámetros del sistema serán guardados en diferentes regiones de la memoria flash. Cabe señalar que los valores mostrados en esta sección no son el tamaño físico de la memoria flash. En esta sección, las designaciones C1 hacen referencia a un mapa de memoria de 1024 *KB* + 1024 *KB.*
SpiAutoSet	Los archivos binarios serán descargados en la memoria flash de acuerdo con el mapa de memoria por defecto. Se recomienda no utilizar esta opción.
DoNotChgBin	Si es seleccionado, la herramienta no modificará ninguno de los archivos binarios.
Detected info	Esta ventana muestra la ID de la memoria flash y la frecuencia del cristal leído del módulo WiFi.

Tabla A.2 *Elementos del panel de descarga [2, 3].*

Elemento	Descripción
START	Iniciará con la actualización y cuando ésta termine aparecerá la palabra *FINISH* en lugar de la palabra *START*.
STOP	Detendrá la actualización.
ERASED	Borra toda la memoria flash del módulo WiFi.
MAC Address	Si la descarga es exitosa, el sistema mostrará las direcciones MAC del módulo en modo estación y punto de acceso.
COM PORT	Selecciona el puerto COM al que está conectado el módulo WiFi
BAUDRATE	Selecciona la tasa de *baud* de descarga. El valor por defecto es 115.200

A.3.3 Determinación de parámetros de actualización

Para determinar los parámetros de actualización, se debe haber realizado ya alguna de las conexiones establecidas en *A.2 Hardware* y en la interfaz de actualización se deberá seleccionar el puerto en el panel de descarga asignado al adaptador serial USB-TTL, para posteriormente:

- Reiniciar el módulo WiFi (mantener presionado el botón de reinicio por tres segundos).
- Presionar el botón *default* que se encuentra en el panel de configuración SPI.
- Desmarcar todas las casillas de verificación del panel de configuración de rutas de descarga.
- Seleccionar la tasa de *baud* de descarga. El valor por defecto es de 115.200.
- Presionar el botón *START* que se encuentra en el panel de descarga.

Tras esto, en la parte de *DETECTED INFO* del panel de configuración SPI aparecerá la información del módulo WiFi, mientras que en el panel de descarga aparecerán las direcciones MAC de las interfaces del módulo WiFi, tal y como se observa en la figura A.9.

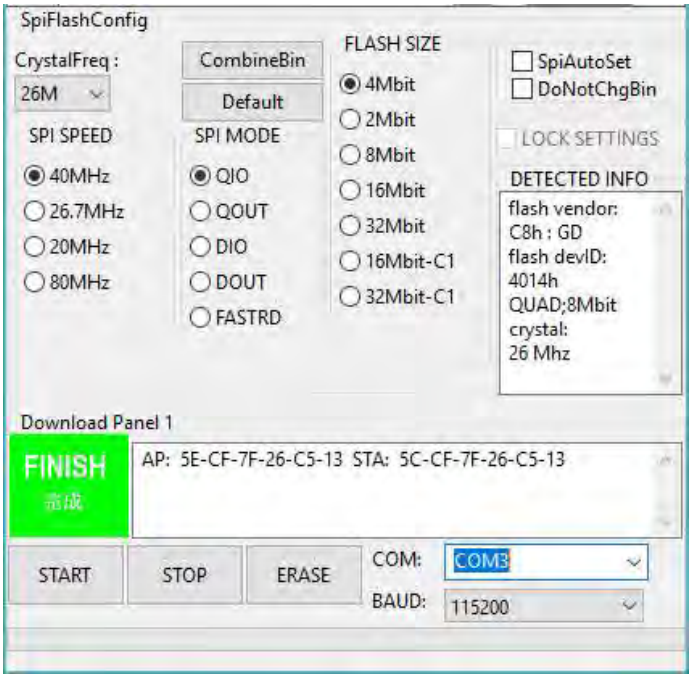


Figura A.9 *Información del módulo WiFi*

Conforme a la información obtenida, se realizarán los siguientes pasos:

- Seleccionar la frecuencia correcta del cristal en *CRYSTAL FREQ.*
- Seleccionar el tamaño correcto de la memoria flash en *FLASH SIZE*.

- Seleccionar en *SPI MODE* el modo correcto, que generalmente es QIO para la versión SDKO NONOS.

Por tanto, el panel de configuración SPI quedaría como lo muestra la figura A.10.

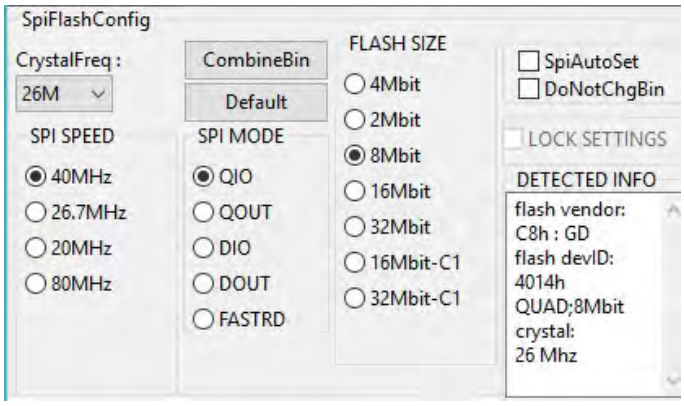


Figura A.10 *Parámetros establecidos en el panel de configuración SPI*

A.3.4 Archivos y direcciones

Los archivos binarios por utilizar y su respectiva dirección de memoria dependerán principalmente del tamaño de la memoria flash, el SoC que incorpora el módulo WiFi y si está basado en FOTA (*Firmware Over The Air*) o non-FOTA, es decir, si soporta o no descargar el *firmware* del módulo desde *Espressif Cloud* a través de WiFi. En este caso, el *firmware* de comandos AT basado en el SDK NONOS que se modificó está basado en non-FOTA. A continuación, se detallan los archivos y direcciones para estas versiones:

a) SDK NONOS

Al estar basado en non-FOTA, los archivos requeridos y sus respectivas direcciones de memoria, para diferentes tamaños de memoria flash, están especificados en la tabla A.3.

Tabla A.3 *Archivos y direcciones SDK NONOS[2].*

Binarios	**Tamaño de memoria flash [KB]**					
	512	1024	2048	4096	8192	16*1024
blank.bin	0x7B000	0xFB000	0x1FB000	0x3FB000	0x7FB000	0xFFB000
esp_init_data_default.bin	0x7C000	0xFC000	0x1FC000	0x3FC000	0x7FC000	0xFFC000
blank.bin	0x7E000	0xFE000	0x1FE000	0x3FE000	0x7FE000	0xFFE000
eagle.flash.bin	0x00000					
eagle.irom0text.bin	0x10000					

b) ESP-IDF

Los archivos requeridos y sus respectivas direcciones de memoria dependerán principalmente del tamaño de la memoria flash, el módulo WiFi, las APIs habilitadas y el SoC que incorpora el módulo WiFi. Los archivos requeridos (ubicados en la carpeta *build*) y sus respectivas direcciones de memoria se pueden obtener mediante la instrucción *make print_flash_cmd* (ver figura A.11 y figura A.12), llamada justo después de compilar el *firmware* de comandos AT con sus respectivas modificaciones [4] o inclusive, puede actualizarse directamente el *firmware* del módulo tras la compilación de éste, para mayor información de ambos casos consultar [5].

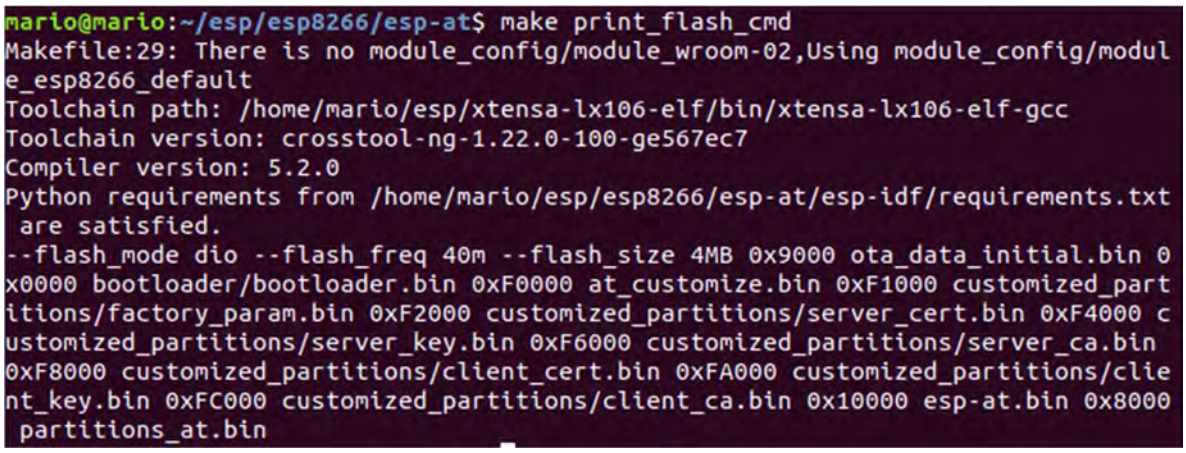


Figura A.11 Ejemplo de *archivos y direcciones de descarga para ESP8266 4MB.*

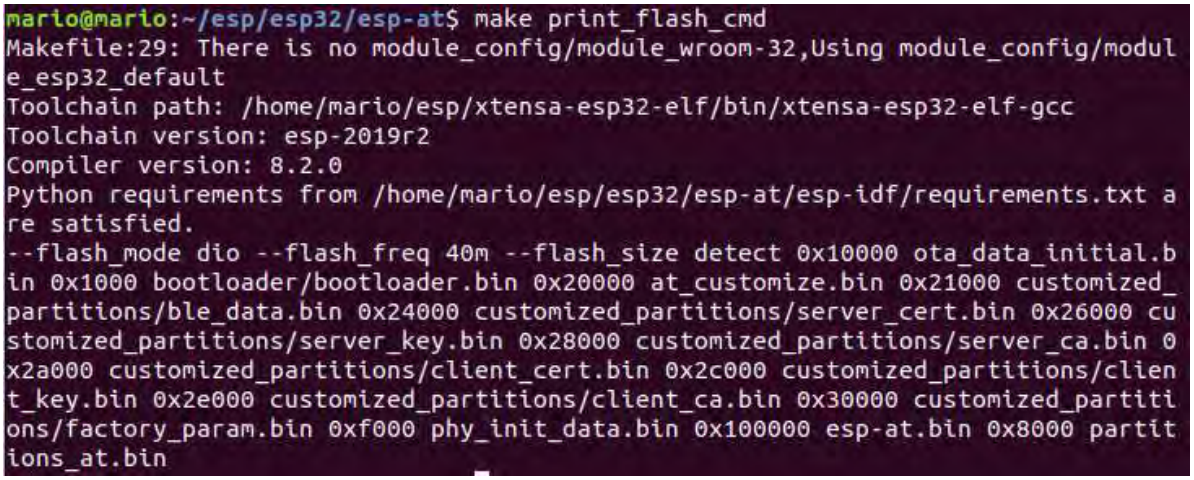


Figura A.12 *Ejemplo de archivos y direcciones de descarga para ESP32 4MB.*

A.3.5 Actualización

Para iniciar la actualización del *firmware* y tras haber configurado el panel de configuración ISP como en *A.3.3 Determinación de parámetros de actualización,* el primer paso es configurar el panel de configuración de rutas de descarga, siguiendo el procedimiento que a continuación se describe para cada uno de los archivos binarios del *firmware*.

1. Hacer clic en una casilla para colocar una marca de verificación, en ese momento toda la línea de entrada de datos se volverá de color rojo (ver figura A.13)

Figura A.13 *Línea de entrada de datos.*

2. Hacer clic sobre el botón de puntos suspensivos (…) correspondiente a la línea de entrada de datos, buscar la ubicación del archivo y hacer doble clic sobre el mismo para agregarlo. Todos los archivos necesarios para la versión SDK NONOS se encuentran dentro de la carpeta *bin*, la cual a su vez se encuentra dentro de la carpeta del *firmware*.
3. Repetir los pasos 1 y 2 para cada archivo requerido.
4. Ingresar las direcciones hexadecimales correspondientes de los archivos con extensión (.bin) en el área que se encuentra a la derecha de los botones de puntos suspensivos.

Una vez configurado el panel de configuración de rutas de descarga, en este caso para una memoria de 8 *Mbit*, éste se deberá ver tal y como se muestra en la figura A.14.

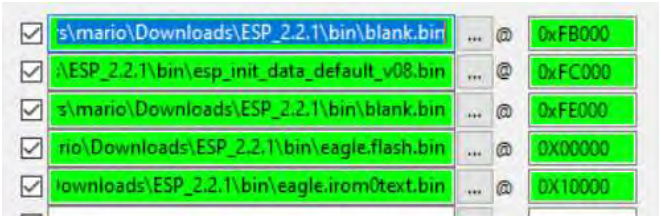


Figura A.14 *Panel de configuración de rutas de descarga configurado.*

Posteriormente se deberá reiniciar el módulo WiFi y hacer clic en el botón *START,* el cual está ubicado en el panel de descargas. Después de unos segundos, tendrá que observarse en la parte inferior de la interfaz una barra de color verde, la cual indica el progreso de la actualización (ver figura A.15) y el botón con la palabra *START*, ahora

mostrará la palabra *DOWNLOAD*. En caso de que la barra de actualización no aparezca, hacer clic en *STOP*, reiniciar el módulo WiFi y hacer clic en *START* nuevamente.

Una vez finalizada la actualización, el botón con la palabra *DOWNLOAD* mostrará ahora la palabra *FINISH y* la barra de actualización cubrirá todo el ancho de la interfaz del programa (ver figura A.16).

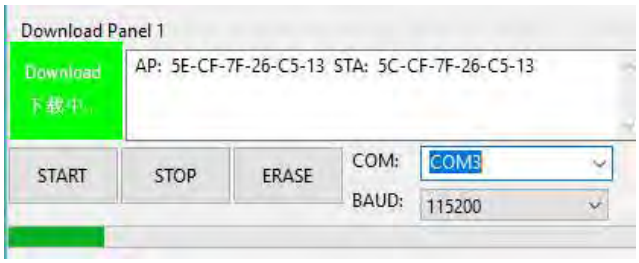


Figura A.15 *Actualización en progreso.*

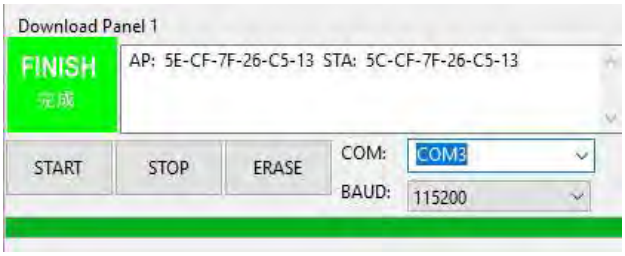


Figura A.16 *Actualización completada.*

A.3.6 Comprobación de la actualización

Para verificar que el módulo ha sido actualizado de manera correcta, se debe utilizar uno de los circuitos mostrados en la figura A.18 y figura A.19 dependiendo del SoC que incorpore. Así mismo, se deberá utilizar una terminal para comunicarse directamente con la interfaz de comunicación del *firmware* de comandos AT. En este caso se utilizó el programa Hterm® para este fin. Una vez establecida la conexión entre el módulo WiFi y la terminal se deberá reiniciar el módulo y, por tanto, recibir el mensaje "*ready*" como se muestra en la figura A.17, el cual indica que el módulo WiFi funciona correctamente.

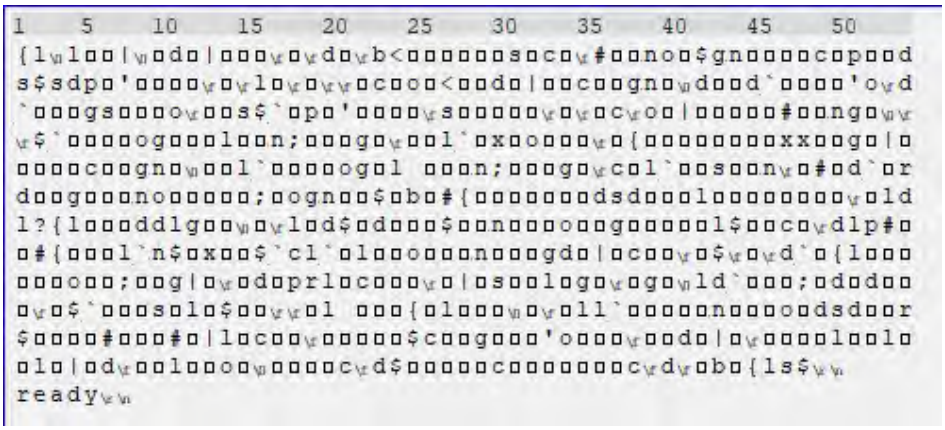


Figura A.17 *Mensaje de arranque correcto.*

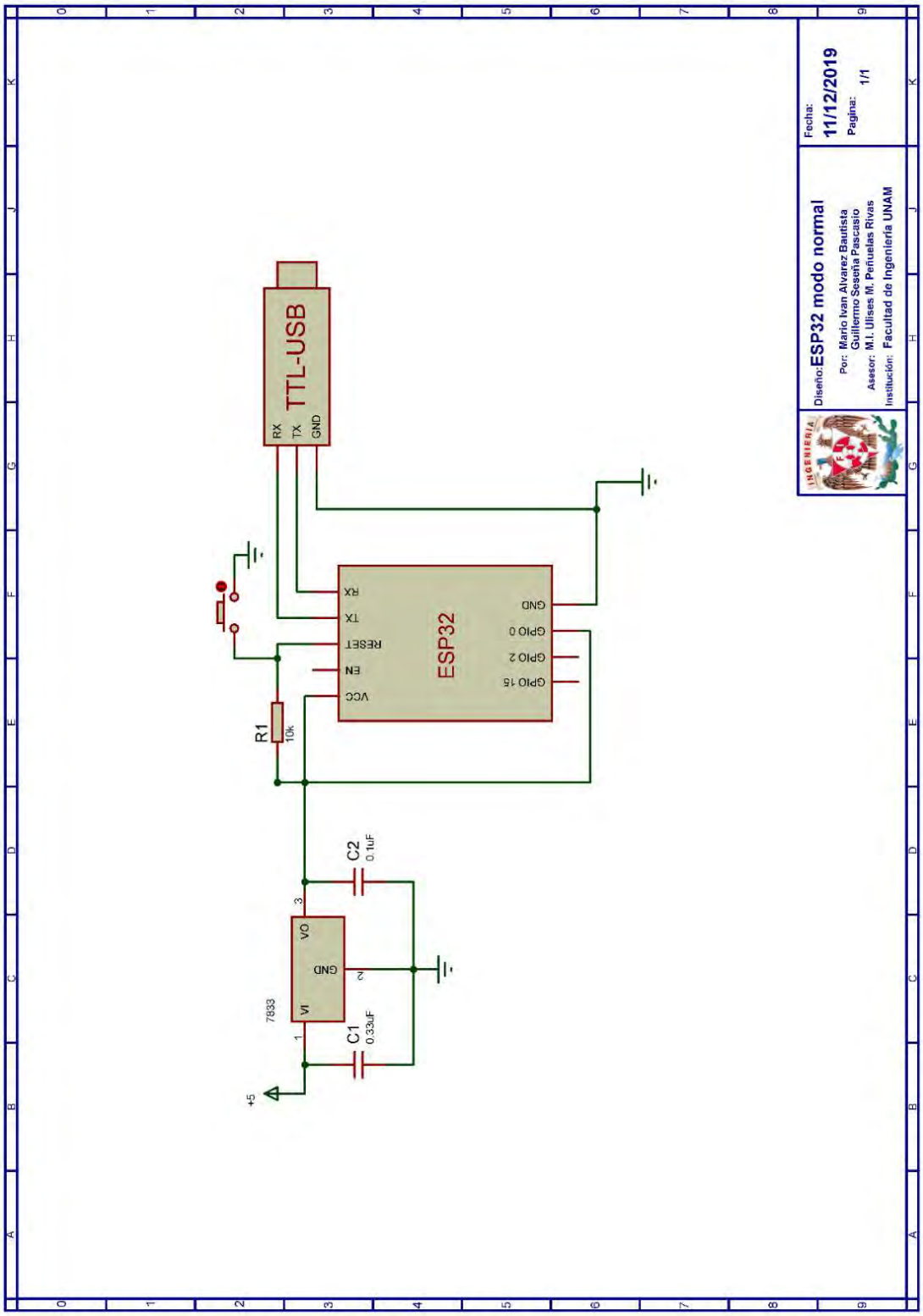


Figura A.18 *ESP32 modo normal*

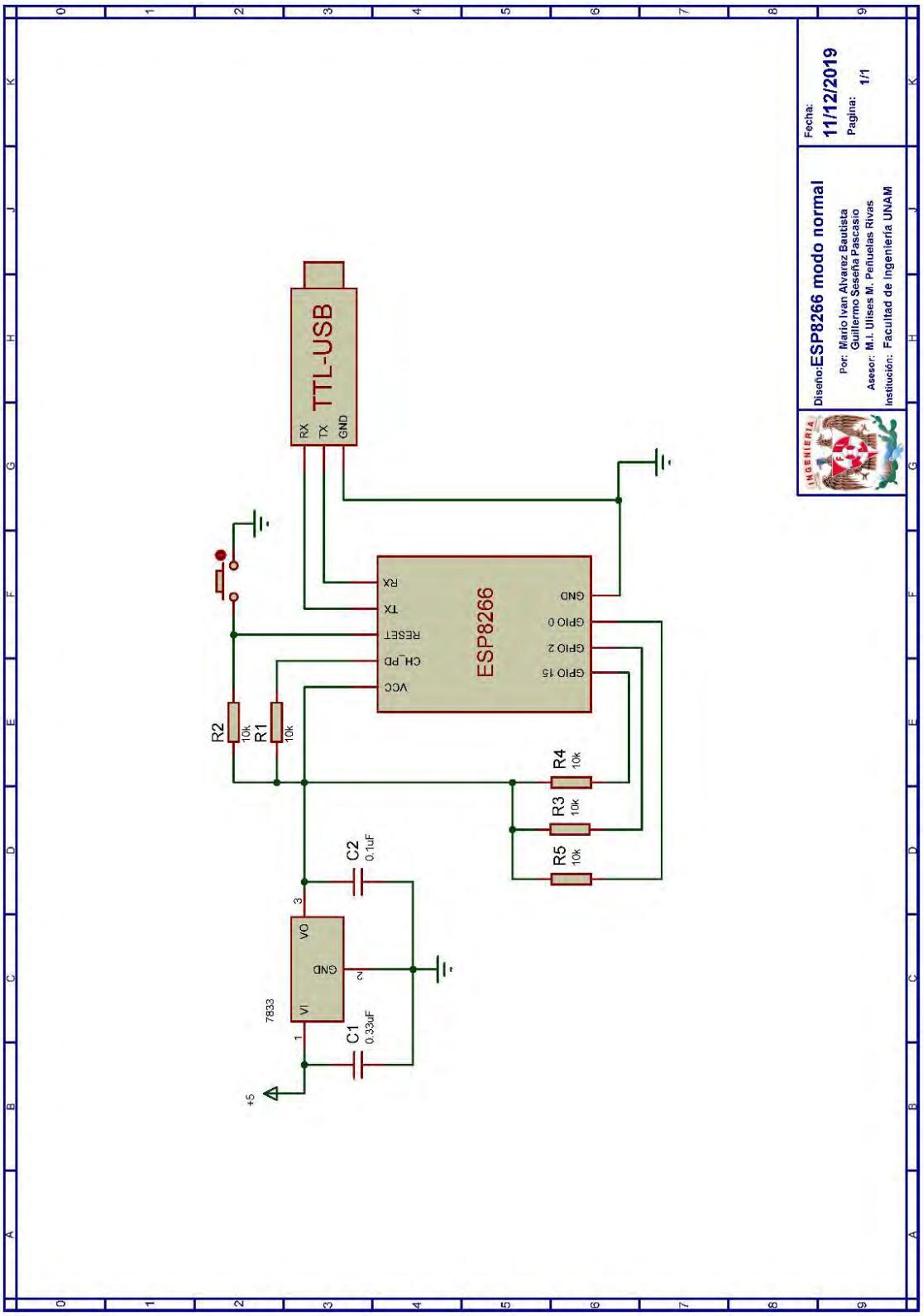


Figura A.19 *ESP8266 modo normal*

A.4 Referencias

1. Espressif Systems. esptool.py. esptool [Citado 2020 Enero]; Disponible en:https://github.com/espressif/esptool.
2. Espressif Systems, ESP8266 SDK Getting Started Guide Version 3.2. 2019.
3. Espressif Systems, Download Tool GUI instruction. 2018.
4. Espressif Systems, ESP8266 AT Instruction Set Version 3.0.2. 2019.
5. Espressif Systems. AT application for ESP32 ESP-IDF & ESP8266 ESP8266_RTOS_SDK. [Citado 2020 Febrero]; Disponible en:https://github.com/espressif/esp-at.

Apéndice B

Creación de comandos AT

Los comandos AT son comandos que permiten el control de diversas funcionalidades del SoC que incorporan los módulos WiFi, siguiendo un patrón de consulta/respuesta[1]. Estos comandos inician con ”AT+” seguido por el nombre del comando, el cual debe estar compuesto por letras en idioma inglés y no debe contener caracteres o números[2]. El tamaño máximo soportado para un comando AT es de 128 bytes, el cual comprende el nombre del comando AT, sus parámetros y el terminador de éste (\r\n) [3].

Independientemente de si el *firmware* de comandos AT está basado en SDK NONOS o en ESP-IDF, se distinguen 4 tipos de un comando (ver tabla B.1):

Tabla B.1 *Tipos de comando AT [2].*

Tipo de comando	Formato de comando	Descripción
Prueba	AT+<x>=?	Consulta los parámetros del comando y sus rangos de valores.
Consulta	AT+<x>?	Retorna el valor actual de los parámetros.
Configuración	AT+<x>=<…>	Establece el valor de los parámetros y ejecuta el comando.
Ejecución	AT+<x>	Ejecuta el comando sin definir parámetros.

A continuación, se describe el procedimiento para crear comandos AT para cada una de las versiones del *firmware* de comandos AT.

B.1 Creación de comandos AT en SDK NONOS

Como el *firmware* de comandos AT está basado en el SDK NONOS, se deberá descargar una versión de éste para crear nuevos comandos AT (o utilizar los archivos de la versión ya modificada). La versión más reciente del SDK NONOS es la v3.0.3 que contiene la versión AT v1.7.3.0, mientras que una versión previa, es decir, la versión v2.2.1 contiene la versión AT v1.6.2. Ambas versiones del SDK NONOS pueden ser descargadas desde la página oficial de Espressif Systems, cuyo enlace es:

https://www.espressif.com/en/support/download/sdks-demos

Una vez descargado el SDK NONOS, en la carpeta de éste se observará el directorio de carpetas mostrado en la figura B.1, las cuales contienen:

- *bin*: archivos binarios compilados *del firmware,* listos para ser descargados directamente en la memoria flash del módulo WiFi.
- *documents*: documentos o enlaces.
- *driver_lib*: archivos de biblioteca que controlan periféricos como UART, I2C y GPIO.
- *examples*: códigos de ejemplo para desarrollos secundarios como *IoT Demo*.
- *include*: archivos de cabecera preinstalados en el SDK. Los archivos contienen funciones de las APIs y otras macro definiciones (Se sugiere no modificar).
- *ld*: enlazador de scripts (Se sugiere no modificar).
- *lib*: archivos de biblioteca provistas en el SDK.
- *third_party*: archivos de biblioteca de código abierto de terceros.
- *tools*: herramientas necesarias para la compilación de archivos binarios (Se sugiere no modificar) [4].

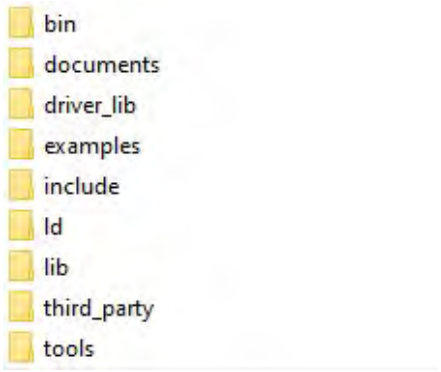


Figura B.1 *Directorio de carpetas SDK NONOS original.*

Dentro de la carpeta *examples* están las carpetas de los proyectos *at, at_espconn y at_sdio,* los cuales fueron creados por Espressif Systems para personalizar, mediante la creación de nuevos comandos, el *firmware* de comandos AT [2]. En este caso, se emplea el proyecto de comandos AT que utiliza el UART como interfaz de comunicación, por lo que se deberá copiar la carpeta *at* a la raíz del directorio del SDK tal y como se muestra en la figura B.2, cuando se trabaje en una versión nueva, ya que en la versión modificada este directorio ya está en la carpeta raíz.

La creación de comandos AT se divide en dos partes, una parte corresponde a la definición del nombre, del cual se pueden derivar hasta 4 tipos de comandos AT, y el registro de éstos, mientras que la otra parte involucra el desarrollo de las funciones que se ejecutarán en cada tipo posible de un comando y que podrán utilizar cualquiera de las APIs especificadas en [5]. Estas funciones son del tipo habitual de funciones utilizadas en programación de C embebido, cuyo tipo de retorno como lista de parámetros están definidos por la estructura *at_funcationType,* utilizada para definir los cuatro tipos de un comando [2, 5]. Estas funciones estarán contenidas en archivos con extensión (.c), los cuales estarán ubicados en *at/user*.

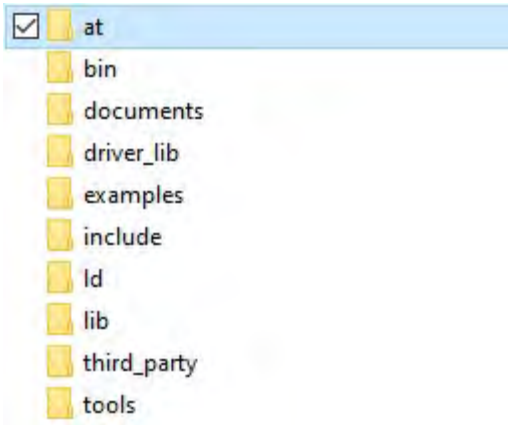


Figura B.2 *Proyecto de comandos AT.*

Las funciones de los comandos de tipo prueba, consulta y ejecución de un comando son de tipo *void,* seguido por el atributo *ICACHE_FLASH_ATTR* para almacenarlas en la IROM, esto debido al limitado tamaño de la IRAM [3], y presentan un único parámetro, mientras que las funciones de tipo configuración difieren de éstas al presentar dos parámetros.

Como ejemplo de las funciones para los comandos tipo prueba, consulta y ejecución se tiene la función mostrada en la figura B.3.Tal y como se observa, su estructura es la misma que el de una función creada para un microcontrolador PIC, sin embargo, como los comandos AT siguen una estructura consulta/respuesta, estas funciones deben retornar siempre a través de la interfaz de comunicación UART un mensaje que notifique si el comando se ha ejecutado de manera correcta o no. Para este fin, se utilizan las funciones

at_response_ok() o *at_response_error(),* también es posible retornar una respuesta personalizada mediante la funciones *os_sprintf* y *at_port_print* que son utilizadas principalmente para transmitir la información requerida a través de la interfaz de comunicación.

Es importante señalar que las funciones *at_response_ok()* o *at_response_error()* no fuerzan una salida inmediata de la función, sino que únicamente notifican el estado de ejecución del comando. Por lo que, si estas funciones no se encuentran al final de la función, es necesario llamar *return;* después de éstas para realizar una la salida inmediata de la función del comando.

```
void ICACHE_FLASH_ATTR
Nombre_funcion1(uint8_t id)
{
    //Variables locales
    uint8 valor=1;
    uint8 buffer[32] = {0};
    //Acciones
    valor=valor+1;
    os_sprintf(buffer, "%s,%d\r\n","El valor es:",valor);
    at_port_print(buffer);
    //Mensaje de retorno
    at_response_ok();
    //at_responser_error();
}
```

Figura B.3 *Ejemplo de función para comando tipo ejecución, consulta y prueba.*

Como ejemplo de las funciones para los comandos tipo configuración (como el comando ejemplo de la figura B.4) se tiene la función de la figura B.5, siendo de ésta el apuntador *pPara* el parámetro más importante, ya que es el que permite acceder a los parámetros que son declarados en este tipo de comandos. *pPara* apuntará inicialmente a la dirección de memoria que contiene el primer carácter que aparece después del nombre del comando (representado como posición cero en la figura B.4) que generalmente corresponde al símbolo “=”. Como en la figura B.4 el primer parámetro es un número, éste se obtendrá con la función *at_get_next_int_dec* (ver figura B.5*)*, la cual requiere que *pPara* apunte a la dirección de memoria del primer dígito o signo “-“ para el caso de números negativos (posición uno en la figura B.4.), una variable para almacenar el valor numérico(*result*) y una variable para almacenar un código de error(*err*) en caso de que no sea posible obtener el parámetro numérico.

AT+CMD	=	2	3	,	“	h	o	l	a	“	\r	\n
pPara	0	1	2	3	4	5	6	7	8	9	10	11

Figura B.4 *Posiciones de parámetros en comando de tipo configuración SDK NONOS.*

Tras obtener el parámetro numérico, la función *at_get_next_int_dec* hará que *pPara* apunte a la dirección de memoria que contiene el carácter que sigue después del último dígito del parámetro numérico (posición tres en la figura B.4). Este carácter debe corresponder a una “,” ya que éste es el carácter utilizado para separar los parámetros de un comando AT, y su comprobación permite verificar que el control para la obtención de los parámetros ha sido correcto hasta dicho punto.

Como en la figura B.4, el segundo parámetro es un *string,* éste se obtendrá con la función *at_data_str_copy,* la cual requiere que *pPara* apunte a las comillas que indican el inicio del *string* (posición cuatro en la figura B.4), un arreglo donde se almacenará el *string* y especificar el número máximo de caracteres que serán copiados. Esta función devuelve la longitud del *string* que haya sido copiado, la cual puede ser utilizada para verificar si la cantidad de información obtenida ha sido correcta, además de que inserta el terminador \0 tras el último carácter copiado en el arreglo destino, por lo que esto se debe de tomar en cuenta al definir el tamaño de este arreglo. Tras haber realizado la copia del *string, pPara* apuntará a la dirección de memoria que contiene el carácter que sigue después de las comillas de cierre del *string* (posición 10 en la figura B.4).

```
void ICACHE_FLASH_ATTR
Nombre_funcion(uint8_t id, char *pPara)
{
    int result = 0, err = 0, flag = 0;
    uint8 buffer[32] = {0};
    pPara++; // saltar '='
    //Obtiene primer parámetro (número)
    at_get_next_int_dec(&pPara, &result, &err);
    if (*pPara!= ',')
    {
        at_response_error();
        return;
    }
    *pPara++; // saltar ','
    os_sprintf(buffer, "Primer parámetro:%d\r\n", result);
    at_port_print(buffer);
    //Obtiene segundo parámetro
    flag=at_data_str_copy(buffer, &pPara, 10);
    if(flag<=0 || (*pPara!=',' && *pPara='\r'))
    {
        at_response_error();
        return;
    }
    at_port_print("Segundo parámetro:");
    at_port_print(buffer);
    at_port_print("\r\n");
    if (*pPara == ',')
    {
        //Procedimiento para obtener el parámetro opcional
    }
    at_response_ok();
}
```

Figura B.5 *Ejemplo de función para comando tipo configuración SDK NONOS.*

El carácter en la posición 10 puede ser “\r” que indicaría el final del comando AT o a una “,” que indicaría que existe otro parámetro opcional, ya que en la figura B.5 no existe una expresión condicional que obligue a que este carácter sea únicamente “\r”. En caso de ser un parámetro opcional, se debe obtener su valor conforme si éste es un número o un *string.*

Otro tipo de comando de utilidad, en especial en las APIs que involucran transmisión o procesamiento de información, es el que permite enviar desde un elemento externo información a la interfaz de comunicación sin que ésta sea interpretada como un comando AT, tal y como sucede con el comando AT+CIPSEND.

Como ejemplo de este tipo de comandos tenemos las funciones de la figura B.6. La función *Nombre_funcion_data* corresponde a un comando de tipo configuración y cuyo único parámetro especifica el número de datos que serán almacenados internamente en el módulo WiFi. Tras ejecutar el comando, es decir, la función *Nombre_funcion_data,* éste regresará “\r\n>” y comenzará a recibir datos a través de la interfaz de comunicación

```
int data_len=0;
uint8 buffer[32]={0};

void ICACHE_FLASH_ATTR
Obtencion_datos(uint8*  data, int32 len)
{
    static uint8 i=0; // Contador de caracteres
    int32 j=0;
    for(j=0;j<len<;j++)
    {
        if(i<data_len)
        {
            buffer[i]=data[j]
            i++;
        }
        else
        {
            i=0;
            at_register_uart_rx_intr(NULL);
            at_response_ok();
        }
    }
}
void ICACHE_FLASH_ATTR
Nombre_funcion_data(uint8_t id, char *pPara)
{
    int result = 0, err = 0, flag = 0;
    pPara++; // saltar '='
    at_get_next_int_dec(&pPara, &data_len, &err);
    at_register_uart_rx_intr(Obtención_datos);
    at_port_print("r\n>");
}
```

Figura B.6 *Ejemplo de función para la recepción de datos SDK NONOS.*

UART. Cuando la longitud de los datos definida es alcanzada, regresará " \r\nOK\r\n" y por tanto, los siguientes caracteres que sean recibidos serán interpretados como un comando AT. Para logra esto, se hace uso de la función *at_register_uart_rx_intr* que permite al usuario utilizar el UART mediante la función *callback* que se especifica en éste, en este caso, la función *obtención_datos.*

Se recomienda consultar *5.1 AT APIs* en [5], para obtener más información sobre las funciones que están directamente relacionadas con los comandos AT, como *at_response_ok, at_response_error, entre otras.*

Una vez que se han desarrollado las funciones para los comandos AT y sus tipos, el siguiente paso es establecer el nombre del comando AT y registrar las funciones para este comando. Esto se realiza en el archivo *user_main.c* que está ubicado en *at/user*. El primer paso es establecer los prototipos de las funciones tal y como se muestra en la figura B.7

```
extern void Nombre_funcion_data(uint8_t id, char *pPara);
extern void Nombre_funcion(uint8_t id, char *pPara);
extern void Nombre_funcion1(uint8_t id);
```

Figura B.7 *Prototipos de funciones de comandos AT SDK NONOS.*

Posteriormente, estas funciones se registran en el arreglo *at_custom_cmd,* donde cada elemento de éste corresponde a un comando AT. En éste se define:

- El nombre del comando AT.
- La longitud del nombre del comando.
- Nombre de la función para el comando tipo prueba o NULL si este tipo de comando no será habilitado.
- Nombre de la función para el comando tipo consulta o NULL si este tipo de comando no será habilitado.
- Nombre de la función para el comando tipo configuración o NULL si este tipo de comando no será habilitado.
- Nombre de la función para el comando tipo ejecución o NULL si este tipo de comando no será habilitado.

Este proceso para las funciones de la figura B.3, figura B.5 y figura B.6 se muestra en la figura B.8.

```
at_funcationType at_custom_cmd[] = {
    {"+TEST", 5, NULL, NULL, Nombre_funcion, Nombre_funcion1},
    {"+SND", 4, NULL, NULL,  Nombre_funcion_data, NULL}
};
```

Figura B.8 *Registro de comandos AT SDK NONOS.*

Se recomienda consultar los archivos del código fuente (.c) de los comandos AT creados para el *firmware* modificado de comandos AT, con la finalidad de tener una mayor cantidad de ejemplos sobre el desarrollo de nuevos comandos, así como del uso de cabeceras, temporizadores y demás APIs.

Estos archivos están contenidos en los *firmware,* los cuales pueden ser descargados en el siguiente enlace:

https://bit.ly/2BsLtJt

Si se desea reutilizar algunos de los comandos ya creados en una versión nueva del SDK, se deben tomar en cuenta si estos comandos involucran algunos de los cambios mencionados en el Apéndice C.

B.2 Generación de los archivos binarios SDK NONOS

Este proceso se sigue de igual manera que para los archivos ya modificados que son proporcionados en este trabajo en su propia carpeta SDK.

Para compilar el *firmware* de comandos AT y obtener los archivos binarios, es necesario descargar e instalar el ESP8266 Toolkit, tal y como se explica en *3.3ESP8266 Toolkit* en [4]. Como este *firmware* está basado en non-FOTA, se debe modificar en *ESP8266_NONOS_SDK/ld/eagle.app.v6.ld* el campo *len* de *irom0_0_seg* conforme al tamaño de la memoria flash[4]. Este campo está marcado dentro de un rectángulo rojo en la figura B.9. También, en esta misma figura está marcado con un rectángulo azul el *offset* de la dirección de memoria (0x10000) donde se debe descargar *eagle.irom0.text.bin* [4]. Es necesario verificar este campo, ya que puede cambiar en diferentes versiones del SKD, aunque para las versiones aquí utilizadas, es decir, el SDK v3.0.3 y SDK v2.2.1 este *offset* es el mismo y corresponde a 0x10000. Los valores para el campo *len* están indicados en tabla B.2.

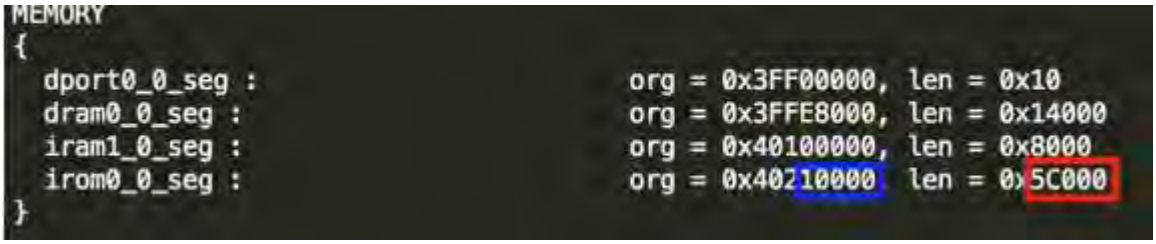


Figura B.9 *Ubicación para irom0.text[4].*

Tabla B.2 *Non-FOTA mapa flash (kB)[4].*

Flash	eagle.flash.bin	eagle.irom0text.bin	Datos del usuario	len
512	≤64	≤368	≥60	0x5C000
1024	≤64	≤752	≥176	0xBC000
2048	≤64	≤768	≥176	0xC0000
4096	≤64	≤768	≥176	0xC0000
8192	≤64	≤768	≥176	0xC0000
16*1024	≤64	≤768	≥176	0xC0000

Así mismo, al estar basado en Non-FOTA (Non *Firmware Over The Air,* Sin *Firmware* por aire) se debe comentar la línea *#define AT_CUSTOM_UPGRADE* en *ESP8266_NONOS_SDK/at/include/user_config.h,* tal y como se muestra en la figura B.10.

```
25  #ifndef __USER_CONFIG_H__
26  #define __USER_CONFIG_H__
27
28  //#define AT_CUSTOM_UPGRADE
29
30  #ifdef AT_CUSTOM_UPGRADE
31      #ifndef AT_UPGRADE_SUPPORT
32      #error "upgrade is not suppo
33      #endif
34  #endif
```

Figura B.10 *Desactivación de OTA SDK NONOS.*

Previo a iniciar el proceso de compilación, se deberá copiar la carpeta del SDK a la carpeta compartida (C:\VM\share) de la máquina virtual (ESP8266 *Toolkit*). Posteriormente se deben realizar los siguientes pasos:

- Iniciar Linux (ESP8266 *Toolkit*)
- Correr la LXTerminal que está en el escritorio de la máquina virtual.
- Ejecutar *./mount.sh* e ingresar la contraseña: *espressif*
- Utilizar el comando *cd* para acceder a la carpeta del proyecto AT, contenida en el SDK. (*cd Share/(Nombre de la carpeta que contiene el SDK)/at*)
- Ejecutar *./gen_misc.sh* [4].

El sistema mostrará el mensaje de la figura B.11.

```
Please follow below steps(1-5) to generate specific bin(s):
```

Figura B.11 *Configuración del proceso de compilación SDK NONOS.*

Por lo tanto, se deberán seleccionar las siguientes opciones:

- Para el paso 1, *choose boot version,* se elegirá la opción 2.
- Para el paso 2, *choose bin generate,* se elegirá la opción 0.
- Para el paso 3, *choose spi speed,* se elegirá generalmente la opción 2.
- Para el paso 4, *choose spi mode,* se elegirá generalmente la opción 0.
- Para el paso 5, *choosespi size and map,* se elegirá la opción que corresponda al tamaño de la memoria flash del módulo WiFi.

Una vez ingresada la última opción, iniciará el proceso de compilación donde se verificará que las funciones de los comandos AT no contengan ningún error. Si todo el proceso es exitoso se mostrará el mensaje de la figura B.12, todos los archivos requeridos para realizar la actualización del *firmware* estarán dentro de la carpeta *bin,* que a su vez estará dentro de la carpeta del proyecto de comandos AT.

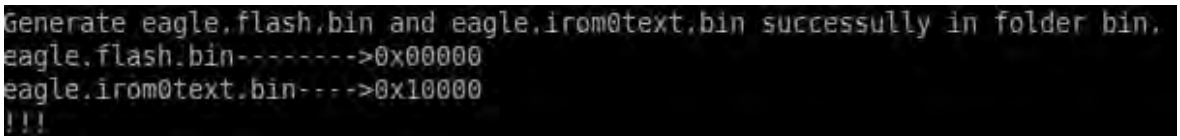


Figura B.12 *Compilación exitosa SDK NONOS.*

Es importante mencionar que, si al ejecutar uno de los nuevos comando AT el módulo WiFi se reinicia, es probable que exista un error en el código del comando como lo puede ser la llamada de APIs en orden incorrecto, por lo que se deberá verificar el código del comando AT. También, se debe dejar al menos 22 *KB* de memoria RAM en el módulo WiFi.

B.3 Traslado de modificaciones a una nueva versión SDK NONOS

Si se desea utilizar una nueva versión del SDK con la biblioteca ESP, se debe realizar en primer lugar las modificaciones que se mencionan en el Apéndice C para el *firmware* basado en NONOS, posteriormente se copiará la carpeta *at* que está en la raíz del directorio del SDK v3.0.3 que se brinda en este trabajo, a la raíz del directorio de la nueva versión de SDK. Tras esto, ya se puede realizar la compilación de los archivos como se mencionó anteriormente.

B.4 Creación de comandos AT en ESP-IDF

Como el *firmware* de comandos AT está basado en ESP-IDF, se debe completar la instalación del entorno de compilación para ESP-IDF tal y como se especifica en

ESP_AT_Get_started.md en la sección *compiling and flashing the project* en [6] y seguir los pasos hasta la copia del directorio de carpetas del proyecto de comandos AT. Así mismo, para la instalación de las diversas herramientas se debe seguir el proceso que se indica en el siguiente enlace (en el momento que se escribe este documento) hasta el paso 4:

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html

Si se crean comandos para el ESP8266 se recomienda instalar de manera independiente la *xtensa-lx106-elf* tal y como se describe en el siguiente enlace, en la sección *toolchain setup*:

https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/get-started/linux-setup.html

Mientras que para los ESP32 se recomienda instalar de manera independiente la *toolchain* xtensa-esp32-elf tal y como se describe en el siguiente enlace, en la sección *toolchain setup*:

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started-legacy/linux-setup.html (v2020) o https://docs.espressif.com/projects/esp-idf/en/v4.0/get-started-legacy/linux-setup.html (v2019)

Se recomienda realizar todo este proceso en una computadora con sistema operativo Linux o mediante una máquina virtual con Linux. Previo a la instalación del entorno IDF, se debe ejecutar en la terminal lo siguiente:

- sudo apt-get update
- sudo apt-get install build-essential gcc make perl
- reiniciar e instalar las *guest additions*.

Una vez realizado todos los pasos que se especificaron anteriormente, en *home/esp* se encontrará la carpeta del proyecto de comandos AT *esp-at,* la cual contendrá el directorio de carpetas que se muestra en la figura B.13. En este caso, el proyecto de comandos AT está configurado por defecto para utilizar UART como interfaz de comunicación por lo que no se debe realizar ninguna modificación para este fin. Si se va a añadir más comandos AT a los archivos ya modificados, se deberán copiar todos los archivos que inician con *user_(Nombre del archivo).c* (que se proporcionan en este trabajo) a la carpeta *main* del directorio que se muestra en la figura B.13. Así mismo, se deberán realizar las modificaciones especificadas en el Apéndice C a los archivos *app_main.c* y *at_uart_task.c* (ubicado en *interface/uart*) pertenecientes al directorio de la figura B.13

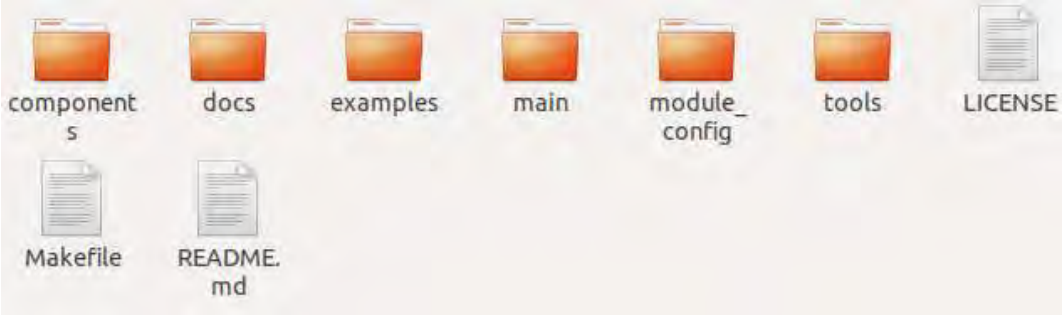


Figura B.13 *Directorio de carpetas ESP-IDF.*

Al igual que el *firmware* de comandos AT basado en SDK NONOS, la creación de comandos AT se divide en dos partes, una parte corresponde a la definición del nombre, del cual se pueden derivar hasta 4 tipos de comandos AT, y el registro éstos, mientras que la otra parte involucra el desarrollo de las funciones que se ejecutarán con cada tipo de comando y que podrán utilizar cualquiera de las APIs especificadas, al momento de escribir este documento, en la versión v3.3 de ESP-IDF, las cuales pueden ser consultadas en [7]. Estas funciones estarán contenidas en archivos con extensión (.c), los cuales estarán ubicados en *home/esp/esp-at/main.*

Todas las funciones sin importar el tipo de comando retornan un uint8_t, presentan un único parámetro y utilizan principalmente las funciones especialmente diseñadas para los comandos AT, cuyos prototipos, códigos de error y códigos de retorno están en *home/esp/esp-at/componets/at/includes/esp_at.h.* Las funciones de los comandos de tipo prueba, consulta y ejecución de un comando tienen la misma estructura y como ejemplo de éstas, se tiene la función mostrada en la figura B.14.

```
uint8_t Nombre_funcion_pce(uint8_t *cmd_name)
{
    //Variables locales
    uint8_t valor=1;
    uint8_t buffer[32]={0};
    //Acciones
    valor=valor+1;
    snprintf((char*)buffer,sizeof(buffer) - 1,"El valor es:%d\r\n",(uint8_t)valor);
    esp_at_port_write_data(buffer,strlen((char*)buffer));
    return ESP_AT_RESULT_CODE_OK;
}
```

Figura B.14 *Ejemplo de función para comando tipo ejecución, consulta y prueba.*

Tal y como se observa, su estructura es la misma que el de una función creada para un microcontrolador PIC, cuyo parámetro es un apuntador hacia el nombre del comando AT que llama a esta función. Sin embargo, como los comandos AT siguen una estructura consulta/respuesta, estas funciones deben retornar siempre a través de la interfaz de comunicación UART un mensaje que notifique si el comando se ha ejecutado de manera

correcta o no. Para este fin, se puede utilizar la instrucción return result_code; que imprimirá un mensaje en la interfaz de comunicación y realizará una la salida inmediata de la función del comando; o la función esp_at_response_result(uint8_t result_code) que imprimirá un mensaje en la interfaz de comunicación, pero no ejecuta una salida inmediata de la función. Los result_code validos están en la tabla B.3.

Tabla B.3 *Códigos de respuesta comandos AT.*

Valor	Código	Mensaje
0x00	ESP_AT_RESULT_CODE_OK	\r\nOK\r\n
0x01	ESP_AT_RESULT_CODE_ERROR	\r\nERROR\r\n
0x02	ESP_AT_RESULT_CODE_FAIL	\r\nFAIL\r\n
0x03	ESP_AT_RESULT_CODE_SEND_OK	\r\nSEND OK\r\n
0x04	ESP_AT_RESULT_CODE_SEND_FAIL	\r\nSEND FAIL\r\n
0x05	ESP_AT_RESULT_CODE_IGNORE	*
0x06	ESP_AT_RESULT_CODE_PROCESS_DONE	**

*No genera resultado, pero no puede procesar el siguiente comando, al intentarlo retornará BUSY.

** No genera resultado, pero puede procesar el siguiente comando.

También, es posible retornar una respuesta personalizada mediante las funciones *snprintf* y *esp_at_port_write_data* que son utilizadas principalmente para transmitir la información requerida a través de la interfaz de comunicación.

Como ejemplo de las funciones para los comandos tipo configuración (ver figura B.15) se tiene la función de la figura B.16, siendo de ésta *para_num* el parámetro más importante puesto que indica el número de parámetros que el usuario ingresó con el comando AT, permitiendo verificar si el comando cuenta con los parámetros necesarios para ejecutarse (ver figura B.16). Como en la figura B.15 el primer parámetro es un número, éste se obtendrá con la función esp*_at_get_para_as_digit*, la cual requiere el índice de la posición del parámetro (cero en la figura B.15) y la dirección de memoria de la variable donde se almacenará el valor de éste. Esta función retorna un valor que indica si la obtención del parámetro se realizó de manera exitosa o no.

AT+CMD=	23	,	“hola”	\r\n
Índice de parámetros	0		1	

Figura B.15 *Posiciones de parámetros en comando de tipo configuración ESP-IDF.*

Como el segundo parámetro de la figura B.15 es un *string,* éste se obtendrá con la función *esp_at_get_para_as_str* que requiere del índice de la posición del parámetro (uno en la figura B.15) y un apuntador para almacenar la dirección de memoria del inicio del *string.* También, esta función retorna un valor que indica si la obtención de la dirección del *string* se realizó de manera exitosa o no. Si el *string* será ocupado de manera global, es decir,

ocupado por otras funciones o comandos tras la ejecución del comando AT, se deberá almacenar éste en un arreglo declarado de forma global. Así mismo, si el parámetro *string* del comando contendrá una coma, ésta debe ser escrita como “\,”, ya que de no ser así *para_num* obtendrá un valor incorrecto respecto al número de parámetros declarados.

Para parámetros opcionales, *para_num* debe ser mayor que el número de parámetros generalmente establecido, por lo que se deberá comprobar con una expresión condicional (ver figura B.16) y se debe obtener su valor conforme si éste es un número o un *string.* Los valores con los que se pueden comparar los valores de retorno de las funciones esp*_at_get_para_as_digit* y *esp_at_get_para_as_str* están en la *tabla* B.4*.*

```
uint8_t Nombre_funcion(uint8_t para_num)
{
    int32_t valor = 0, cnt = 0;
    uint8_t buffer[32]={0};
    uint8_t* ptr_str = NULL;
    if(para_num<2) // Comprobación de parámetros mínimos
        return ESP_AT_RESULT_CODE_ERROR;
    //Obtiene primer parámetro (número)
    if (esp_at_get_para_as_digit(cnt,&valor) != ESP_AT_PARA_PARSE_RESULT_OK)
    {
        return ESP_AT_RESULT_CODE_ERROR;
    }
    cnt++; // Siguiente parámetro
    snprintf((char*)buffer,sizeof(buffer) - 1,"Primer parámetro:%d\r\n",valor);
    esp_at_port_write_data(buffer,strlen((char*)buffer));
    //Obtiene segundo parámetro
    if(esp_at_get_para_as_str(cnt,&ptr_str) != ESP_AT_PARA_PARSE_RESULT_OK)
    {
        return ESP_AT_RESULT_CODE_ERROR;
    }
    esp_at_port_write_data((uint8_t*)"Segundo parámetro:",strlen("Segundo parámetro"));
    esp_at_port_write_data(ptr_str,strlen(ptr_str);
    if(para_num>=3)
    {
        //Procedimiento para obtener el parámetro opcional
    }
    return ESP_AT_RESULT_CODE_OK;
}
```

Figura B.16 *Ejemplo de función para comando tipo configuración ESP-IDF.*

Tabla B.4 *Retorno de funciones para obtención de parámetros ESP-IDF.*

Valor	Código	Mensaje
-1	ESP_AT_PARA_PARSE_RESULT_FAIL	error de análisis, quizás el tipo de parámetro no coincida o esté fuera de rango
0	ESP_AT_PARA_PARSE_RESULT_OK	Exitoso
1	ESP_AT_PARA_PARSE_RESULT_OMITTED	El parámetro es omitido

Otro tipo de comando útil en especial en las APIs que involucran transmisión o procesamiento de información es el que permite enviar desde un elemento externo información a la interfaz de comunicación sin que ésta sea interpretada como un comando AT, tal y como sucede con el comando AT+CIPSEND. Como ejemplo de este tipo de comandos tenemos las funciones de la figura B.18.

La función *Nombre_funcion_get* corresponde a un comando de tipo configuración y cuyo único parámetro especifica el número de datos que serán almacenados internamente en el módulo WiFi. Tras ejecutar el comando, es decir, la función *Nombre_funcion_get,* éste regresará "\r\n>" e iniciará a recibir datos a través de la interfaz de comunicación UART. Cuando la longitud de los datos definida es alcanzada, regresará " \r\nSEND OK\r\n" y por tanto, los siguientes caracteres que sean recibidos serán interpretados como un comando AT. Para logra esto, se utiliza un semáforo binario y la función *esp_at_port_enter_specific* para establecer la *callback,* la función *wait_data_callback,* que será llamada tras recibir estos datos y que liberará el semáforo.

Para la obtención y almacenamiento de los datos se debe utilizar la función e*sp_at_port_read_data* que requiere de la dirección de memoria del *buffer* donde se almacenarán los datos, así como el número máximo de datos que se recibirán. Esta función retorna el número de datos que fueron recibidos.

Una vez alcanzado el número de datos a recibir, se llama a la función *esp_at_port_exit_specific* para desactivar la *callback* y se libera el semáforo. Se recomienda consultar el siguiente enlace para obtener más detalles sobre la creación de comandos AT para ESP-IDF:

https://blog.csdn.net/espressif/article/details/102777084

Una vez que se han desarrollado las funciones para los comandos AT y sus tipos, el siguiente paso es establecer el nombre del comando AT y registrar las funciones para este comando. Esto se realiza en el archivo *app_main.c* que está ubicado en *home/esp/esp_at/main*. El primer paso es establecer los prototipos de las funciones tal y como se muestra en la figura B.17.

```
extern uint8_t Nombre_funcion_pce(uint8_t *cmd_name);
extern uint8_t Nombre_funcion(uint8_t para_num);
extern uint8_t Nombre_funcion_get(uint8_t para_num);
```

Figura B.17 *Prototipos de funciones ESP-IDF.*

```c
static void wait_data_callback (void)
{
    xSemaphoreGive(sync_sema);
}

uint8_t Nombre_funcion_get(uint8_t para_num)
{
    int32_t len_total=0, cnt=0;
    uint16_t aux_len=0;
    uint8_t buffer[32]={0};
    if(para_num <1)
    {
        return ESP_AT_RESULT_CODE_ERROR;
    }
    //Número de datos a recibir
    if(esp_at_get_para_as_digit(cnt,&len_total)!= ESP_AT_PARA_PARSE_RESULT_OK)
    {
        return ESP_AT_RESULT_CODE_ERROR;
    }
    vSemaphoreCreateBinary(sync_sema);
    xSemaphoreTake(sync_sema,portMAX_DELAY);
    esp_at_port_write_data((uint8_t *)"\r\n>",strlen("\r\n>"));
    esp_at_port_enter_specific(wait_data_callback);
    //Obtención de datos
    while(xSemaphoreTake(sync_sema,portMAX_DELAY))
    {
        aux_len=(uint16_t)esp_at_port_read_data(&buffer[aux_len],(total_len-aux_len))+ aux_len;
        if(aux_len>=total_len)
        {
            esp_at_port_exit_specific();
            xSemaphoreGive(sync_sema);
            break;
        }
    }
    return ESP_AT_RESULT_CODE_SEND_OK;
}
```

Figura B.18 *Ejemplo de función para la recepción de datos ESP-IDF.*

Posteriormente, se crea un arreglo de la estructura *esp_at_cmd_struct* donde cada elemento de éste corresponde a un comando AT. En éste se define:

- El nombre del comando AT.
- Nombre de la función para el comando tipo prueba o NULL si este tipo de comando no será habilitado.
- Nombre de la función para el comando tipo consulta o NULL si este tipo de comando no será habilitado.

- Nombre de la función para el comando tipo configuración o NULL si este tipo de comando no será habilitado.
- Nombre de la función para el comando tipo ejecución o NULL si este tipo de comando no será habilitado.

Este proceso para las funciones de la figura B.14, figura B.16 y figura B.18 se muestran en la figura B.19.

```
esp_at_cmd_struct at_user_cmd[] =
{
    {"+TEST", NULL, NULL, Nombre_funcion, Nombre_funcion_pce},
    {"+SND", NULL, NULL, Nombre_funcion_get, NULL}
}
```

Figura B.19 *Registro de comandos AT ESP-IDF.*

Finalmente, para completar el registro de los comandos AT se deberá llamar *esp_at_custom_cmd_array_regist,* cuyos parámetros son el arreglo que contiene los nuevos comandos AT y el número de elementos de éste, previo a *at_custom_init* tal y como se observa en la *figura* B.20. Todo esto es dentro de la función *app_main* contenida en el mismo archivo *app_main*.c.

```
esp_at_custom_cmd_array_regist (at_user_cmd, sizeof(at_user_cmd)/sizeof(at_user_cmd[0]));
at_custom_init();
```

Figura B.20 *Declaración del registro de comandos AT ESP-IDF.*

Se recomienda consultar los archivos (.c) de los comandos AT creados para el *firmware* modificado de comandos AT, con la finalidad de tener una mayor cantidad de ejemplos sobre el desarrollo de nuevos comandos, así como del uso de cabeceras, temporizadores y demás APIs.

Estos archivos están contenidos en los *firmware,* los cuales pueden ser descargados en el siguiente enlace:

https://bit.ly/2BsLtJt

B.5 Generación de los archivos binarios ESP-IDF

El proceso que se explicará aquí considera la utilización de una computadora con sistema operativo Linux o mediante una máquina virtual con Linux.

Previo a iniciar con la compilación del *firmware* se deberá especificar el SoC y el módulo WiFi para el cual se compilará. Esto se realiza en el archivo *makefile* que está en *home/esp/esp_at.* Para el SoC ESP32 se deberá establecer como se muestra en la figura B.21, mientras que para el SoC ESP8266 se deberá establecer como se muestra en la figura B.22 [6].

```
export ESP_AT_PROJECT_PLATFORM ?= PLATFORM_ESP32
export ESP_AT_MODULE_NAME ?= WROOM-32
```

Figura B.21 *Configuración para SoC ESP32 [6].*

```
export ESP_AT_PROJECT_PLATFORM ?= PLATFORM_ESP8266
export ESP_AT_MODULE_NAME ?= WROOM-02
```

Figura B.22 *Configuración para SoC ESP8266 [6].*

Para iniciar con la compilación del *firmware* de comandos AT y generar los archivos binarios se deberá abrir la terminal e ingresar hasta el directorio *home/esp/esp_at,* una vez ahí, se ejecutarán los siguientes comandos:

- *rm sdkconfig*, para eliminar la configuración anterior.
- *make defconfig* para establecer la configuración predeterminada.
- *make menuconfig:*
    - *Serial flasher->flash size* para establecer el tamaño de la memoria del módulo
    - *Component config->AT-> AT MQTT* (Deshabilitar)

      ->AT HTTP (Deshabilitar)

      ->AT ETHERNET (Deshabilitar, sólo disponible en ESP32)

      ->Mbedtls->symmetric ciphers (Habilitar todos menos *des block cipher*)

      *->Mdns->*Establecer 2 como el número de servicios

      *->Phy->*Establecer 33 en *vdd const value* (Disponible sólo en ESP8266)
- *make* para compilar el *firmware* de comandos AT [6].

Los cambios establecidos en *make menuconfig* son los cambios requeridos para la biblioteca ESP. Si la compilación fue exitosa, en la terminal se mostrará un mensaje similar al de la figura B.23. En este mismo mensaje se especifican los archivos y las direcciones de memoria donde se deberán descargar. Estos archivos se encontrarán en *home/esp/esp_at/build*. Para más detalles sobre este proceso consultar [6].

```
To flash all build output, run 'make flash' or:
python /home/mario/esp/esp-at/esp-idf/components/esptool_py/esptool/esptool.py
--chip esp32 --port /dev/ttyUSB0 --baud 115200 --before default_reset --after h
ard_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_size detect
0x10000 /home/mario/esp/esp-at/build/ota_data_initial.bin 0x1000 /home/mario/es
p/esp-at/build/bootloader/bootloader.bin 0x20000 /home/mario/esp/esp-at/build/a
t_customize.bin 0x21000 /home/mario/esp/esp-at/build/customized_partitions/ble_
data.bin 0x24000 /home/mario/esp/esp-at/build/customized_partitions/server_cert
.bin 0x26000 /home/mario/esp/esp-at/build/customized_partitions/server_key.bin
0x28000 /home/mario/esp/esp-at/build/customized_partitions/server_ca.bin 0x2a00
0 /home/mario/esp/esp-at/build/customized_partitions/client_cert.bin 0x2c000 /h
ome/mario/esp/esp-at/build/customized_partitions/client_key.bin 0x2e000 /home/m
ario/esp/esp-at/build/customized_partitions/client_ca.bin 0x30000 /home/mario/e
sp/esp-at/build/customized_partitions/factory_param.bin 0x37000 /home/mario/esp
/esp-at/build/customized_partitions/mqtt_cert.bin 0x39000 /home/mario/esp/esp-a
t/build/customized_partitions/mqtt_key.bin 0x3B000 /home/mario/esp/esp-at/build
/customized_partitions/mqtt_ca.bin 0xf000 /home/mario/esp/esp-at/build/phy_init
_data.bin 0x100000 /home/mario/esp/esp-at/build/esp-at.bin 0x8000 /home/mario/e
sp/esp-at/build/partitions_at.bin
```

Figura B.23 *Compilación exitosa del firmware ESP-IDF.*

Es importante mencionar que, si al ejecutar uno de los nuevos comando AT el módulo WiFi se reinicia, es probable que en la función del comando esté llamando APIs en orden incorrecto, por lo que se deberá verificar. También, se debe dejar al menos 22 *KB* de memoria RAM en el módulo WiFi, ya que para sus aplicaciones originales este usa memoria dinámica.

B.6 Traslado de modificaciones a una nueva versión ESP-IDF

Si se desea utilizar una nueva versión de ESP-IDF con la biblioteca ESP, independientemente de si es para el SoC ESP2866 o los ESP32, se debe realizar en primer lugar las modificaciones que se mencionan en el Apéndice C para el *firmware* basado en ESP-IDF en los archivos descargados de la nueva versión. Posteriormente, se deberán copiar todos los archivos que inician con *user_(Nombre del archivo).c,* que se proporcionan en este trabajo para la versión IDF, a la carpeta *main* del directorio que se muestra en la figura B.13 que corresponderá con los archivos descargados de la nueva versión. Así mismo, se deberán realizar las modificaciones que a continuación se mencionan en el archivo *app_main.c* que ya está en el directorio de la figura B.13.

- Se deberá copiar del archivo *app_main.c* que se proporciona en este trabajo, los prototipos de las funciones de los comandos AT, así como su declaración en la estructura de comandos AT:
    - Línea 145-253 en IDF ESP32.
    - Línea 145-251 en IDF ESP8266
- Reemplazar la función *static esp_err_t at_wifi_event_handler(void *ctx, system_event_t *event)* por la contenida en el archivo *app_main.c* que se proporciona en este trabajo, y que se encuentra en la línea 113 tanto en IDF ESP32 como en IDF ESP8266
- Copiar la función *esp_at_custom_cmd_array_regist* antes de la función *at_custom_init();* la función que debe ser copiada está en:
    - *Línea 431 en IDF ESP32*
    - *Línea 429 en IDF ESP8266*

B.7 Referencias

1. Ciuffoletti, A., *Design and implementation of a low cost modular sensor.* Second International Workshop on Smart Environments and Urban Networking, 2017.
2. Espressif Systems, *ESP8266 AT Instruction Set Version 3.0.2*. 2019.
3. Espressif Systems, *ESP8266EX Frequently Asked Questions*, in *Version 1.7*. 2019.
4. Espressif Systems, *ESP8266 SDK Getting Started Guide Version 3.2*. 2019.
5. Espressif Systems, *ESP8266 Non-OS SDK API Reference Version 3.0.1*. 2019.
6. Espressif Systems. *AT application for ESP32 ESP-IDF & ESP8266 ESP8266_RTOS_SDK*. [Citado 2020 Febrero]; Disponible en:https://github.com/espressif/esp-at.
7. Espressif Systems. *API Reference*. [Citado 2020 Marzo]; Disponible en:https://docs.espressif.com/projects/esp-idf/en/v3.3/api-reference/index.html.

Apéndice C

Modificaciones y configuraciones realizadas a los *firmware* de comandos AT

C.1 Modificaciones del *firmware* basado en SKD NONOS

A continuación, se detallan las modificaciones realizadas que permitieron la creación de ciertos comandos y herramientas en la biblioteca ESP.

C.1.1 Modificaciones para SDK NONOS V2.2.1

a) Añadir biblioteca MBEDTLS

Se modificó este *firmware* para utilizar la biblioteca MBEDTLS en lugar de la biblioteca SSL. Entre las características que ofrece esta nueva biblioteca están:

- Soporta TLS 1.0, TLS 1.1 y TLS 1.2. Ya no soporta SSL 3.0.
- Memoria caché de 2048 hasta 8192 bytes.
- Soporta algoritmos de cifrado AES-128 y AES-256 y el modo CBC.
- Soporta algoritmos *hash* SHA-1, SHA-256, SHA-384 y SHA-512.
- Soporta algoritmos RSA-512, RSA-1024, RSA-2048.
- Soporta certificados en formato PEM y DER.

- Soporta autentificación tanto unidireccional como bidireccional.
- Soporta el análisis de la cadena de certificados de tres niveles.
- No soporta la verificación de la cadena de certificados de tres niveles con el certificado raíz.

Para realizar esta modificación se deben descargar los archivos de esta biblioteca, contenidos en ESP8266 NONOS SDK MBEDTLS 20160718. Tras la descarga, se observará el directorio de carpetas de la figura C.1 [1].

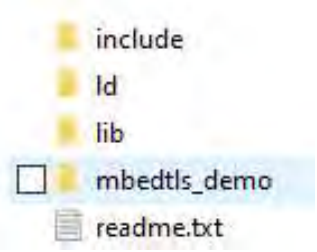


Figura C.1 *Directorio de carpetas de la biblioteca MBEDTLS.*

```
LINKFLAGS_eagle.app.v6 = \
    -L../lib        \
    -nostdlib   \
    -T$(LD_FILE)   \
    -Wl,--no-check-sections \
    -Wl,--gc-sections   \
    -u call_user_start  \
    -Wl,-static                 \
    -Wl,--start-group                   \
    -lc                 \
    -lgcc                   \
    -lhal                   \
    -lphy   \
    -lpp    \
    -lnet80211  \
    -llwip  \
    -lwpa   \
    -lwpa2  \
    -lcrypto    \
    -lmain  \
    -lpwm   \
    -ljson  \
    -lupgrade   \
    -lmbedtls       \
    -lwps       \
    -lsmartconfig   \
    -lairkiss       \
    $(DEP_LIBS_eagle.app.v6)
```

Figura C.2 *Modificación de makefile para agregar MBEDTLS.*

El contenido de la carpeta *lib* debe ser copiado a la carpeta *lib* del SDK y en el archivo *makefile* que está contenido en la carpeta del proyecto de comandos AT, así como el archivo *makefile* contenido en la carpeta *third_party,* se debe remplazar *"-lssl"* por *"-lmbedtls"* (ver figura C.2). Tal y como se mencionó en el Apéndice B, la carpeta del proyecto de comandos AT llamada *at* que está ubicada en *ESP8266_NONOS_SDK/examples* debe ser copiada a la raíz del directorio de carpetas de *ESP8266_NONOS_SDK* [1].

b) Añadir API espnow

Para utilizar las funciones de la API *espnow* se debe modificar el archivo *makefile* que está contenido en la carpeta del proyecto de comandos AT, agregando *"-lespnow" a* la lista de bibliotecas de dicho archivo, tal y como se muestra en la figura C.3.

```
LINKFLAGS_eagle.app.v6 = \
    -L../lib        \
    -nostdlib   \
    -T$(LD_FILE)   \
    -Wl,--no-check-sections \
    -Wl,--gc-sections   \
    -u call_user_start  \
    -Wl,-static                 \
    -Wl,--start-group                   \
    -lc                 \
    -lgcc                   \
    -lhal                   \
    -lphy   \
    -lpp    \
    -lnet80211  \
    -llwip  \
    -lwpa   \
    -lwpa2  \
    -lcrypto    \
    -lmain  \
    -lespnow    \
    -lpwm   \
    -ljson  \
    -lupgrade   \
    -lmbedtls       \
    -lwps       \
    -lsmartconfig   \
    -lairkiss       \
    $(DEP_LIBS_eagle.app.v6)
```

Figura C.3 *Modificación de makefile para agregar ESPNOW y PWM.*

c) Habilitar funciones criptográficas, ciphersuites y modificaciones para uso de fingerprint

Para habilitar el uso de ciertos algoritmos de cifrado y herramientas, se modificó el archivo *config_esp.h* que se encuentra en *ESP8266_NONOS_SDK/third_party/include/mbedtls.* Las modificaciones que se realizaron son:

- Descomentar la línea 1463 del código para habilitar ARC4 y añadir los siguientes *ciphersuites* para SSL*:*
    - mbedtls_tls_ecdh_ecdsa_with_rc4_128_sha
    - mbedtls_tls_ecdh_rsa_with_rc4_128_sha
    - mbedtls_tls_ecdhe_ecdsa_with_rc4_128_sha
    - mbedtls_tls_ecdhe_rsa_with_rc4_128_sha
    - mbedtls_tls_ecdhe_psk_with_rc4_128_sha
    - mbedtls_tls_dhe_psk_with_rc4_128_sha
    - mbedtls_tls_rsa_with_rc4_128_sha
    - mbedtls_tls_rsa_with_rc4_128_md5
    - mbedtls_tls_rsa_psk_with_rc4_128_sha
    - mbedtls_tls_psk_with_rc4_128_sha
- Descomentar la línea 1528 del código para habilitar *blowfish*.
- Descomentar la línea 2397 del código para habilitar XTEA.

Para desarrollar los comandos AT que permitieran emplear el uso del *fingerprint* de un certificado para su validación*,* se modificó el archivo *ssl_tls.c* el cual se encuentra en *ESP8266_NONOS_SDK/third_party/mbedtls/library/ssl_tls.c* Las modificaciones que se realizaron en el archivo que se proporciona en este trabajo son:

- Líneas 57-59, se declaran las variables que serán ocupadas para desarrollar *fingerprint.*
- Líneas 4193-4238, función *setupfingerprint* que es utilizada para habilitar o deshabilitar la obtención del *fingerprint* durante la conexión. El método para obtener el *fingerprint* es determinado por la longitud del *fingerprint* dado por el usuario y que será utilizado para comparar con el obtenido durante la negociación.
- Líneas 4401-4426, expresión condicional que es ejecutada durante el *handshake* cuando se ha habilitado la obtención del *fingerprint* y el módulo WiFi actúa como un cliente SSL. Dentro de ésta se obtiene el *fingerprint* con el método establecido en *setupfingerprint* y compara éste con el dado por el usuario. En caso de ser iguales prosigue con el proceso de conexión, en caso contrario, detiene este proceso y notifica de este error en la interfaz de comunicación.

Para que estos cambios sean aplicados se debe compilar nuevamente la biblioteca MBEDTLS. Esta compilación se realiza en el *ESP8266 Toolkit,* cuyo proceso de instalación se explica en *3.3ESP8266 Toolkit* en [2]. Así mismo, se deberá copiar la carpeta *xtensa* dentro de la carpeta *ESP8266_NONOS_SDK/third_party/include*, ya que de no hacer esto, durante el proceso de compilación mostrará el error de la figura C.4.

Figura C.4 *Error de compilación MBEDTLS.*

La carpeta *xtensa está* ubicada en *ESP8266_RTOS_SDK/extra_include* puede ser substraída del ESP8266 RTOS SDK v2.0.0, el cual se descarga del siguiente enlace:

https://www.espressif.com/en/support/download/sdks-demos

Los pasos por seguir para realizar la compilación de la biblioteca MBEDTLS son:

- Iniciar Linux (ESP8266 *Toolkit*).
- Correr la LXTerminal que está en el escritorio de la máquina virtual.
- Ejecutar *./mount.sh* e ingresar la contraseña: *espressif.*
- Utilizar el comando *cd* para acceder a la carpeta *third_party* contenida en el SDK. (*cd Share/(Nombre de la carpeta que contiene el SDK)/ third_party*).
- Ejecutar el comando *make_lib.sh mbedtls.*

Si la compilación fue exitosa, mostrará el mensaje de la figura C.5.

Figura C.5 *Compilación exitosa MBEDTLS.*

d) Modificaciones a los archivos de la API MDNS.

Con la finalidad de dar mayor flexibilidad en cuanto al uso de la API MDNS, se realizaron modificaciones conforme a lo establecido en el RFC 6762 y el RFC 1035. Estas modificaciones fueron hechas en dos archivos. El primero corresponde al archivo de cabecera de la API MDNS llamado *mdns.h,* que está ubicado en *ESP8266_NONOS_SDK/third_party/include/lwip* y cuyas modificaciones son:

- Líneas 96-99, se modificó la estructura *mdns_info* para agregar cuatro nuevos miembros para anunciar dos servicios MDNS.

El segundo archivo que fue modificado es *mdns.c* el cual está ubicado en ESP*8266_NONOS_SDK/third_party/lwip/core* y cuyas modificaciones son:

- Línea 52, se agregó el archivo de cabecera *c_types.h.*
- Líneas 124-127, definiciones de valores ocupados en este archivo.
- Líneas 213-228, declaración de variables que serán ocupadas en este archivo.
- Líneas 244-256, creación de función interna *mdns_length* para obtener la longitud del *hostname* almacenado en un *buffer* con los terminadores 0x00 o 0xC0.
- Líneas 286-300, modificación de la función interna *mdns:compare_name* para no distinguir entre mayúsculas y minúsculas con la finalidad de evitar duplicidad en nombres reclamados.
- Líneas 325, 357-375, 404, 422-430, 438-445, 474-481, 539, 554-561, se modificó la función interna *mdns_answer* para responder únicamente con el *hostname* o con uno de los dos servicios anunciado. También se hicieron modificaciones para implementar en la respuesta el TTL personalizado.
- Líneas 687, 719-726, 752-759, 764, 815, 817-834, 841-854, 856-901, 903, 926, 928, 941-949,996, se modificó la función interna *mdns_send_service* para enviar la respuesta de uno de los dos servicios que se pueden anunciar.
- Líneas 1052-1141, creación de la función interna *mdns_query_hostname* utilizada para la creación de la consulta MDNS, en el formato correcto, para obtener la IP de *hostname* y realizar su envió al grupo MDNS.
- Líneas 1147-1208, modificación de la función *mdns_close* para eliminar la información almacenada de los servicios y *hostname.*
- Líneas 1210-1219, modificación de la función *mdns_disable* para desarmar los temporizadores creados.
- Se modificó la función interna *mdns_recv* que se llama cada vez que arriva un paquete. A continuación, se detallan las modificaciones de esta función
    - Líneas 1229-1232,1234,1235,1238, declaración de variables utilizadas en esta función.
    - Líneas 1268-1361, se crearon las expresiones condicionales para la etapa de *probing* y *query.* En el caso de la etapa de *probing,* al cumplirse la

expresión condicional, significa que otro *hostname* ya ha sido reclamado el nombre que se desea, por lo que notifica al usuario mediante la interfaz de comunicación de este problema y deshabilita la API MDNS. En el caso de la etapa *query,* al cumplirse la expresión condicional, se procede a obtener del paquete de respuesta la IPV4 y TTL del *hostname* consultado. Estos datos son enviados a través de la interfaz de comunicación.

  - Líneas 1362-1425, se modificó para analizar y responder a todas las consultas que puedan estar en un solo paquete de consulta con o sin método de compresión (originalmente sólo se respondía a la primera de las consultas contenidas en el paquete). Se modificaron y crearon expresiones condicionales para responder con el *hostname* y los servicios anunciados, uno de los dos o ambos servicios anunciados o únicamente el *hostname,* conservando en éstos el ID del paquete (anteriormente siempre se asignaba el ID cero).

- Líneas 1437-1452, se creó la función interna mdns*_query_func,* la cual lleva el control del tiempo de la consulta de la IP de un *hostname.* Si después de 4 segundos no se ha obtenido respuesta, notifica a través de la interfaz de comunicación de esta situación y detiene la espera de la respuesta.
- Líneas 1454-1476, se creó la función *mdns_make_query* para iniciar con el proceso de la consulta de la IO de un *hostname.*
- Línea 1490, se modificó la función *enable_mdns* para usar la nueva variable de control MDNS.
- Líneas 1500-1521, se creó la función *mdns_good_bye* utilizada para notificar a los demás participantes del grupo MDNS de la salida del módulo con la finalidad de que éstos borren de su memoria caché la existencia de éste.
- Líneas 1541-1571, se modificó la función *mdns_set_hostname* para establecer el formato del *hostname* para las etapas *probing* y *announcing.*
- Líneas 1586-1601, se creó la función *mdns_set_servername2* para establecer el nombre del segundo servicio.
- Líneas 1603-1617, se creó la función *mdns_enable_service* para habilitar el anuncio y uso de los servicios.
- Líneas 1673-1688, se creó a función *mdns_set_ttl* para establecer un TTL personalizado. Este será aplicado a los servicios como al *hostname.*
- Líneas 1690-1738, se modificó la función interna *mdns_reg* para realizar las etapas *probing* y *announcing.*
- Líneas 1748-1750, 1761-1838,1885,1889-1893, se modificó la función *mdns_init* para anunciar hasta dos servicios o únicamente el *hostname* que se desea reclamar.

Para que estos cambios tengan efecto se debe compilar la biblioteca *lwip.* Esta compilación se realiza en el *ESP8266 Toolkit,* cuyo proceso de instalación se explica en *3.3ESP8266 Toolkit* en [2]. Sin embargo, esta biblioteca presenta 2 errores que impiden su compilación, el primero de ellos es el mostrado en la figura C.6.

```
In file included from sntp.c:54:0:
../../include/lwip/app/time.h:43:6: error: conflicting types for 'gettimeofday'
 int  gettimeofday(struct timeval* t, void* timezone);
      ^
In file included from /opt/xtensa-lx106-elf/lib/gcc/xtensa-lx106-elf/4.8.2/include/string.h:10:0,
                 from ../../../include/osapi.h:28,
                 from ../../include/arch/cc.h:40,
                 from ../../include/lwip/arch.h:43,
                 from ../../include/lwip/debug.h:35,
                 from ../../include/lwip/opt.h:46,
                 from ../../include/lwip/sntp.h:4,
                 from sntp.c:45:
/opt/xtensa-lx106-elf/lib/gcc/xtensa-lx106-elf/4.8.2/include/sys/time.h:73:5: note: previous declaration of 'g
ettimeofday' was here
 int _EXFUN(gettimeofday, (struct timeval *__p, struct timezone *__z));
     ^
```

Figura C.6 *Primer error en biblioteca LWIP.*

El primero soluciona comentando la función *gettimeofday* y escribiendo el prototipo de la función *extern int _EXFUN(gettimeofday, (struct timeval *__p, struct timezone *__z)),* tal y como aparece en la figura C.7, en el archivo *time.h* que está ubicado en *ESP8266_NONOS_SDK/third_party/include/lwip/app.*

```
43   extern int _EXFUN(gettimeofday, (struct timeval *__p, struct timezone *__z));
44   //int  gettimeofday(struct timeval* t, void* timezone);
```

Figura C.7 *Modificación de archivo time.h.*

El segundo error de compilación se muestra en la figura C.9

```
sntp.c:342:1: error: unknown type name '__tzrule_type'
 __tzrule_type sntp__tzrule[2];
 ^
```

Figura C.9 *Segundo error de compilación LWIP.*

Este error se soluciona creando la estructura *_tzrule_struct* que se muestra en la figura C.8, en el archivo *sntp.c,* en la línea 259, que está ubicado en *ESP8266_NONOS_SDK/third_party/lwip/core.*

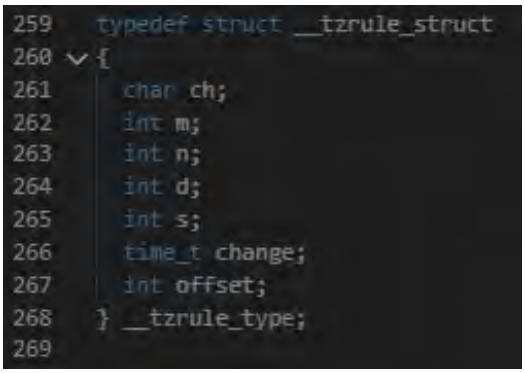


Figura C.8 *Estructura _tzrule_struct.*

Los pasos por seguir para realizar la compilación de la biblioteca LWIP son:

- Iniciar Linux (ESP8266 *Toolkit*).
- Correr la LXTerminal que está en el escritorio de la máquina virtual.
- Ejecutar *./mount.sh* e ingresar la contraseña: *espressif.*
- Utilizar el comando *cd* para acceder a la carpeta *third_party* contenida en el SDK. (*cd Share/(Nombre de la carpeta que contiene el SDK)/ third_party*).
- Ejecutar el comando *make_lib.sh lwip.*

Si la compilación fue exitosa, mostrará el mensaje de la figura C.10.

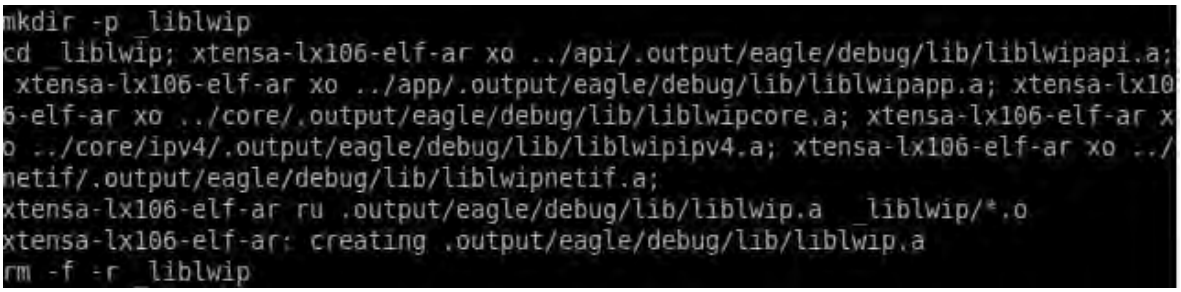


Figura C.10 *Compilación exitosa de LWIP.*

e) Modificaciones para añadir *upgrade server evento por hardware.*

Para habilitar la creación del denominado *upgrade_server,* que permite la actualización del programa de un microcontrolador PIC, se realizaron las modificaciones que se listan a continuación:

- Líneas 31-33, definición de variables.
- Línea 46, definición del prototipo de la función *server_upgrade*.
- Líneas 200-224, función *check_gpio* que es la función *callback* de un temporizador creado dentro la función *user_init.* Esta función es llamada 10 veces cada 10 *ms* tras el arranque del módulo y se comprueba el estado del GPIO2. Si el GPIO2 se

mantiene en estado alto, el módulo trabajará normalmente, en caso contrario se ejecutará la función *Server_upgrade* para iniciar el proceso de actualizar el programa de un microcontrolador PIC.

Dentro de la función *user_init* se modificaron las siguientes líneas:

- Líneas 239-243, Configuración del GPIO0 como salida en estado alto y configuración del GPIO2 como entrada con un resistor interno en configuración *pull-up.* Estas modificaciones también son necesarias para realizar el *upgrade_server* mediante *software.*
- Líneas 244-246, establece el valor inicial de la variable de control del temporizador y se lanza el temporizador que llamará cada 10 *ms* a la función *check_gpio.*

C.1.2 Modificaciones para SDK NONOS V3.0.3

a) Añadir APIs espnow y pwm

Para utilizar las funciones de las APIs *espnow y pwm* se debe modificar el archivo *makefile* que está contenido en la carpeta del proyecto de comandos AT, agregando *"-lespnow" y "-lpwm"* a la lista de bibliotecas de dicho archivo, tal y como se muestra en la figura C.3

b) Habilitar funciones criptográficas, ciphersuites y modificaciones para uso de fingerprint

Para habilitar el uso de ciertos algoritmos de cifrado y herramientas, se modificó el archivo *config_esp.h* que se encuentra en ESP8266_NONOS_SDK/third_party/include/mbedtls. Las modificaciones que se realizaron son:

- Descomentar la línea 326 del código para habilitar *Cipher Feedback mode* (CFB).
- Descomentar las líneas 420, 422, 429, 431 para habilitar las siguientes curvas elípticas:
    - SECP192R1
    - SECP256R1
    - BP384R1
    - CURVE25519
- Descomentar la línea 522 del código para habilitar los modos de cifrado basados en ECDHE-PSK en SSL / TLS, añadiendo los siguientes *ciphersuites*
    - mbedtls_tls_ecdhe_psk_with_aes_256_cbc_sha384
    - mbedtls_tls_ecdhe_psk_with_aes_256_cbc_sha
    - mbedtls_tls_ecdhe_psk_with_camellia_256_cbc_sha384
    - mbedtls_tls_ecdhe_psk_with_aes_128_cbc_sha256
    - mbedtls_tls_ecdhe_psk_with_aes_128_cbc_sha

    - mbedtls_tls_ecdhe_psk_with_camellia_128_cbc_sha256
    - mbedtls_tls_ecdhe_psk_with_3des_ede_cbc_sha
    - mbedtls_tls_ecdhe_psk_with_rc4_128_sha
- Descomentar la línea 626 del código para habilitar los modos de cifrado basados en ECDHE-RSA en SSL / TLS, añadiendo los siguientes *ciphersuites:*
    - mbedtls_tls_ecdhe_rsa_with_aes_256_gcm_sha384
    - mbedtls_tls_ecdhe_rsa_with_aes_256_cbc_sha384
    - mbedtls_tls_ecdhe_rsa_with_aes_256_cbc_sha
    - mbedtls_tls_ecdhe_rsa_with_camellia_256_gcm_sha384
    - mbedtls_tls_ecdhe_rsa_with_camellia_256_cbc_sha384
    - mbedtls_tls_ecdhe_rsa_with_aes_128_gcm_sha256
    - mbedtls_tls_ecdhe_rsa_with_aes_128_cbc_sha256
    - mbedtls_tls_ecdhe_rsa_with_aes_128_cbc_sha
    - mbedtls_tls_ecdhe_rsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_ecdhe_rsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_ecdhe_rsa_with_3des_ede_cbc_sha
    - mbedtls_tls_ecdhe_rsa_with_rc4_128_sha
- Comentar la línea 917 del código.
- Descomentar la línea 1464 del código para habilitar ARC4, añadiendo los siguientes *ciphersuites:*
    - mbedtls_tls_ecdh_ecdsa_with_rc4_128_sha
    - mbedtls_tls_ecdh_rsa_with_rc4_128_sha
    - mbedtls_tls_ecdhe_ecdsa_with_rc4_128_sha
    - mbedtls_tls_ecdhe_rsa_with_rc4_128_sha
    - mbedtls_tls_ecdhe_psk_with_rc4_128_sha
    - mbedtls_tls_dhe_psk_with_rc4_128_sha
    - mbedtls_tls_rsa_with_rc4_128_sha
    - mbedtls_tls_rsa_with_rc4_128_md5
    - mbedtls_tls_rsa_psk_with_rc4_128_sha
    - mbedtls_tls_psk_with_rc4_128_sha
- Descomentar la línea 1529 del código para habilitar *blowfish.*
- Descomentar la línea 1584 del código para habilitar *camellia*, añadiendo los siguientes *ciphersuites:*
    - mbedtls_tls_ecdh_ecdsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_ecdh_ecdsa_with_camellia_256_cbc_sha384
    - mbedtls_tls_ecdh_rsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_ecdh_rsa_with_camellia_256_cbc_sha384
    - mbedtls_tls_ecdh_ecdsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_ecdh_ecdsa_with_camellia_256_gcm_sha384

    - mbedtls_tls_ecdh_rsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_ecdh_rsa_with_camellia_256_gcm_sha384
    - mbedtls_tls_ecdhe_ecdsa_with_camellia_256_gcm_sha384
    - mbedtls_tls_ecdhe_rsa_with_camellia_256_gcm_sha384
    - mbedtls_tls_dhe_rsa_with_camellia_256_gcm_sha384
    - mbedtls_tls_ecdhe_ecdsa_with_camellia_256_cbc_sha384
    - mbedtls_tls_ecdhe_rsa_with_camellia_256_cbc_sha384
    - mbedtls_tls_dhe_rsa_with_camellia_256_cbc_sha256
    - mbedtls_tls_dhe_rsa_with_camellia_256_cbc_sha
    - mbedtls_tls_ecdhe_ecdsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_ecdhe_rsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_dhe_rsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_ecdhe_ecdsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_ecdhe_rsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_dhe_rsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_dhe_rsa_with_camellia_128_cbc_sha
    - mbedtls_tls_dhe_psk_with_camellia_256_gcm_sha384
    - mbedtls_tls_ecdhe_psk_with_camellia_256_cbc_sha384
    - mbedtls_tls_dhe_psk_with_camellia_256_cbc_sha384
    - mbedtls_tls_dhe_psk_with_camellia_128_gcm_sha256
    - mbedtls_tls_dhe_psk_with_camellia_128_cbc_sha256
    - mbedtls_tls_ecdhe_psk_with_camellia_128_cbc_sha256
    - mbedtls_tls_rsa_with_camellia_256_gcm_sha384
    - mbedtls_tls_rsa_with_camellia_256_cbc_sha256
    - mbedtls_tls_rsa_with_camellia_256_cbc_sha
    - mbedtls_tls_rsa_with_camellia_128_gcm_sha256
    - mbedtls_tls_rsa_with_camellia_128_cbc_sha256
    - mbedtls_tls_rsa_with_camellia_128_cbc_sha
    - mbedtls_tls_rsa_psk_with_camellia_256_gcm_sha384
    - mbedtls_tls_rsa_psk_with_camellia_256_cbc_sha384
    - mbedtls_tls_rsa_psk_with_camellia_128_gcm_sha256
    - mbedtls_tls_rsa_psk_with_camellia_128_cbc_sha256
    - mbedtls_tls_psk_with_camellia_256_gcm_sha384
    - mbedtls_tls_psk_with_camellia_256_cbc_sha384
    - mbedtls_tls_psk_with_camellia_128_gcm_sha256
    - mbedtls_tls_psk_with_camellia_128_cbc_sha256
- Descomentar la línea 1706 del código para habilitar la *biblioteca elliptic curve Diffie-Hellman*
- Descomentar la línea 1721 del código para habilitar la biblioteca *elliptic curve DSA*.

- Descomentar la línea 1754 del código para habilitar la biblioteca *elliptic curve over GF(p)*
- Descomentar la línea 1867 del código para habilitar MD4

Las modificaciones para hacer uso del *fingerprint* de un certificado y la forma para compilar la biblioteca *mbedtls* para que los cambios mencionados anteriormente sean habilitados, se realiza de la misma manera que como se mencionó en C.1.1c).

c) Modificaciones a los archivos de la API MDNS.

Las modificaciones y la forma de compilar la biblioteca *lwip* para que estos sean aplicados son los que se mencionan en C.1.1d)

d) Modificaciones para añadir *upgrade server evento por hardware.*

Son los mismos cambios que se mencionan en C.1.1e), aunque como dicho archivo puede contener más líneas por los comandos que se crearon para esta versión, se debe observar la posición en el código en lugar del número de línea que ocupan las modificaciones.

C.2 Modificaciones del *firmware* basado en ESP-IDF

A continuación, se detallan las modificaciones realizadas que permitieron de ciertos comandos y herramientas en la biblioteca ESP para ambos SoC.

C.2.1 Modificación *upgrade_server* por hardware

Para habilitar la creación del denominado *upgrade_server,* que permite la actualización del programa de un microcontrolador PIC por un evento de *hardware*, se realizaron modificaciones en el archivo *at_uart_task.c,* el cual se encuentra ubicado en *home/esp/esp-at/main/interface/uart.* Las modificaciones se listan a continuación:

- Línea 42, se añade *#include user_def.h*
- Líneas 43-44, definición de variables.
- Línea 45, definición del prototipo de la función *upgrade_pic_function*.
- Líneas 610-638, función *check_gpio* que es la función *callback* de un temporizador creado dentro la función *at_custom_init.* Esta función es llamada 10 veces cada 10 *ms* tras el arranque del módulo y se comprueba el estado del UPGRADE_GPIO2. Si el UPGRADE_GPIO2 se mantiene en estado alto, el módulo trabajará normalmente, en caso contrario se ejecutará la función *upgrade_pic_function* para iniciar el proceso de actualizar el programa de un microcontrolador PIC.

Dentro de la función *at_custom_init* se modificaron las siguientes líneas:

- Líneas 643-657, configuración del UPGRADE_GPIO1 como salida en estado alto y configuración del UPGRADE_GPIO2 como entrada con un resistor interno en configuración *pull-up.* Estas modificaciones también son necesarias para realizar el *upgrade_server* mediante *software.*
- Líneas 642, 658, 659, Establece el valor inicial de la variable de control del temporizador e inicia el temporizador que llamará cada 10 *ms* a la función *check_gpio.*

Como este *firmware* de comandos AT está siendo constantemente actualizado, se recomienda realizar estas modificaciones en las nuevas versiones del archivo en lugar de simplemente reemplazarlo.

C.2.2 Habilitación de métodos de encriptación

El proceso que se explicará aquí considera la utilización de una computadora con sistema operativo Linux o mediante una máquina virtual con Linux.

Como el *firmware* de comandos AT está basado en ESP-IDF, se debe completar la instalación del entorno de compilación para ESP-IDF tal y como se especifica en *ESP_AT_Get_started.md* en la sección *compiling and flashing the project* en [3]. La habilitación de las herramientas necesarias para este *firmware* se realiza durante el proceso de compilación del *firmware,* por lo que se deberá abrir la terminal e ingresar hasta el directorio *home/esp/esp_at,* una vez ahí, se ejecutarán los siguientes comandos:

- *rm sdkconfig*
- *make defconfig*
- *make menuconfig*

Al ejecutar este último comando se mostrará la interfaz de la figura C.11. En la interfaz se deberá dirigir a *Component config->mbedTLS-> Symmectric Ciphers* y habilitar los algoritmos Camellia, RC4, DES, Blowfish y XTEA que son ocupados para el desarrollo de los comandos de encriptación, tal y como se ve en la figura C.12

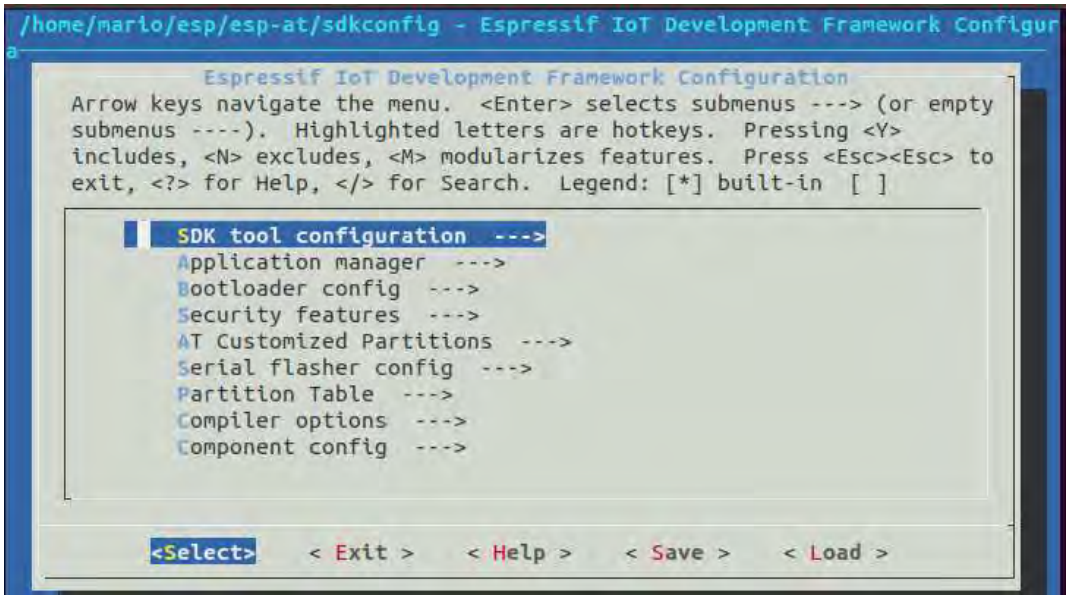


Figura C.11 *Interfaz de configuración ESP-IDF.*

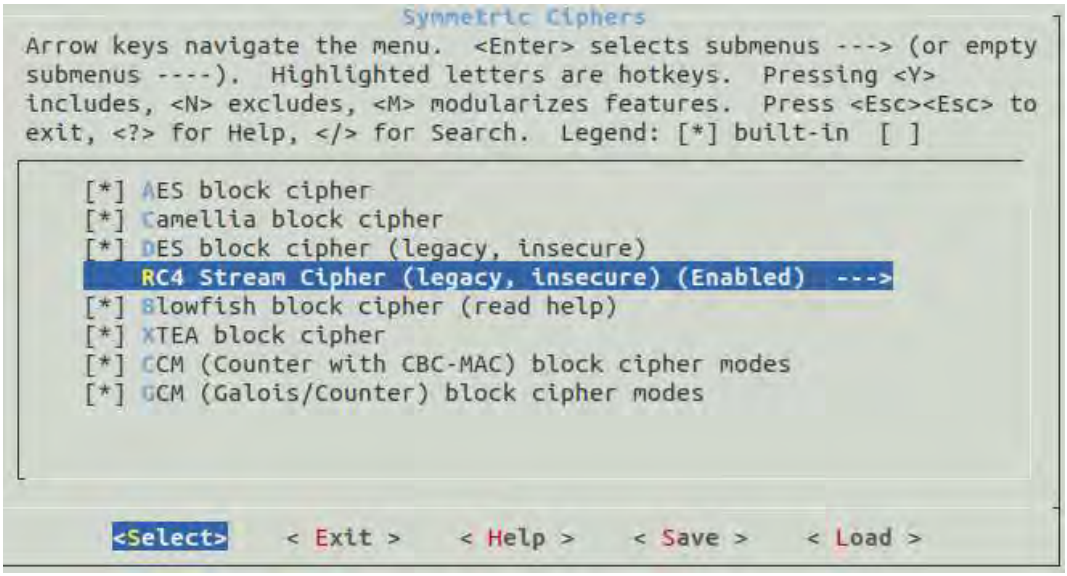


Figura C.12 *Algoritmos habilitados ESP-IDF.*

También, para la biblioteca ESP, se deberá ingresar a la sección de *component config* de la figura C.11, ingresar a AT y deshabilitar AT MQTT, AT HTTP y AT ETHERNET. Así mismo en *mdns* se establecerá dos como el número de servicios. Para el SoC ESP8266 se deberá ingresar a *phy* y se establecerá como 33 a *vd33_const_value.* Se deberán guardan los cambios, salir de esta interfaz y proseguir con el proceso de compilación del *firmware.*

C.3 Referencias

1. Espressif Systems. *ESP8266 NONOS SDK MBEDTLS 20160718*. SDK & DEMOS [Citado 2019; Disponible en:https://www.espressif.com/en/support/download/sdks-demos.
2. Espressif Systems, *ESP8266 SDK Getting Started Guide Version 3.2*. 2019.
3. Espressif Systems. *AT application for ESP32 ESP-IDF & ESP8266 ESP8266_RTOS_SDK*. [Citado 2020 Febrero]; Disponible en:https://github.com/espressif/esp-at.

Apéndice D

Descripción de funciones de la biblioteca ESP

D.1 Generales

- La biblioteca utiliza la directiva #device PASS_STRINGS= IN_RAM por lo que el usuario puede utilizar un *string* como parámetro de las funciones.
- Las funciones con sobrecargas que se diferencíen únicamente por agregar más parámetros serán consideradas como funciones con parámetros extras.
- Las funciones con parámetros con valores por defecto serán consideradas como funciones con parámetros extras.
- En las funciones que cuentan con parámetros extras, éstos estarán indicados entre corchetes. Por ejemplo:

  uint8 suma(uint8 a, uint8 b, [uint8 c])
- Las funciones fueron desarrolladas con base en un sistema de niveles para evitar un erróneo orden de llamado de éstas.
- El nivel principal es utilizado para establecer el orden de llamado de las funciones principales de todas las aplicaciones desarrolladas, el cual comprende:
  - ESP [0-4]
- Se desarrollaron niveles secundarios para ciertas aplicaciones, ya que el llamado de algunas de sus funciones no afecta el comportamiento general de la biblioteca, pero si requieren de un orden de llamado. Estos niveles secundarios son:
  - RSSI [0-4]

- ESPNOW [0-1]
- SIMPLE PAIR [0-1]
- MDNS [0-1]
- GPIO [0-2]
- PMW [0-1]
- SNTP [0-1]

D.2 Funciones generales

Estas funciones pueden ser utilizadas sin tener que habilitar ninguna aplicación en particular.

D.2.1 get_str_length

Firmware	**Cualquiera**
Función	Obtiene la longitud de un string
Nivel de ejecución	Cualquiera
Prototipo	uint16 get_str_length(uint8 *str)
Parámetros	uint8 *str: string
Retorno	Longitud del string
Nota	Si la longitud del arreglo es mayor a la definición MAX_LENGTH_TO_SEND (2048 bytes), la función retornará 0

D.2.2 compare_string

Firmware	**Cualquiera**
Función	Verifica que dos *string* sean iguales o no
Nivel de ejecución	Cualquiera
Prototipo	_bool compare_string(uint8 *x, uint8 *y)
Parámetros	uint8 *x: *string* a comparar uint8 *y: *string* a comparar
Retorno	0: No son iguales 1: Son iguales
Nota	La longitud máxima de los *string* a comparar es de 256

D.2.3 begin_esp

Firmware	**Cualquiera**
Función	Configura al módulo WiFi para su uso con el PIC
Nivel de ejecución	ESP 0 y pasa a ESP 1
Prototipo	uint8 begin_esp(void)
Retorno	0: Éxito De lo contrario: Error
Nota	Está función debe ser llamada antes que cualquier otra función del módulo, exceptuando las funciones generales. Si la configuración falla, ninguna función relacionada con las aplicaciones del módulo se ejecutará.

D.2.4 wifi_mode

Firmware	**Cualquiera**
Función	Establece el modo WiFi (802.11b/g/n)
Nivel de ejecución	Mayor a ESP 0
Prototipo	uint8 wifi_mode(uint8 mode)
Parámetros	uint8 mode: 0: Establece 802.11b 1: Establece 802.11g 2: Establece 802.11n (SDK NONOS) uint8 mode: 0: Establece 802.11b 1: Establece 802.11bg 2: Establece 802.11bgn (IDF)
Retorno	0: Éxito De lo contrario: Error

D.2.5 wifi_tx_power

Firmware	**Cualquiera**
Función	Establece el valor máximo de potencia RF TX
Nivel de ejecución	Mayor a ESP 0
Prototipo	uint8 wifi_tx_power(uint8 power)
Parámetros	uint8 power: Valor [0,82] Unidades 0.25 dBm (SDK NONOS) uint8 power: Valor [40,82] Unidades 0.25 dBm correspondiente a un rango de 10 dBm a 20.5 dBm (IDF)
Retorno	0: Éxito De lo contrario: Error

D.3 GPIO

- Estás funciones sólo controlan los GPIO del módulo WiFi utilizado
- No es compatible con el *Firmware* SDK NONOS V2

D.3.1 gpio_mode

Firmware	**SDK NONOS V3 y ESP-IDF**
Función	Configura el GPIO como entrada o como salida.
Nivel de ejecución	GPIO 0
Prototipo	uint8 gpio_mode(uint8 pin,_bool func_io, uint8 pull) (NONOS) uint8 gpio_mode(uint8 pin,_bool func_io, uint8 pull) (IDF)
Parámetros	uint8 pin: Número del pin _bool func_io: 0: Entrada 1: Salida uint8 pull: 0: pull-down IDF, pull up deshabilitado SDK 1: pull-up IDF, pull-up habilitado SDK 2: Ninguno IDF; No valido en SDK
Retorno	0: Éxito De lo contrario: Error
Nota	Los GPIO 0 y 2 en NONOS, y los *GPIO_UPGRADE* 1 y 2 en IDF, están bloqueados para ser utilizados en esta función y, por tanto, marcarán error.

D.3.2 gpio_ready

Firmware	**SDK NONOS V3 y ESP-IDF**
Función	Indica que se han terminado de configurar los GPIO que se ocuparán
Nivel de ejecución	GPIO 0 y pasa a GPIO 1
Prototipo	uint8 gpio_ready(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.3.3 gpio_write

Firmware	**SDK NONOS V3 y ESP-IDF**
Función	Establece en nivel de voltaje en el GPIO
Nivel de ejecución	GPIO 1
Prototipo	uint8 gpio_write(uint8 pin, _bool level)
Parámetros	uint8 pin: Número del pin _bool level: 0: Nivel bajo 1: Nivel alto
Retorno	0: Éxito De lo contrario: Error
Nota	Los GPIO 0 y 2 en NONOS, y los *GPIO_UPGRADE* 1 y 2 en IDF, están bloqueados para ser utilizados en esta función y, por tanto, marcarán error.

D.3.4 gpio_read

Firmware	**SDK NONOS V3 y ESP-IDF**
Función	Obtiene el nivel de voltaje presente en el GPIO
Nivel de ejecución	GPIO 1
Prototipo	uint8 gpio_read(uint8 pin)
Parámetros	uint8 pin: Número del pin
Retorno	0: Nivel bajo 1: Nivel alto De lo contrario: Error
Nota	Los GPIO 0 y 2 en NONOS, y los *GPIO_UPGRADE* 1 y 2 en IDF, están bloqueados para ser utilizados en esta función y, por tanto, marcarán error.

D.4 Punto de acceso

D.4.1 begin_ap

Firmware	**Cualquiera**
Función	Crea un punto de acceso
Nivel de ejecución	ESP 1 y pasa a ESP 2
Prototipo	uint8 begin_ap(uint8 ecn, uint8 *ssid[, uint8 *pwd[, uint8 ch]])
Parámetros	uint8 ecn: Método de encriptación 0: Abierta 2: WPA_PSK 3: WPA2_PSK 4: WPA_WPA2_PSK uint8*ssid: Nombre del AP, *string* [1-20] bytes, uint8*pwd: Contraseña para conectarse al AP, *string* [8-64] bytes ASCII uint8 ch: Identificador del canal [1-13]
Retorno	0: Nivel bajo 1: Nivel alto De lo contrario: Error
Nota	• Los parámetros opcionales no especificados tomarán sus valores por defecto. • La contraseña por defecto es “12345678” • El canal por defecto es 1 • Si se requiere especificar el canal en un AP abierto, se debe establecer una contraseña, sin embargo, ésta no será tomada en cuenta.

D.4.2 disconnect_station_ap

Firmware	**SDK NONOS**
Función	Desconecta una estación del AP
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 disconnect_station_ap(uint8 *mac)
Parámetros	uint8 *mac: string de la dirección MAC, en formato hexadecimal con caracteres de separación, de la estación a desconectar.
Retorno	0: Éxito De lo contrario: Error
Nota	La dirección MAC no es *case sensitive*

D.4.3 set_mac_ap

Firmware	**Cualquiera**
Función	Establece la dirección MAC del AP a crear
Nivel de ejecución	ESP 1
Prototipo	uint8 set_mac_ap(uint8 *mac)
Parámetros	uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del punto de acceso
Formato	Formato dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	Esta función debe ejecutarse antes de begin_ap La dirección MAC no es *case sensitive*

D.4.4 get_mac_ap

Firmware	**Cualquiera**
Función	Obtiene la dirección MAC del AP creado
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 get_mac_ap(uint8 *mac, uint8 mac_length)
Parámetros	uint8 *mac: Arreglo donde se almacenará la dirección MAC, en formato hexadecimal con caracteres de separación uint8 mac_length: Tamaño del arreglo donde se almacenará la dirección MAC
Formato	Formato de la dirección MAC almacenada en el arreglo: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño mínimo del arreglo para almacenar la dirección MAC es de 18 bytes

D.4.5 set_ip_ap

Firmware	**Cualquiera**
Función	Establece la dirección IP del AP creado
Nivel de ejecución	ESP 2
Prototipo	uint8 set_ip_ap(uint8 *ip)
Parámetros	uint8 *ip: *string* de la dirección IP, en formato decimal con caracteres de separación, del punto de acceso
Formato	Formato de la dirección IP: 192.168.4.1
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo de la dirección IP es de 15 bytes

D.4.6 get_ip_ap

Firmware	**Cualquiera**
Función	Obtiene la dirección IP del AP creado
Nivel de ejecución	ESP 2
Prototipo	uint8 get_ip_ap(uint8 *ip,uint8 ip_length)
Parámetros	uint8 *ip: Arreglo donde se almacenará la dirección IP, en formato decimal con caracteres de separación uint8 ip_length: Tamaño del arreglo donde se almacenará la dirección IP
Formato	Formato de la dirección IP almacenada en el arreglo: 192.168.4.1
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño mínimo del arreglo para almacenar la dirección IP es de 16 bytes

D.5 Estación

D.5.1 begin_station

Firmware	**Cualquiera**
Función	Se conecta al AP especificado
Nivel de ejecución	ESP 1 y pasa a ESP 2
Prototipo	uint8 begin_station(uint8 *name, uint8 *ssid, uint8 *pwd)
Parámetros	uint8 *name: *string* del nombre que tomará la estación. uint8 *ssid: *string* del nombre del AP al que se conectará uint8 *pwd: *string* de la contraseña del AP al que se conectará
Retorno	0: Éxito De lo contrario: Error
Nota	La sintaxis de carácter de escape es necesitada si el SSID y PWD tienen caracteres especiales

D.5.2 scan_ap_station

Firmware	**Cualquiera**
Función	En serial UART del usuario se imprimirán los nombres de todos los AP cercanos y el usuario deberá ingresar el nombre y la contraseña del AP a conectarse a través del mismo serial
Nivel de ejecución	ESP 1 y pasa a ESP 2
Prototipo	uint8 scan_ap_station(uint8 *name)
Parámetros	uint8 *name: *string* del nombre que tomará la estación.
Retorno	0: Éxito De lo contrario: Error

D.5.3 begin_strongestap_station

Firmware	**Cualquiera**
Función	Se conectará al AP con la señal más fuerte de los proporcionados
Nivel de ejecución	ESP 1 y pasa a ESP 2
Prototipo	uint8 begin_strongestap_station(uint8 *name, uint8 *ssid1, uint8 *pwd1, uint8 *ssid2, uint8 *pwd2 [, uint8 *ssid3, uint8 *pwd3])
Parámetros	uint8 *name: *string* del nombre que tomará la estación uint8 *ssid1: *string* del nombre del primer AP uint8 *pwd1: *string* de la contraseña del primer AP uint8 *ssid2: *string* del nombre del segundo AP uint8 *pwd2: *string* de la contraseña del segundo AP uint8 *ssid3: *string* del nombre del tercer AP uint8 *pwd3: *string* de la contraseña del tercer AP
Retorno	0: Éxito De lo contrario: Error

D.5.4 set_mac_station

Firmware	**Cualquiera**
Función	Establece la dirección MAC de la estación
Nivel de ejecución	ESP 1
Prototipo	uint8 set_mac_station(uint8 *mac)
Parámetros	uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, de la estación
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	La dirección MAC no es *case sensitive*

D.5.5 get_mac_station

Firmware	**Cualquiera**
Función	Obtiene la dirección MAC de la estación
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 get_mac_station(uint8 *mac,uint8 mac_length)
Parámetros	uint8 *mac: Arreglo donde se almacenará la dirección MAC uint8 mac_length: Tamaño del arreglo donde se almacenará la dirección MAC
Formato	Formato de la dirección MAC almacenada en el arreglo: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño mínimo del arreglo para almacenar la dirección MAC es de 18 bytes

D.5.6 set_ip_station

Firmware	**Cualquiera**
Función	Establece la dirección IP de la estación
Nivel de ejecución	ESP 2
Prototipo	uint8 set_ip_station(uint8 *ip)
Parámetros	uint8 *ip: *string* de la dirección IP, en formato decimal con caracteres de separación, del punto de acceso
Formato	Formato de la dirección IP: 192.168.4.1
Retorno	0: Éxito De lo contrario: Error
Nota	• Únicamente se podrá establecer la parte de la dirección IP que identifica al host, la parte que identifica a la red no podrá ser cambiada. • El tamaño máximo de la IP es de 15 bytes

D.5.7 get_ip_station

Firmware	**Cualquiera**
Función	Obtiene la dirección IP de la estación
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 get_ip_station(uint8 *ip,uint8 ip_length)
Parámetros	uint8 *ip: Arreglo donde se almacenará la dirección IP, en formato decimal con caracteres de separación uint8 ip_length: Tamaño del arreglo donde se almacenará la dirección IP
Formato	Formato de la dirección IP almacenada en el arreglo: 192.168.4.1
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño mínimo del arreglo para almacenar la dirección IP es de 16 bytes

D.5.8 disconnect_station

Firmware	**Cualquiera**
Función	Se desconecta del AP al cual está conectado
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 1
Prototipo	uint8 disconnect_station(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.5.9 enable_active_packet

Firmware	**Cualquiera**
Función	Comienza la emisión de paquetes *probe request*
Nivel de ejecución	ESP 2 y 3
Prototipo	uint8 enable_active_packet(uint32 probe_time)
Parámetros	uint32 probe_time: Tiempo en milisegundos entre la emisión de cada paquete *probe request*. [5,6000000]
Retorno	0: Éxito De lo contrario: Error
Nota	En IDF elegir un tiempo más pequeño que en SDK NONOS

D.5.10 disable_active_packet

Firmware	**Cualquiera**
Función	Detiene la emisión de paquetes *probe request*
Nivel de ejecución	ESP 2 y 3
Prototipo	uint8 disable_active_packet(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.5.11 up_level2_to_level3

Consultar D.25.1 up_level2_to_level3

D.5.12 up_to_level2

Consultar D.28.1 up_to_level2

D.6 DNS

- Algunas de estas funciones requieren de acceso a internet.
- El módulo WiFi debe ser configurado como estación

D.6.1 enable_dns

Firmware	**Cualquiera**
Función	Configura el módulo WiFi para el uso de las funciones DNS
Nivel de ejecución	ESP 1
Prototipo	uint8 disconnect_station(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.6.2 request_dns

Firmware	**Cualquiera**
Función	Obtiene la dirección IP del nombre de dominio dado.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 request_dns(uint8 *name,uint8 *ip, uint8 buf_len)
Parámetros	uint8 *name:*string* del nombre de dominio a obtener su dirección IP. uint8 *ip: Arreglo donde se almacenará la dirección IP, en formato decimal con caracteres de separación uint8 buf_len: Tamaño del arreglo donde se almacenará la dirección IP
Retorno	0: Éxito De lo contrario: Error
Nota	• En algún punto del programa se debió haber llamado previamente la función enable_dns • El tamaño mínimo del arreglo para almacenar la dirección IP es de 16 bytes • Esta función requiere de acceso a internet

D.7 Funciones generales de cliente

- Estas funciones se pueden utilizar en los clientes TCP, UDP y SSL.
- Algunas de estas funciones son utilizadas por el primer servidor.

D.7.1 set_buffer_client

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado por los clientes TCP, UDP, HTTP y SSL para almacenar los mensajes recibidos
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_client(uint8 *buffer, uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más dos bytes extras que son ocupados internamente (los últimos 2). • El tamaño mínimo del buffer es de 3 bytes • El buffer es compartido entre todos los tipos de clientes

D.7.2 Compare_Cbuffer

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido del buffer que fue establecido con set_buffer_client
Nivel de ejecución	Cualquiera
Prototipo	_bool compare_Cbuffer(uint8 *str[,_bool is_new])
Parámetros	uint8 *str: *string* que se comparará _bool is_new: 0: Comparará cada vez que la función es llamada 1: Comparará únicamente cuando existe nuevos datos en el buffer de los clientes y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia

D.7.3 send_client (uniconexión)

Firmware	**Cualquiera**
Función	Envía datos al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3
Macros	send_udp send_tcp send_ssl
Prototipo	uint8 send_client(uint8 *message) uint8 send_udp(uint8 *message) uint8 send_tcp(uint8 *message) uint8 send_ssl(uint8 *message)
Parámetros	uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados. • Las macros proporcionadas tienen el objetivo de dar claridad a la documentación de los programas

D.7.4 send_client (multiconexión)

Firmware	**Cualquiera**
Función	Envía datos al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3
Macros	send_udp send_tcp send_ssl send_server1 send_request_part_http send_request_tcp_part_http send_request_ssl_part_http
Prototipo	uint8 send_client(uint8 channel, uint8 *message) uint8 send_udp(uint8 channel, uint8 *message) uint8 send_tcp(uint8 channel, uint8 *message) uint8 send_ssl(uint8 channel, uint8 *message) uint8 send_server1(uint8 channel, uint8 *message) uint8 send_request_part_http(uint8 channel, uint8 *message) uint8 send_request_tcp_part_http(uint8 channel, uint8 *message) uint8 send_request_ssl_part_http(uint8 channel, uint8 *message)
Parámetros	uint8 channel: ID asignado al cliente uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados. • Las macros proporcionadas tienen el objetivo de dar claridad a la documentación de los programas • Aplicable a conexiones persistentes

D.7.5 send_bytes_client (uniconexión)

Firmware	**Cualquiera**
Función	Envía datos al servidor al cual el cliente está conectado (arreglos de bytes, se puede enviar el valor \0)
Nivel de ejecución	ESP 3
Macros	send_bytes_udp send_bytes_tcp send_bytes_ssl
Prototipo	uint8 send_bytes_client(uint8 *message, uint16 length) uint8 send_ bytes_udp(uint8 *message, uint16 length) uint8 send_ bytes_tcp(uint8 *message, uint16 length) uint8 send_ bytes_ssl(uint8 *message, uint16 length)
Parámetros	uint8 *message: Arreglo bytes que contiene los datos a enviar uint16 length: Longitud del arreglo de a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados. • Las macros proporcionadas tienen el objetivo de dar claridad a la documentación de los programas

D.7.6 send_bytes_client (multiconexión)

Firmware	**Cualquiera**
Función	Envía datos al servidor al cual el cliente está conectado (arreglos de bytes, se puede enviar el carácter \0)
Nivel de ejecución	ESP 3
Macros	send_udp send_tcp send_ssl send_server1
Prototipo	uint8 send_ bytes_client(uint8 channel, uint8 *message, uint16 length) uint8 send_ bytes_udp(uint8 channel, uint8 *message, uint16 length) uint8 send_ bytes_tcp(uint8 channel, uint8 *message, uint16 length) uint8 send_ bytes_ssl(uint8 channel, uint8 *message, uint16 length) uint8 send_ bytes_server1(uint8 channel, uint8 *message, uint16 length)
Parámetros	uint8 channel: ID asignado al cliente uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados. • Las macros proporcionadas tienen el objetivo de dar claridad a la documentación de los programas • Aplicable a conexiones persistentes

D.7.7 send_client_const (uniconexión)

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3

Macro	#define send_client_const(m)
	{
	  presend_client_const(sizeof(m)-1);
	  sending_client_const(m);
	}

Macros	send_client_tcp_const send_client_ssl_const send_client_udp_const
Prototipo	send_client_const(m) send_client_tcp_const(m) send_client_ssl_const(m) send_client_udp_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno
Nota	Aplicable a conexiones persistentes

D.7.8 ret_send_client_const (uniconexión)

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3

Macro	#define ret_send_client_const(m,r)
	{
	presend_client_const(sizeof(m)-1);
	r=sending_client_const(m);
	}

Macros	ret_send_client_tcp_const ret_send_client_ssl_const ret_send_client_udp_const
Prototipo	ret_send_client_const(m,r) ret_send_client_tcp_const(m,r) ret_send_client_ssl_const(m,r) ret_send_client_udp_const(m,r)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno
Nota	Aplicable a conexiones persistentes

D.7.9 presend_client_const (uniconexión)

Firmware	**Cualquiera**
Función	Establece el número de bytes a enviar
Nivel de ejecución	Cualquiera
Prototipo	void presend_client_const(uint16 message_length)
Parámetros	uint16 message_length: Número de bytes a enviar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_client_const

D.7.10 sending_client_const (uniconexión)

Firmware	**Cualquiera**
Función	Envía el número de bytes establecido en presend_client_const al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3
Prototipo	uint8 sending_client_const(uint8 message)
Parámetros	uint8 message_length: Arreglo almacenado en la memoria ROM que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función se debió haber llamado previamente presend_client_const

D.7.11 send_mclient_const (multiconexión)

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3

Macro	#define send_mclient_const(c,m)
	{
	  presend_mclient_const(c,sizeof(m)-1);
	  sending_mclient_const(m);
	}

Macros	send_mclient_tcp_const send_mclient_ssl_const send_mclient_udp_const send_mclient_server1_const
Prototipo	send_mclient_const(c,m) send_mclient_tcp_const(c,m) send_mclient_ssl_const(c,m) send_mclient_udp_const(c,m) send_mclient_server1_const(c,m)
Parámetros	c: ID asignado al cliente m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno
Nota	Aplicable a conexiones persistentes

D.7.12 ret_send_mclient_const (multiconexión)

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3

Macro	#define ret_send_mclient_const(c,m,r)
	{
	  presend_mclient_const(c,sizeof(m)-1);
	  r=sending_mclient_const(m);
	}

Macros	ret_send_mclient_tcp_const ret_send_mclient_ssl_const ret_send_mclient_udp_const ret_send_mclient_server1_const
Prototipo	ret_send_mclient_const(c,m,r) ret_send_mclient_tcp_const(c,m,r) ret_send_mclient_ssl_const(c,m,r) ret_send_mclient_udp_const(c,m,r) ret_send_mclient_server1_const(c,m,r)
Parámetros	c: ID asignado al cliente m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno
Nota	Aplicable a conexiones persistentes

D.7.13 resend_mclient_const

Firmware	**Cualquiera**
Función	Establece el canal y el número de bytes a enviar
Nivel de ejecución	Cualquiera
Prototipo	void presend_mclient_const(uint8 channel, uint16 message_length)
Parámetros	uint8 channel: ID asignado al cliente uint16 message_length: Número de bytes a enviar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_mclient_const

D.7.14 sending_mclient_const

Firmware	**Cualquiera**
Función	Envía el número de bytes establecido en presend_mclient_const al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3
Prototipo	uint8 sending_mclient_const(uint8 message)
Parámetros	uint8 message_length: Arreglo almacenado en la memoria ROM que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función se debió haber llamado previamente presend_mclient_const

D.7.15 delete_client (uniconexión)

Firmware	**Cualquiera**
Función	Desconecta al cliente del servidor al cual está conectado
Nivel de ejecución	ESP 3 y pasa a ESP 2
Macros	delete_tcp_client delete_ssl_client delete_udp_client
Prototipo	uint8 delete_client(void) uint8 delete_tcp_client(void) uint8 delete_udp_client(void) uint8 delete_ssl_client(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.7.16 delete_client (multiconexión)

Firmware	**Cualquiera**
Función	Desconecta al cliente del servidor al cual está conectado
Nivel de ejecución	ESP 3 y pasa a ESP 2*
Macros	delete_tcp_client delete_ssl_client delete_udp_client close_client_server1
Prototipo	uint8 delete_client(uint8 channel) uint8 delete_tcp_client(uint8 channel) uint8 delete_udp_client(uint8 channel) uint8 delete_ssl_client(uint8 channel) uint8 close_client_server1(uint8 channel)
Parámetros	uint8 channel: ID del cliente a desconectar [0-4]
Retorno	0: Éxito De lo contrario: Error
Nota	Pasa a nivel 2 siempre y cuando no exista otro canal abierto y no se esté ejecutando un servidor

D.8 Cliente SSL

D.8.1 enable_fingerprint_ssl

Firmware	**SDK NONOS**
Función	Habilita y establece el *fingerprint* utilizado para autentificar el servidor al cual se conectará.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 enable_fingerprint_ssl(uint8 *fingerprint)
Parámetros	uint8 *fingerprint: *String* del *fingerprint*
Formato	Formato del *fingerprint* (SHA1): 4e92bde81ab27334ec2c69b9b083ef7e422a5c71
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño del *fingerprint* es el que determina el método que se aplicará al certificado recibido desde el servidor durante la negociación. String de 32 bytes será MD5 String de 40 bytes será SHA1 String de 64 bytes será SHA256 Debido a que cada 2 bytes forman un número hexadecimal

D.8.2 disable_fingerprint_ssl

Firmware	**SDK NONOS**
Función	Deshabilita el uso del *fingerprint* para autentificar el servidor durante la conexión.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 disable_fingerprint_ssl(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.8.3 create_ssl_pclient (uniconexión)

Firmware	**Cualquiera**
Función	Crea un único cliente SSL persistente
Nivel de ejecución	ESP 2 y pasa a ESP 3
Macro	create_ssl_http_client
Prototipo	uint8 create_ssl_pclient(uint8 *server, uint16 port) uint8 create_ssl_http_client(uint8 *server, uint16 port)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor
Retorno	0: Éxito De lo contrario: Error
Nota	No usar esta función si el servidor no mantiene la conexión activa.

D.8.4 create_ssl_pclient (multiconexión)

Firmware	**Cualquiera**
Función	Crea un único cliente SSL persistente
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Macro	create_ssl_http_client
Prototipo	uint8 create_ssl_pclient(uint8 *server, uint16 port) uint8 create_ssl_http_client(uint8 *server, uint16 port)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor
Retorno	0: Éxito De lo contrario: Error
Nota	No usar esta función si el servidor no mantiene la conexión activa.

D.8.5 send_ssl (uniconexión)

Consultar D.7.3 send_client (uniconexión)

D.8.6 send_ssl (multiconexión)

Consultar D.7.4 send_client (multiconexión)

D.8.7 send_bytes_ssl (uniconexión)

Consultar D.7.5 send_bytes_client (uniconexión)

D.8.8 send__bytes_ssl (multiconexión)

Consultar D.7.6 send_bytes_client (multiconexión)

D.8.9 send_client_ssl_const

Consultar D.7.7 send_client_const (uniconexión)

D.8.10 ret_send_client_ssl_const

Consultar D.7.8 ret_send_client_const (uniconexión)

D.8.11 send_mclient_ssl_const

Consultar D.7.11 send_mclient_const (multiconexión)

D.8.12 ret_send_mclient_ssl_const

Consultar D.7.12 ret_send_mclient_const (multiconexión)

D.8.13 nsend_ssl_client

Firmware	**Cualquiera**
Función	Se conecta y envía datos a un servidor con una conexión no persistente.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 nsend_ssl_client(uint8 *server, uint16 port, uint8 *message)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• Esta función debe utilizarse cuando el *timeout* del servidor es muy pequeño • El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.8.14 delete_ssl_client (uniconexión)

Consultar D.7.15 delete_client (uniconexión)

D.8.15 delete_ssl_client (multiconexión)

Consultar D.7.16 delete_client (multiconexión)

D.9 Cliente TCP

D.9.1 create_tcp_pclient (uniconexión)

Firmware	**Cualquiera**
Función	Crea un único cliente TCP persistente
Nivel de ejecución	ESP 2 y pasa a ESP 3
Macro	create_tcp_http_client
Prototipo	uint8 create_tcp_pclient(uint8 *server, uint16 port) uint8 create_tcp_http_client(uint8 *server, uint16 port)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor
Retorno	0: Éxito De lo contrario: Error
Nota	No usar esta función si el servidor no mantiene la conexión activa.

D.9.2 create_tcp_pclient

Firmware	**Cualquiera**
Función	Crea un único cliente TCP persistente
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Macro	create_tcp_http_client
Prototipo	uint8 create_tcp_pclient(uint8 *server, uint16 port) uint8 create_tcp_http_client(uint8 *server, uint16 port)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor
Retorno	0: Éxito De lo contrario: Error
Nota	No usar esta función si el servidor no mantiene la conexión activa.

D.9.3 send_tcp (uniconexión)

Consultar D.7.3 send_client (uniconexión)

D.9.4 send_tcp (multiconexión)

Consultar D.7.4 send_client (multiconexión)

D.9.5 send_bytes_tcp (uniconexión)

Consultar D.7.5 send_bytes_client (uniconexión)

D.9.6 send__bytes_tcp (multiconexión)

Consultar D.7.6 send_bytes_client (multiconexión)

D.9.7 send_client_tcp_const

Consultar D.7.7 send_client_const (uniconexión)

D.9.8 ret_send_client_tcp_const

Consultar D.7.8 ret_send_client_const (uniconexión)

D.9.9 send_mclient_tcp_const

Consultar D.7.11 send_mclient_const (multiconexión)

D.9.10 ret_send_mclient_tcp_const

Consultar D.7.12 ret_send_mclient_const (multiconexión)

D.9.11 nsend_tcp_client

Firmware	**Cualquiera**
Función	Se conecta y envía datos a un servidor con una conexión no persistente.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 nsend_tcp_client(uint8 *server, uint16 port, uint8 *message)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• Esta función debe utilizarse cuando el *timeout* del servidor es muy pequeño • El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.9.12 delete_tcp_client (uniconexión)

Consultar D.7.15 delete_client (uniconexión)

D.9.13 delete_tcp_client (multiconexión)

Consultar D.7.16 delete_client (multiconexión)

D.10 Cliente UDP

D.10.1 create_udp_client (uniconexión)

Firmware	**Cualquiera**
Función	Crea un único cliente UDP
Nivel de ejecución	ESP 2 y pasa a ESP 3
Prototipo	uint8 create_udp_client(uint8 *server, uint16 remote_port,uint16 local_port,uint8 mode)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 remote_port: Puerto habilitado por el servidor uint16 local_port: Puerto habilitado por el cliente uint8 mode: Funcionamiento del cliente 0: La pareja destino del cliente no cambiará 1: La pareja destino del cliente puede cambiar una vez 2: La pareja destino del cliente puede cambiar
Retorno	0: Éxito De lo contrario: Error

D.10.2 send_udp (uniconexión)

Consultar D.7.3 send_client (uniconexión)

D.10.3 send_udp (multiconexión)

Consultar D.7.4 send_client (multiconexión)

D.10.4 send_bytes_udp (uniconexión)

Consultar D.7.5 send_bytes_client (uniconexión)

D.10.5 send__bytes_udp (multiconexión)

Consultar D.7.6 send_bytes_client (multiconexión)

D.10.6 send_client_udp_const

Consultar D.7.7 send_client_const (uniconexión)

D.10.7 ret_send_client_udp_const

Consultar D.7.8 ret_send_client_const (uniconexión)

D.10.8 send_mclient_udp_const

Consultar D.7.11 send_mclient_const (multiconexión)

D.10.9 ret_send_mclient_udp_const

Consultar D.7.12 ret_send_mclient_const (multiconexión)

D.10.10 delete_udp_client (uniconexión)

Consultar D.7.15 delete_client (uniconexión)

D.10.11 delete_udp_client (multiconexión)

Consultar D.7.16 delete_client (multiconexión)

D.11 Cliente transparente

D.11.1 set_buffer_tclient

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado por el cliente transparente
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_tclient(uint8 *buffer, uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más dos bytes extras que son ocupados internamente (los últimos 2). • El tamaño mínimo del buffer es de 3 bytes

D.11.2 compare_TCbuffer

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido del buffer que fue establecido con set_buffer_tclient
Nivel de ejecución	Cualquiera
Prototipo	_bool compare_TCbuffer(uint8 *str)
Parámetros	uint8 *str: *string* que se comparará
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia

D.11.3 create_tclient

Firmware	**Cualquiera**
Función	Crea un cliente TCP o UDP transparente
Nivel de ejecución	ESP 2 y pasa a ESP 4
Prototipo	uint8 create_tclient(_bool mode,uint8 *server, uint16 remote_port, [uint16 local_port])
Parámetros	_bool mode: Tipo de cliente 0: TCP 1: UDP uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 remote_port: Puerto habilitado por el servidor uint16 local_port: Puerto habilitado por el cliente (Sólo en UDP)
Retorno	0: Éxito De lo contrario: Error

D.11.4 send_tclient

Firmware	**Cualquiera**
Función	Envía datos al servidor al cual el cliente está conectado
Nivel de ejecución	ESP 4
Prototipo	uint8 send_tclient(uint8 *message)
Parámetros	uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.11.5 delete_tclient

Firmware	**Cualquiera**
Función	Desconecta al cliente del servidor al cual está conectado
Nivel de ejecución	ESP 4 y pasa a ESP 2
Prototipo	uint8 delete_tclient(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.11.6 fail_delete_tclient

Firmware	**Cualquiera**
Función	Elimina los restos de la configuración establecida en create_tclient
Nivel de ejecución	ESP 2
Prototipo	uint8 fail_delete_tclient(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Esta función sólo debe ser llamada en caso de que la función delete_tclient devuelva un error de tipo RET_FAIL_TRANS_DISC

D.12 Primer Servidor

- El primer servidor por su naturaleza implica el uso de multiconexiones

D.12.1 set_buffer_server1

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado por el primer servidor
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_server1(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más dos bytes extras que son ocupados internamente (los últimos 2). • El tamaño mínimo del buffer es de 3 bytes

D.12.2 compare_S1buffer

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido del buffer que fue establecido con set_buffer_server1
Nivel de ejecución	ESP 3
Prototipo	_bool compare_S1buffer(uint8 *str,[_bool is_new])
Parámetros	uint8 *str: *string* que se comparará _bool is_new: 0: Comparará cada vez que la función es llamada 1: Comparará únicamente cuando existe nuevos datos en el buffer de los clientes y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia

D.12.3 create_server1

Firmware	Cualquiera
Función	Crea un servidor TCP o SSL en el puerto especificado
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Prototipo	uint8 create_server1(uint16 port, uint8 max_conn, uint16 timeout, bool ssl)
Parámetros	uint16 port: Puerto que habilitará el servidor uint8 max_conn: Número máximo de clientes que podrán conectarse al servidor [1-5] uint16 timeout: Tiempo en segundos que el servidor mantendrá conectado a un cliente inactivo [0-7200] _bool ssl: (IDF) 0: Crea un servidor TCP 1: Crea un servidor SSL
Retorno	0: Éxito De lo contrario: Error
Nota	• Si el *timeout* es cero, el servidor no desconectará al cliente, y por tanto, deberá ser el cliente el que cierre la conexión. • Si el segundo servidor es activado el número máximo de conexiones serán 2.

D.12.4 send_server1

Consultar D.7.4 send_client (multiconexión)

D.12.5 send_bytes_server1

Consultar D.7.6 send_bytes_client (multiconexión)

D.12.6 send_server1_const

Consultar D.7.11 send_mclient_const (multiconexión)

D.12.7 ret_send_server1_const

Consultar D.7.12 ret_send_mclient_const (multiconexión)

D.12.8 close_client_server1

Consultar D.7.16 delete_client (multiconexión)

D.12.9 delete_server1

Firmware	**Cualquiera**
Función	Elimina el servidor TCP creado
Nivel de ejecución	ESP 3 y pasa a ESP 2*
Prototipo	uint8 delete_server1(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	• Para ejecutar esta función, el servidor no debe tener conexiones abiertas • *En caso de que esté activo el segundo servidor, servidor SSL o el módulo WiFi esté conectado como cliente a un servidor, se mantendrá en nivel ESP 3

D.12.10 client_connected

Firmware	**Cualquiera**
Función	Indica si está conectado un cliente en el canal especificado o si ya está conectado por lo menos un cliente en algún canal.
Nivel de ejecución	Cualquiera
Prototipo	_bool client_connected(uint8 channel)
Parámetros	uint8 channel: 0: Indica si está conectado un cliente en el canal 0. 1: Indica si está conectado un cliente en el canal 1. 2: Indica si está conectado un cliente en el canal 2. 3: Indica si está conectado un cliente en el canal 3. 4: Indica si está conectado un cliente en el canal 4. 5: Indica si está conectado al menos un cliente en todos los canales.
Retorno	1: Cliente conectado 0: Cliente no conectado

D.13 Cliente SNTP

- Algunas de estas funciones requieren internet

D.13.1 set_sntp_client

Firmware	**Cualquiera**
Función	Establece la zona horaria de acuerdo con el tiempo universal coordinado UTC.
Nivel de ejecución	ESP 2 o 3, y SNTP 0
Prototipo	uint8 set_sntp(sint8 time_zone)
Parámetros	sint8 time_zone: zona horaria a consultar hora y fecha [-11,13]
Retorno	0: Éxito De lo contrario: Error

D.13.2 get_sntp_client

Firmware	**Cualquiera**
Función	Consulta hora y fecha de la zona horaria establecida.
Nivel de ejecución	ESP 2 o 3 y SNTP 1
Prototipo	uint8 get_sntp(uint8 *date, uint8 length)
Parámetros	uint8 *date: Arreglo donde se almacenará la fecha y hora uint8 length: Tamaño del arreglo
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño mínimo del arreglo es de 25 bytes. • Esta función requiere de acceso a internet

D.14 Smart config

- Requiere el uso de la aplicación EspTouch desarrollada por Espressif Systems o una similar desarrollada por el usuario.

D.14.1 run_smart_station

Firmware	**Cualquiera**
Función	Realiza la conexión a un punto de acceso. El SSID y la contraseña son suministrados desde una aplicación mediante la tecnología ESP-TOUCH
Nivel de ejecución	ESP 1 y pasa a ESP2
Prototipo	uint8 run_smart_station(uint8 *name)
Parámetros	uint8 *name: *String* que contiene el nombre que tomará la estación al conectarse
Retorno	0: Éxito De lo contrario: Error

D.15 Servidor HTTP

- En esta sección están incluidas las funciones para el servidor WEB y las de servicio REST

D.15.1 create_http_server

Firmware	**Cualquiera**
Función	Crea un servidor HTTP en el puerto especificado
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Prototipo	uint8 create_http_server(uint16 port,uint8 max_conn, uint16 timeout,uint8 *resource, , _bool ssl=0)
Parámetros	uint16 port: Puerto que habilitará el servidor uint8 max_conn: Número máximo de clientes que podrán conectarse al servidor [1-5] uint16 timeout: Tiempo en segundos que el servidor mantendrá conectado a un cliente inactivo [0-7200] uint8 *resource: Recurso que se está atendiendo. _bool ssl: 0: Conexiones TCP 1: Conexiones SSL/TLS
Retorno	0: Éxito De lo contrario: Error
Nota	• Si el timeout es cero, el servidor no desconectará al cliente, y por tanto, deberá ser el cliente el que cierre la conexión. • Si el segundo servidor es activado el número máximo de conexiones serán 2.

D.15.2 http_send

Firmware	**Cualquiera**
Función	Envía datos al cliente HTTP que está siendo atendido.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_send(uint8 *message)
Parámetros	uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.15.3 http_send_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al cliente HTTP que está siendo atendido
Nivel de ejecución	ESP 3

Macro	#define send_http_const(m)
	{
	presend_http_const((sizeof(m)-1));
	sending_http_const(m);
	}

Prototipo	send_http_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno

D.15.4 presend_http_const

Firmware	**Cualquiera**
Función	Captura la longitud del arreglo constante que contiene los datos a enviar
Nivel de ejecución	ESP 3
Prototipo	void presend_http_const(uint16 message_length)
Parámetros	uint16 message_length: Número de bytes a enviar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_http_const

D.15.5 sending_http_const

Firmware	Cualquiera
Función	Envía la cantidad de bytes establecido en presend_http_const a un cliente HTTP.
Nivel de ejecución	ESP 3
Prototipo	uint8 sending_http_const(uint8 message)
Parámetros	uint8 message: Arreglo almacenado en la memoria ROM que contiene los datos a enviar.
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función se debió haber llamado previamente presend_http_const

D.15.6 http_finish

Firmware	Cualquiera
Función	Finaliza la comunicación con el cliente HTTP que está siendo atendido.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_finish(_bool waiting_favicon)
Parámetros	_bool waiting_favicon: 0: No espera atención a petición del favicon 1: Espera atención a petición del favicon
Retorno	0: Éxito De lo contrario: Error

D.15.7 http_default_resource

Firmware	Cualquiera
Función	Atiende peticiones a recursos no declarados.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_default_resource(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Se utiliza en el caso *default* de una estructura de control *switch* que atiende las peticiones HTTP.

D.15.8 http_upgrade_resource

Firmware	**Cualquiera**
Función	Actualiza el valor del recurso que se está atendiendo.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_upgrade_resource(void)
Parámetros	Ninguno
Retorno	Ninguno

D.15.9 http_response_put

Firmware	**Cualquiera**
Función	Responde y finaliza una petición del método PUT.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_response_put(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Debe ser usado solamente para los recursos que esperan el método PUT.

D.15.10 http_response_get

Firmware	**Cualquiera**
Función	Responde y finaliza una petición del método GET.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_response_get(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	La respuesta no tiene contenido. Debe ser usado solamente para los recursos que esperan el método GET.

D.15.11 http_response_json(uint8 *json)

Firmware	**Cualquiera**
Función	Responde incluyendo un JSON y finaliza una petición del método GET.
Nivel de ejecución	ESP 3
Prototipo	uint8 http_response_json(uint8 *json)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Es declarado solamente para los recursos que esperan devolver un JSON.

D.16 Segundo servidor TCP

D.16.1 create_server2

Firmware	**Cualquiera**
Función	Crea un segundo servidor TCP en el puerto especificado
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Prototipo	int8 internal_create_server2(uint16 port, uint8 max_conn, uint16 timeout) (SDK NONOS) uint8 internal_create_server2(uint16 port, uint16 timeout) (IDF)
Parámetros	uint16 port: Puerto que habilitará el servidor [1- 65535] uint8 max_conn: Número máximo de clientes que podrán conectarse al servidor [1-2] uint16 timeout: Tiempo en segundos que el servidor mantendrá conectado a un cliente inactivo [0-7200]
Retorno	0: Éxito De lo contrario: Error
Nota	• Si el timeout es cero, el servidor no desconectará al cliente, y por tanto, deberá ser el cliente el que cierre la conexión. • En SDK NONOS el servidor permite hasta dos clientes, en IDF sólo un cliente • Los clientes tomarán los ID 2 y 3.

D.16.2 delete_server2

Firmware	**Cualquiera**
Función	Elimina el segundo servidor TCP creado
Nivel de ejecución	ESP 3 y pasa a ESP 2*
Prototipo	uint8 delete_server2(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función, el servidor no debe tener conexiones abiertas *En caso de que esté activo el primer servidor o el módulo WiFi esté conectado como cliente a un servidor, se mantendrá en nivel ESP 3

D.16.3 set_buffer_server2

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado por el segundo servidor
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_server2(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más dos bytes extras que son ocupados internamente (los últimos 2). • El tamaño mínimo del buffer es de 3 bytes

D.16.4 compare_S2buffer

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido del buffer que fue establecido con set_buffer_server2
Nivel de ejecución	Cualquiera
Prototipo	_bool compare_S1buffer(uint8 *str,[_bool is_new])
Parámetros	uint8 *str: *string* que se comparará _bool is_new: 0: Comparará cada vez que la función es llamada 1: Comparará únicamente cuando existe nuevos datos en el buffer de los clientes y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia

D.16.5 send_server2

Firmware	**Cualquiera**
Función	Envía datos del segundo servidor al cliente especificado
Nivel de ejecución	ESP 3
Prototipo	uint8 send_server2(uint8 channel, uint8 *message) (SDK NONOS) uint8 send_server2(uint8 *message) (IDF)
Parámetros	uint8 channel: ID asignado al cliente uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.16.6 send_server2_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM del segundo servidor al cliente especificado
Nivel de ejecución	ESP 3

Macro	#define send_server2_const(m) (IDF)
	{
	  presend_server2_const(sizeof(m)-1);
	  sending_server2_const(m);
	}
	#define send_server2_const(c,m) (NONOS SDK)
	{
	  presend_server2_const(c, sizeof(m)-1);
	  sending_server2_const(m);
	}

Prototipo	send_server2_const(m) (IDF) send_server2_const(c,m) (NONOS SDK)
Parámetros	c: ID asignado al cliente m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno

D.16.7 ret_send_server2_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM del segundo servidor al cliente especificado
Nivel de ejecución	ESP 3

Macro	#define ret_send_server2_const(m,r) (IDF)
	{
	  presend_server2_const(sizeof(m)-1);
	  r=sending_server2_const(m);
	}
	#define ret_send_server2_const(c,m,r) (NONOS SDK)
	{
	  presend_server2_const(c, sizeof(m)-1);
	  r=sending_server2_const(m);
	}

Prototipo	send_server2_const(m,r) (IDF) send_server2_const(c,m,r) (NONOS SDK)
Parámetros	c: ID asignado al cliente m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno

D.16.8 presend_server2_const

Firmware	**Cualquiera**
Función	Establece el canal y el número de bytes a enviar
Nivel de ejecución	Cualquiera
Prototipo	void presend_server2_const(uint8 channel, uint16 message_length) (SDK NONOS) void presend_server2_const(uint16 message_length) (IDF)
Parámetros	uint8 channel: ID asignado al cliente uint16 message_length: Número de bytes a enviar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_server2_const

D.16.9 sending_server2_const

Firmware	**Cualquiera**
Función	Envía el número de bytes establecido en presend_server2_const del segundo servidor al cliente especificado
Nivel de ejecución	ESP 3
Prototipo	uint8 sending_server2_const(uint8 message)
Parámetros	uint8 message_length: Arreglo almacenado en la memoria ROM que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función se debió haber llamado previamente presend_server2_const

D.16.10 close_server2

Firmware	**Cualquiera**
Función	Cierra la conexión con el cliente especificado
Nivel de ejecución	ESP 3
Prototipo	uint8 close_server2(uint8 id)
Parámetros	uint8 id: ID del cliente a desconectar [2,3,5]
Retorno	0: Éxito De lo contrario: Error
Nota	Cuando el ID == 5, se cierran ambos clientes

D.16.11 client_connected

Consultar D.12.9 client_connected

D.17 Servidor SSL

D.17.1 create_server_ssl

Firmware	**SDK NONOS**
Función	Crea un servidor SSL en el puerto especificado
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Prototipo	uint8 create_server_ssl(uint16 port, uint16 timeout)
Parámetros	uint16 port: Puerto que habilitará el servidor [1- 65535] uint16 timeout: Tiempo en segundos que el servidor mantendrá conectado a un cliente inactivo [0-7200]
Retorno	0: Éxito De lo contrario: Error
Nota	Si el timeout es cero, el servidor no desconectará al cliente, y por tanto, deberá ser el cliente el que cierre la conexión. Este servidor sólo permite un cliente

D.17.2 delete_server_ssl

Firmware	**SDK NONOS**
Función	Elimina el servidor SSL creado
Nivel de ejecución	ESP 3 y pasa a ESP 2*
Prototipo	uint8 delete_server_ssl(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	• Para ejecutar esta función, el servidor no debe tener conexiones abiertas • *En caso de que esté activo el primer servidor o el módulo WiFi esté actuando como cliente, se mantendrá en nivel ESP 3

D.17.3 set_buffer_ssl_server

Firmware	**SDK NONOS**
Función	Establece el buffer que será utilizado por el servidor SSL
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_ssl_server(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará los datos recibidos uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más dos bytes extras que son ocupados internamente (los últimos 2). • El tamaño mínimo del buffer es de 3 bytes

D.17.4 send_server_ssl

Firmware	**SDK NONOS V2 y V3**
Función	Envía datos al cliente que está conectado
Nivel de ejecución	ESP 3
Prototipo	uint8 send_server_ssl(uint8 *message)
Parámetros	uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.17.5 send_server_ssl_const

Firmware	**SDK NONOS V2 y V3**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al cliente que está conectado
Nivel de ejecución	ESP 3

Macro	#define send_server_ssl_const(m)
	{
	presend_server_ssl_const(sizeof(m)-1);
	sending_server_ssl_const(m);
	}

Prototipo	send_server_ssl_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno

D.17.6 ret_send_server_ssl_const

Firmware	**SDK NONOS V2 y V3**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al cliente que está conectado
Nivel de ejecución	ESP 3

Macro	#define ret_send_server_ssl_const(m,r)
	{
	presend_server_ssl_const(sizeof(m)-1);
	r=sending_server_ssl_const(m);
	}

Prototipo	ret_send_server_ssl_const(m,r)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno

D.17.7 compare_SSLbuffer

Firmware	**SDK NONOS V2 y V3**
Función	Compara un *string* con el contenido del buffer que fue establecido con set_buffer_ssl_server
Nivel de ejecución	Cualquiera
Prototipo	_bool compare_S1buffer(uint8 *str,[_bool is_new])
Parámetros	uint8 *str: *string* que se comparará _bool is_new: 0: Comparará cada vez que la función es llamada 1: Comparará únicamente cuando existe nuevos datos en el buffer de los clientes y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia

D.17.8 presend_server_ssl_const

Firmware	**SDK NONOS**
Función	Establece el número de bytes a enviar
Nivel de ejecución	Cualquiera
Prototipo	void presend_server_ssl_const(uint16 message_length)
Parámetros	uint16 message_length: Número de bytes a enviar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_server_ssl_const

D.17.9 sending_server_ssl_const

Firmware	**SDK NONOS**
Función	Envía el número de bytes establecido en presend_server_ssl_constal servidor al cual el cliente está conectado
Nivel de ejecución	ESP 3
Prototipo	uint8 sending_server_ssl_const(uint8 message)
Parámetros	uint8 message: Arreglo almacenado en la memoria ROM que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función se debió haber llamado previamente presend_server_ssl_const

D.17.10 close_server_ssl

Firmware	**Cualquiera**
Función	Cierra la conexión con el cliente especificado
Nivel de ejecución	ESP 3
Prototipo	uint8 close_server_ssl
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.18 JSON

- Estas funciones solo están disponibles para cliente HTTP, servidor HTTP y Websocket

D.18.1 set_http_cjsonbuffer

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado para almacenar los mensajes JSON recibidos para clientes HTTP.
Nivel de ejecución	ESP 2
Prototipo	uint8 set_http_cjsonbuffer(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	Es utilizada en cliente HTTP El tamaño del buffer debe ser igual al tamaño deseado más un byte extra. El tamaño mínimo del buffer es de 3 bytes El buffer es compartido entre todos los clientes HTTP.

D.18.2 set_http_sjsonbuffer

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado para almacenar los mensajes JSON recibidos para el servidor HTTP.
Nivel de ejecución	ESP 2
Prototipo	uint8 set_http_sjsonbuffer(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• Es utilizada en servidor HTTP • El tamaño del buffer debe ser igual al tamaño deseado más un byte extra. • El tamaño mínimo del buffer es de 3 bytes

D.18.3 set_ws_jsonbuffer

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado para almacenar los mensajes JSON recibidos para el servidor websocket.
Nivel de ejecución	ESP 2
Prototipo	uint8 set_ws_jsonbuffer(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• Es utilizada en Websocket • El tamaño del buffer debe ser igual al tamaño deseado más un byte extra. • El tamaño mínimo del buffer es de 3 bytes

D.18.4 compare_http_sjson

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido de una clave del buffer de servidor HTTP para mensajes JSON
Nivel de ejecución	ESP 3
Prototipo	_bool compare_http_sjson(uint8 *key[,uint8 *value,_bool is_new])
Parámetros	uint8 *key: *string* que se comparará uint8 *value: *string* que se comparará _bool is_new: 0: Compara cada vez que la función es llamada 1: Compara únicamente cuando existe nuevos datos en el buffer del servidor http y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia
Nota	Es utilizada en servidor HTTP

D.18.5 compare_http_cjson

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido de una clave del buffer de clientes HTTP para mensajes JSON
Nivel de ejecución	ESP 3
Prototipo	_bool compare_http_cjson(uint8 *key[,uint8 *value,_bool is_new])
Parámetros	uint8 *key: *string* que se comparará uint8 *value: *string* que se comparará _bool is_new: 0: Comparará cada vez que la función es llamada 1: Comparará únicamente cuando existe nuevos datos en el buffer del servidor http y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia
Nota	Es utilizada en cliente HTTP

D.18.6 compare_wsjson

Firmware	**Cualquiera**
Función	Compara un *string* con el contenido de una clave del buffer del Websocket para mensajes JSON
Nivel de ejecución	ESP 3
Prototipo	_bool compare_wsjson (uint8 *key[,uint8 *value,_bool is_new])
Parámetros	uint8 *key: *string* que se comparará uint8 *value: *string* que se comparará _bool is_new: 0: Comparará cada vez que la función es llamada 1: Comparará únicamente cuando existe nuevos datos en el buffer del servidor http y no se ha presentado previamente una coincidencia
Retorno	0: No hay coincidencia o no hay un nuevo mensaje 1: Coincidencia
Nota	Es utilizada en Websocket

D.18.7 get_value_http_cjson

Firmware	**Cualquiera**
Función	Obtiene el contenido de una clave en el buffer de mensaje JSON de los clientes HTTP
Nivel de ejecución	ESP 3
Prototipo	_bool get_value_http_cjson(uint8 *key,uint8 *value_buf,uint8 value_buf_length)
Parámetros	uint8 *key: *string* que se comparará uint8 * value_buf: *string* que se comparará uint8 value_buf_length:
Retorno	0: Éxito De lo contrario: Error
Nota	Es utilizada en cliente HTTP

D.18.8 get_value_http_sjson

Firmware	**Cualquiera**
Función	Obtiene el contenido de una clave en el buffer de mensaje JSON del servidor HTTP
Nivel de ejecución	ESP 3
Prototipo	_bool get_value_http_sjson(uint8 *key,uint8 *value_buf,uint8 value_buf_length)
Parámetros	uint8 *key: *string* que se comparará uint8 * value_buf: *string* que se comparará uint8 value_buf_length:
Retorno	0: Éxito De lo contrario: Error
Nota	Es utilizada en servidor HTTP

D.18.9 get_value_wsjson

Firmware	**Cualquiera**
Función	Obtiene el contenido de una clave en el buffer de mensaje JSON del Websocket
Nivel de ejecución	ESP 3
Prototipo	_bool get_value_wsjson (uint8 *key,uint8 *value_buf,uint8 value_buf_length)
Parámetros	uint8 *key: *string* que se comparará uint8 * value_buf: *string* que se comparará uint8 value_buf_length:
Retorno	0: Éxito De lo contrario: Error
Nota	Es utilizada en Websocket

D.19 Cliente HTTP

D.19.1 request_http_client

Firmware	**Cualquiera**
Función	Envía una petición completa a un servidor mediante un método HTTP.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint16 request_http_client(uint8 *server, uint8 request_methods, uint8 *uri, uint8 channel[, uint16 port[, uint8 *json[, uint8 *authorization]]])
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor. uint8 request_methods: Método de la petición HTTP. 1: GET 2: PUT 3: POST 4: DELETE uint8 *uri: Arreglo que contiene la URI del recurso al cual se realiza la petición. uint8 channel: ID asignado al cliente. uint16 port: Puerto habilitado por el servidor. uint8 *json: Arreglo el cual contiene un JSON. uint8 *authorization: Arreglo que contiene autorización básica ("Username:Password")
Retorno	<100: Error >99: Códigos de estado HTTP
Nota	El puerto utilizado por defecto es el número 80

D.19.2 finish_request_http_client

Firmware	**Cualquiera**
Función	Envía la última parte de una petición HTTP, obteniendo el código de estado HTTP, si es el caso almacena el mensaje JSON en buffer de clientes http.
Nivel de ejecución	ESP 3
Prototipo	uint16 finish_request_http_client(uint8 channel, uint8 *message);
Parámetros	uint8 channel: ID asignado al cliente. uint8 *message: Arreglo que contiene la última parte de un petición HTTP.
Retorno	<100: Error >99: Códigos de estado HTTP

D.19.3 create_ssl_http_client

Consultar D.8.4 create_ssl_pclient (multiconexión)

D.19.4 create_tcp_http_client

Consultar D.9.2 create_tcp_pclient

D.19.5 delete_ssl_http_client

Consultar D.7.16 delete_client (multiconexión)

D.19.6 delete_tcp_http_client

Consultar D.7.16 delete_client (multiconexión)

D.19.7 send_request_part_http

Consultar D.7.4 send_client (multiconexión)

D.19.8 send_request_tcp_part_http

Consultar D.7.4 send_client (multiconexión)

D.19.9 send_request_ssl_part_http

Consultar D.7.4 send_client (multiconexión)

D.20 Websocket

- No se puede utilizar con servidor SSL
- Para ambos *Firmware* el funcionamiento es el mismo, la única diferencia es que en IDF no se puede desplazar la sesión actual con una nueva ventana, porque solo cuenta con un canal, así que es necesario cerrar la sesión, antes de abrirla de nuevo en otra pestaña.

D.20.1 create_Websocket_server

Firmware	**Cualquiera**
Función	Crea un Websocket en el puerto especificado
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Prototipo	uint8 create_Websocket_server(uint16 port)
Parámetros	uint16 port: Puerto que habilitará el Websocket
Retorno	0: Éxito De lo contrario: Error

D.20.2 delete_Websocket_server

Firmware	**Cualquiera**
Función	Elimina el Websocket creado.
Nivel de ejecución	ESP 3 y pasa a ESP 2*
Prototipo	uint8 delete_Websocket_server(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	* Si no hay más servidores y/o clientes activos pasa a ESP2

D.20.3 request_connection_Websocket

Firmware	**Cualquiera**
Función	Ejecuta la conexión a un cliente Websocket
Nivel de ejecución	ESP 3
Prototipo	uint8 request_connection_Websocket(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error o no existe una solicitud de conexión activa.

D.20.4 send_binary_Websocket

Firmware	**Cualquiera**
Función	Envía mensajes tipo binarios (bytes) al cliente Websocket
Nivel de ejecución	ESP 3
Prototipo	uint8 send_binary_Websocket(uint8 *payload, uint16 length) send_binary_Websocket(uint8 *payload, uint16 length, uint8 in_byte)
Parámetros	uint8 *payload: Arreglo que contiene la carga útil a enviar uint16 length: Longitud del arreglo de bytes a enviar uint8 in_byte: Primer byte insertado en el mensaje.
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.20.5 send_text_Websocket

Firmware	**Cualquiera**
Función	Envía mensajes tipo texto al cliente Websocket.
Nivel de ejecución	ESP 3
Prototipo	uint8 send_binary_Websocket(uint8 *payload)
Parámetros	uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.20.6 send_text_Websocket_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar mensajes tipo texto almacenados en la ROM al cliente del Websocket
Nivel de ejecución	ESP 3

Macro	#define send_text_Websocket_const(m)
	{
	presend_Websocket_const(0,(sizeof(m)-1),0,0);
	sending_Websocket_const(m);
	}

Prototipo	send_text_Websocket_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno

D.20.7 send_binary_Websocket_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar mensajes tipo binarios (bytes) almacenados en la ROM al cliente del Websocket
Nivel de ejecución	ESP 3

Macro	#define send_binary_Websocket_const(m)
	{
	presend_Websocket_const(1,(sizeof(m)-1),0,0);
	sending_Websocket_const(m);
	}

Prototipo	send_binary_Websocket_const(m))
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno

D.20.8 send_binary _insert_Websocket_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar mensajes tipo binarios (bytes) almacenados en la ROM al cliente del Websocket, incrusta un byte inicial
Nivel de ejecución	ESP 3

Macro	#define send_binary_insert_Websocket_const(m)
	{
	presend_Websocket_const(1,(sizeof(m)-1),1,b);
	sending_Websocket_const(b);
	}

Prototipo	send_binary_insert_Websocket_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar b: Byte insertado
Retorno	Ninguno

D.20.9 ret_send_text_Websocket_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar mensajes tipo texto almacenados en la ROM al cliente del Websocket
Nivel de ejecución	ESP 3

Macro	#define ret_send_text_Websocket_const(m,r)
	{
	presend_Websocket_const(0,(sizeof(m)-1),0,0);
	r=sending_Websocket_const(m);
	}

Prototipo	ret_send_text_Websocket_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno

D.20.10 ret_send_binary_Websocket_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar mensajes tipo binarios (bytes) almacenados en la ROM al cliente del Websocket
Nivel de ejecución	ESP 3

Macro	#define ret_send_binary_Websocket_const(m,r)
	{
	presend_Websocket_const(1,(sizeof(m)-1),0,0);
	r=sending_Websocket_const(m);
	}

Prototipo	ret_send_binary_Websocket_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno

D.20.11 ret_send_binary _insert_Websocket_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar mensajes tipo binarios (bytes) almacenados en la ROM al cliente del Websocket, incrusta un byte inicial
Nivel de ejecución	ESP 3

Macro	#define ret_send_binary_insert_Websocket_const(m,r)
	{
	presend_Websocket_const(1,(sizeof(m)-1),1,b);
	r=sending_Websocket_const(b);
	}

Prototipo	ret_send_binary_insert_Websocket_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar b: Byte insertado r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno

D.20.12 presend_Websocket_const

Firmware	**Cualquiera**
Función	Captura los parámetros para el envío de arreglos constantes para Websocket
Nivel de ejecución	ESP 3
Prototipo	void presend_Websocket_const(_bool binary,uint16 message_length,_bool e_insertbyte,uint8 in_byte)
Parámetros	_bool binary: 0: Se envía mensajes tipo texto 1: Se envía mensajes tipo binaries (bytes) uint16 message_length: Número de bytes a enviar _bool e_insertbyte: 0: No inserta byte 1: Inserta bye uint8 in_byte: Byte a insertar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_Websocket_const

D.20.13 sending_Websocket_const

Firmware	**Cualquiera**
Función	Envía la cantidad de bytes establecido en presend_Websocket_const a un cliente del Websocket
Nivel de ejecución	ESP 3
Prototipo	uint8 sending_http_const(uint8 message)
Parámetros	uint8 message: Arreglo almacenado en la memoria ROM que contiene los datos a enviar.
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función se debió haber llamado previamente presend_Websocket_const

D.21 MQTT

D.21.1 create_mqtt

Firmware	**Cualquiera**
Función	Crea un cliente TCP que se conecta a un broker MQTT
Nivel de ejecución	ESP 2 o 3 y pasa a ESP 3
Prototipo	uint8 create_mqtt(uint8 *server, uint16 port,uint8 channel, _bool ssl=0)
Parámetros	uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del bróker MQTT uint16 port: Puerto habilitado por el servidor uint8 channel: ID asignado al cliente _bool ssl: 0: Conexión TCP 1: Conexión TCP/SSL
Retorno	0: Éxito De lo contrario: Error

D.21.2 set_buffer_mqtt

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado para el cliente MQTT
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_mqtt(uint8 *buffer,uint8 buffer_length)
Parámetros	buffer -> Arreglo donde se almacenará los mensajes recibidos uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• Los mensajes almacenados en el buffer contienen cabeceras del protocolo MQTT, por lo que los mensajes no son legibles al acceder directamente. • El tamaño mínimo permitida es de 6, que es longitud necesaria para obtener mensajes de calidad 0 con una longitud de tema y mensajes de un byte. • La longitud es elegida de acuerdo a la calidad, longitud de tema y de mensajes esperados: • Para calidad 0: buffer_length=4 + longitud de tema + longitud máxima de mensaje esperado • Para calidad 1 y 2-> buffer_length=6 + longitud de tema + longitud máxima de mensaje esperado

D.21.3 connect_mqtt

Firmware	**Cualquiera**
Función	Inicia sesión del cliente MQTT
Nivel de ejecución	ESP 3
Prototipo	uint8 connect_mqtt(uint8 *ID, uint8 *user,uint8 *password, uint16 keepalive[, uint8 *will_topic[,uint8 *will_message]]);
Parámetros	uint8 *ID: Arreglo que contiene el ID de usuario (ID de dispositivo, 1 a 23 caracteres) uint8 *user: Arreglo que contiene el nombre de usuario uint8 *password: Arreglo que contiene la contraseña de usuario uint16 keepalive: Cantidad de segundos máximos que el cliente MQTT desee estar conectado al estar inactivo min-1 segundo max-65535 segundos (18 horas 12 minutos y 15 segundos) uint8 *will_topic: Arreglo que contiene el tema, al cual llegará el mensaje de última voluntad uint8 *will_message: Arreglo que contiene el mensaje de última voluntad
Retorno	0: Éxito De lo contrario: Error

D.21.4 subscribe_mqtt

Firmware	**Cualquiera**
Función	Se subscribe a un tema (topic)
Nivel de ejecución	ESP 3
Prototipo	uint8 subscribe_mqtt(uint8 *topic[,uint8 QoS])
Parámetros	uint8 *topic: Arreglo que contiene el tema a suscribir uint8 QoS: Calidad máxima en la que llegaran los mensajes para este tema 0: QoS0 1: QoS1 2: QoS2
Retorno	0: Éxito De lo contrario: Error

D.21.5 unsubscribe_mqtt

Firmware	**Cualquiera**
Función	Cancelar suscripción a cierto tema
Nivel de ejecución	ESP 3
Prototipo	uint8 unsubscribe_mqtt(uint8 *topic)
Parámetros	uint8 *topic: Arreglo que contiene el tema a darse de baja
Retorno	0: Éxito De lo contrario: Error

D.21.6 disconnect_mqtt

Firmware	**Cualquiera**
Función	Termina la sesión del cliente MQTT y desconecta del broker
Nivel de ejecución	ESP 3, si no hay más servidores y/o clientes activos pasa a ESP 2
Prototipo	uint8 disconnect_mqtt(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.21.7 pingreq_mqtt

Firmware	**Cualquiera**
Función	Envía un PING al broker para evitar ser desconectado durante inactividad
Nivel de ejecución	ESP 3
Prototipo	uint8 pingreq_mqtt(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.21.8 publishQoS0_mqtt

Firmware	**Cualquiera**
Función	Publica un mensaje para cierto tema con calidad cero
Nivel de ejecución	ESP 2 y pasa a ESP 3
Prototipo	uint8 publishQoS0_mqtt(uint8 *topic[, uint8 *message,_bool retain])
Parámetros	uint8 *topic: Arreglo que contiene el tema en que se publicará el mensaje uint8 *message: Arreglo que contiene el mensaje que se publicará ,_bool retain: 0: Mensaje no retenido 1: Mensaje retenido
Retorno	0: Éxito De lo contrario: Error

D.21.9 publishQoS1_mqtt

Firmware	**Cualquiera**
Función	Publica un mensaje para cierto tema con calidad uno
Nivel de ejecución	ESP 2 y pasa a ESP 3
Prototipo	uint8 publishQoS1_mqtt(uint8 *topic[, uint8 *message,_bool retain])
Parámetros	uint8 *topic: Arreglo que contiene el tema en que se publicará el mensaje uint8 *message: Arreglo que contiene el mensaje que se publicará ,_bool retain: 0: Mensaje no retenido 1: Mensaje retenido
Retorno	0: Éxito De lo contrario: Error

D.21.10 publishQoS2_mqtt

Firmware	**Cualquiera**
Función	Publica un mensaje para cierto tema con calidad dos
Nivel de ejecución	ESP 3
Prototipo	uint8 publishQoS2_mqtt(uint8 *topic[, uint8 *message,_bool retain])
Parámetros	uint8 *topic: Arreglo que contiene el tema en que se publicará el mensaje uint8 *message: Arreglo que contiene el mensaje que se publicará ,_bool retain: 0: Mensaje no retenido 1: Mensaje retenido
Retorno	0: Éxito De lo contrario: Error

D.21.11 refresh_mqtt

Firmware	**Cualquiera**
Función	Atiende la recepción de mensajes
Nivel de ejecución	ESP 3
Prototipo	uint8 refresh_mqtt(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.21.12 get_payload_mqtt

Firmware	**Cualquiera**
Función	Obtiene la carga útil del mensaje recibido para cierto tema
Nivel de ejecución	ESP 3
Prototipo	uint8 get_payload_mqtt(uint8 *topic,uint8 *buffer,uint8 buffer_lenght)
Parámetros	uint8 *topic: Arreglo que contiene el tema del cual se busca la carga útil del mensaje uint8 * buffer: Arreglo donde se almacenará la carga útil del mensaje uint8 buffer_lenght: Tamaño máximo del buffer.
Retorno	0: Éxito De lo contrario: Error

D.21.13 get_allpayload_mqtt

Firmware	**Cualquiera**
Función	Obtiene la carga útil del mensaje recibido para cualquier tema
Nivel de ejecución	ESP 3
Prototipo	get_allpayload_mqtt(uint8 *buffer,uint8 buffer_lenght)
Parámetros	uint8 * buffer: Arreglo donde se almacenará la carga útil del mensaje uint8 buffer_lenght: Tamaño máximo del buffer.
Retorno	0: Éxito De lo contrario: Error

D.22 SMTP

D.22.1 send_mail_smtp

Firmware	**Cualquiera**
Función	Envía un correo electrónico.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 send_mail_smtp(uint8 *server, uint16 port,uint8 channel,uint8 *from,uint8 *password,uint8 *to,uint8* subject,uint8 *data)
Parámetros	uint8 *server: Arreglo que contiene el servidor SMTP con cifrado SSL/TLS uint16 port: Puerto habilitado por el servidor uint8 channel: ID asignado al cliente uint8 *from; Arreglo que contiene la dirección de correo del emisor uint8 *password: Arreglo que contiene la contraseña de la cuenta de correo uint8 *to: Arreglo que contiene la dirección de correo del receptor uint8 *subject: Arreglo que contiene la dirección del asunto del correo uint8 *data -> Arreglo que contiene el cuerpo del correo.
Retorno	0: Éxito De lo contrario: Error
Nota	Envía un correo completo con texto plano.

D.22.2 begin_part_smtp

Firmware	**Cualquiera**
Función	Inicia un cliente SMTP para el envío de un correo electrónico.
Nivel de ejecución	ESP 2 o 3, y pasa a ESP 3
Prototipo	uint8 begin_part_smtp(uint8 *server, uint16 port,uint8 channel,uint8 *from,uint8 *password,uint8 *to);
Parámetros	uint8 *server: Arreglo que contiene el servidor SMTP con cifrado SSL/TLS uint16 port: Puerto habilitado por el servidor uint8 channel: ID asignado al cliente uint8 *from; Arreglo que contiene la dirección de correo del emisor uint8 *password: Arreglo que contiene la contraseña de la cuenta de correo uint8 *to: Arreglo que contiene la dirección de correo del receptor
Retorno	0: Éxito De lo contrario: Error
Nota	Usado para enviar un correo “personalizado” (Para correos que requieran de un formato diferente al texto plano o de mayor extensión del que permite enviar send_mail_smtp)

D.22.3 send_content_smtp

Firmware	**Cualquiera**
Función	Envía una sección del cuerpo de un correo electrónico
Nivel de ejecución	ESP 3
Prototipo	uint8 send_bodypart_smtp(uint8 *message)
Parámetros	uint8 *message: Arreglo donde se aloja una sección del cuerpo del correo.
Retorno	0: Éxito De lo contrario: Error
Nota	Usado para enviar un correo “personalizado” (Para correos que requieran de un formato diferente al texto plano o de mayor extensión del que permite enviar send_mail_smtp)

D.22.4 send_content_smtp_const

Firmware	**Cualquiera**
Descripción	Es una macro una sección del cuerpo de un correo electrónico almacenados en la ROM
Nivel de ejecución	ESP 3

Macro	#define send_bodypart_smtp_const(m)
	{
	presend_bodypart_smtp_const((sizeof(m)-1));
	sending_bodypart_smtp_const(m);
	}

Prototipo	send_bodypart_smtp_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno
Nota	Usado para enviar un correo “personalizado” (Para correos que requieran de un formato diferente al texto plano o de mayor extensión del que permite enviar send_mail_smtp)

D.22.5 ret_send_content_smtp _const

Firmware	**Cualquiera**
Descripción	Es una macro una sección del cuerpo de un correo electrónico almacenados en la ROM
Nivel de ejecución	ESP 3

Macro	#define ret_send_bodypart_smtp_const(m)
	{
	presend_bodypart_smtp_const((sizeof(m)-1));
	sending_bodypart_smtp_const(m);
	}

Prototipo	send_bodypart_smtp_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	0: Éxito De lo contrario: Error
Nota	Usado para enviar un correo “personalizado” (Para correos que requieran de un formato diferente al texto plano o de mayor extensión del que permite enviar send_mail_smtp)

D.22.6 finish_smtp

Firmware	**Cualquiera**
Función	Finaliza al cliente SMTP, terminando de enviar el correo.
Nivel de ejecución	ESP 3 y pasa a ESP 2*
Prototipo	uint8 finish_part_smtp(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	• Usado para enviar un correo “personalizado” (Para correos que requieran de un formato diferente al texto plano o de mayor extensión del que permite enviar send_mail_smtp). • * si no hay más servidores y/o clientes activos pasa a ESP 2

D.23 Google OAuth 2.0

D.23.1 get_token_oauth20

Firmware	**Cualquiera**
Función	Obtiene el access token y refresh token para acceder a las APIs de Google.
Nivel de ejecución	ESP 2
Prototipo	uint8 get_token_auth20(uint8 *token,uint16 token_length,uint8 *refresh_token,uint16 refresh_token_length)
Parámetros	uint8 token: Arreglo en el que se almacenará el access token uint16 token_length: Tamaño del arreglo del access token uint8 refresh_token: Arreglo en el que se almacenará el refreh token uint16 refresh_token_length: Tamaño del arreglo del access token
Retorno	0: Éxito De lo contrario: Error
Nota	• Necesita interacción en un hiperterminal para ingresar el code otorga Google para la autorización auth2.0. • La longitud de los arreglos del access token y refresh token son variables.

D.23.2 refresh_token_auth20

Firmware	**Cualquiera**
Función	Obtiene un nuevo access token mediante el refresh token
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 refresh_token_oauth20(uint8 channel,uint8 *token,uint8 *refresh_token)
Parámetros	uint8 channel: ID asignado al cliente uint8 token: Arreglo en el que se almacena el access token uint8 refresh_token: Arreglo en el que se almacena el refreh token
Retorno	0: Éxito De lo contrario: Error
Nota	No necesita interacción mediante una hiperterminal.

D.24 Google spreadsheets

D.24.1 get_matrix_spreadsheets

Firmware	**Cualquiera**
Función	Obtiene el access token y refresh token para acceder a las APIs de Google.
Nivel de ejecución	ESP 2
Prototipo	uint16 get_matrix_spreadsheets(uint8 channel, uint8 *message, uint8* buffer, uint8 buffer_length)
Parámetros	uint8 channel: ID asignado al cliente uint8 *message: Última parte de una petición HTTP uint8* buffer: Arreglo en cual se almacenará la matriz de valores solicitados. uint8 buffer_length: Tamaño del buffer
Retorno	<100: Error >99: Códigos de estado HTTP
Nota	Es necesario establecer un cliente HTTP para usar esta función

D.25 RSSI

D.25.1 up_level2_to_level3

Firmware	**Cualquiera**
Función	Pasa de nivel ESP 2 a ESP 3
Nivel de ejecución	ESP 2 y pasa a ESP 3
Prototipo	uint8 up_level2_to_level3(void)
Parámetros	Ninguno
Retorno	0: Exito De lo contrario: Error

D.25.2 get_rssi

Firmware	**Cualquiera**
Función	Obtiene un valor RSSI
Nivel de ejecución	ESP 2 o 3 y RSSI 0
Prototipo	sint8 get_rssi(void)
Parámetros	Ninguno
Retorno	<15: Valor RSSI De lo contrario: Error
Nota	Se necesita estar conectado a un AP

D.25.3 start_mean_rssi

Firmware	Cualquiera
Función	Comienza la obtención del RSSI promedio del número de lecturas especificadas
Nivel de ejecución	ESP 3 y RSSI 0, y pasa a RSSI 1
Prototipo	uint8 start_mean_rssi(uint16 samples,uint16 interval)
Parámetros	uint16 samples: Número de muestras para obtener el valor RSSI promedio [1-600] uint16 interval: Intervalo de tiempo en milisegundos para obtener un nuevo valor RSSI [5-65535]
Retorno	0: Exito De lo contrario: Error
Nota	• Si ocurre un error durante el proceso de obtención de valores RSSI, pasará a RSSI 0 y, por tanto, la función get_mean_rssi retornará un error de nivel • Se necesita estar conectado a un AP

D.25.4 get_mean_rssi

Firmware	Cualquiera
Función	Obtiene el valor RSSI promedio
Nivel de ejecución	RSSI 1 y pasa a RSSI 0
Prototipo	uint8 get_mean_rssi(real32 *rssi_var)
Parámetros	real32 *rssi_var: Dirección de la variable donde se almacenará el valor promedio
Retorno	0: Se ha ignorado la función 1: Se ha realizado la asignación de valores De lo contrario: Error
Nota	• Previamente se debió haber llamado a la función start_mean_rssi • Esta función sólo realizará la asignación de los valores cuando el módulo haya suministrado la información. Esto permite que la ejecución del programa en un uC PIC no se detenga al llegar a esta función. • Valor <15: Valor RSSI, de lo contrario: Error

D.25.5 start_mm_rssi

Firmware	**Cualquiera**
Función	Comienza la obtención de 99 valores RSSI para determinar el máximo, mínimo y promedio
Nivel de ejecución	ESP 3 y RSSI 0, y pasa a RSSI 2
Prototipo	uint8 start_mm_rssi(_bool mode, uint16 interval_time)
Parámetros	_bool mode: Método para obtener los valores máximo, mínimo y promedio 0: Desviación estándar 1: Quartiles 2: Valores atípicos (Calculado con Quartiles) 3: Mayor y menor valor leído uint16 interval_time: Intervalo de tiempo en milisegundos para obtener un nuevo valor RSSI [5- 65535]
Retorno	0: Éxito De lo contrario: Error
Nota	• Si ocurre un error durante el proceso de obtención de valores RSSI, pasará a RSSI 0 y, por tanto, la función get_mm_rssi retornará un error de nivel • Se necesita estar conectado a un AP

D.25.6 get_mm_rssi

Firmware	**Cualquiera**
Función	Obtiene los valores RSSI máximo, mínimo y promedio
Nivel de ejecución	RSSI 2 y pasa a RSSI 0
Prototipo	uint8 get_mm_rssi(real32 *rssi_max,real32 *rssi_min,real32 *rssi_mean)
Parámetros	real32 *rssi_max: Dirección de la variable donde se almacenará el valor máximo real32 *rssi_min: Dirección de la variable donde se almacenará el valor mínimo real32 *rssi_mean: Dirección de la variable donde se almacenará el valor promedio
Retorno	0: Se ha ignorado la función 1: Se ha realizado la asignación de valores De lo contrario: Error
Nota	• Previamente se debió haber llamado a la función start_mm_rssi • Esta función sólo realizará la asignación de los valores cuando el módulo haya suministrado la información Esto permite que la ejecución del programa en un uC PIC no se detenga al llegar a esta función. • Valor <15: Valor RSSI, de lo contrario: Error

D.25.7 start_check_interval_rssi

Firmware	**Cualquiera**
Función	Comienza a constantemente obtener valores RSSI y su comprobar si éstos están dentro de un rango establecido.
Nivel de ejecución	ESP 3 y RSSI 0, y pasa a RSSI 3
Prototipo	uint8 start_check_interval_rssi(uint8 samples, real32 min, real32 max, uint16 interval_time) (NONOS SDK) uint8 start_check_interval_rssi(uint8 samples, uint8 min, uint8 max, uint16 interval_time) (IDF)
Parámetros	uint8 samples: Número de muestras para obtener el valor que se comprobará en el intervalo [1-10] (promedios móviles) min: Valor RSSI mínimo del intervalo max: Valor RSSI máximo del intervalo uint16 interval_time: Intervalo de tiempo en milisegundos para obtener un nuevo valor RSSI [5- 65535]
Retorno	0: Éxito De lo contrario: Error
Nota	• Si simples>2 el tiempo para obtener un valor que será comparando en el intervalo será de Interval_time*smaples • En NONOS SDK min y max pueden ser valores con punto decimal con máximo 3 dígitos a la derecha del punto • En IDF min y max son valores enteros • Si ocurre un error durante el proceso de obtención de valores RSSI, pasará a RSSI 0 y por tanto, la función interval_alarm_rssi retornará un error de nivel • Se necesita estar conectado a un AP

D.25.8 stop_check_interval_rssi

Firmware	**Cualquiera**
Función	Detiene la constante obtención de valores RSSI y su comprobación dentro del rango establecido.
Nivel de ejecución	ESP 3 y RSSI 3, y pasa a RSSI 0
Prototipo	uint8 stop_check_interval_rssi(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Previamente se debió llamar a la función start_check_interval_rssi

D.25.9 interval_alarm_rssi

Firmware	**Cualquiera**
Función	Indica si se ha obtenido un valor RSSI fuera del rango establecido
Nivel de ejecución	RSSI 3
Prototipo	uint8 interval_alarm_rssi(void)
Parámetros	Ninguno
Retorno	0: Valores RSSI dentro del rango 1: Valores RSSI fuera del rango De lo contrario: Error

D.25.10 setup_ssid_rssi

Firmware	**Cualquiera**
Función	Especifica los SSID de los cuales se obtendrá su RSSI
Nivel de ejecución	ESP 2 o 3 y RSSI 0
Prototipo	uint8 setup_ssid_rssi(uint8 *ssid1[, uint8 *ssid2[,uint8 *ssid3[,uint8 *ssid4[,uint8 *ssid5]]]])
Parámetros	uint8 *ssid1: *String* del nombre del primer SSID uint8 *ssid2: *String* del nombre del segundo SSID uint8 *ssid3: *String* del nombre del tercer SSID uint8 *ssid4: *String* del nombre del cuarto SSID uint8 *ssid5: *String* del nombre del quinto SSID
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño de los nombres deber ser máximo de 20 bytes • Esta función debe llamarse antes de start_ssid_rssi

D.25.11 start_ssid_rssi

Firmware	**Cualquiera**
Función	Inicia el proceso para obtener el promedio de los valores RSSI para cada uno de los SSID registrados conforme al número de muestras especificado
Nivel de ejecución	ESP 3 y RSSI 0, y pasa a RSSI 4
Prototipo	uint8 start_ssid_rssi(uint8 samples)
Parámetros	uint8 samples: Número de muestras con las que se obtendrá el promedio de cada SSID, [1-10]
Retorno	0: Éxito De lo contrario: Error
Nota	Esta función debe llamarse despues de setup_ssid_rssi Si ocurre un error durante el proceso de obtención de valores RSSI, pasará a RSSI 0 y por tanto, la función get_ssid_rssi retornará un error de nivel No se necesita estar conectado a un AP

D.25.12 get_ssid_rssi

Firmware	**Cualquiera**
Función	Obtiene el promedio de los valores RSSI para cada uno de los SSID registrados conforme al número de muestras especificado
Nivel de ejecución	RSSI 4 y pasa a RSSI 0
Prototipo	_bool get_ssid_rssi(real32 *rssi1[, real32 *rssi2[, real32 *rssi3[, real32 *rssi4[, real32 *rssi5]]]])
Parámetros	real32 *rssi1: Dirección de la variable donde se almacenará el valor RSSI del primer SSID real32 *rssi2: Dirección de la variable donde se almacenará el valor RSSI del segundo SSID real32 *rssi3: Dirección de la variable donde se almacenará el valor RSSI del tercer SSID real32 *rssi4: Dirección de la variable donde se almacenará el valor RSSI del cuarto SSID real32 *rssi5: Dirección de la variable donde se almacenará el valor RSSI del quinto SSID
Retorno	0: Se ha ignorado la función 1: Se ha realizado la asignación de valores
Nota	• Previamente en algún punto del programa se debió llamar a la función start_ssid_rssi • Esta función sólo realizará la asignación de los valores cuando el módulo haya suministrado la información. Esto permite que la ejecución del programa en un uC PIC no se detenga al llegar a esta función. • Valor <15: Valor RSSI, de lo contrario: Error

D.26 Status led

D.26.1 enable_status_led

Firmware	**SDK NONOS V3**
Función	Habilita un GPIO del módulo WiFi para indicar mediante un LED el estado de la conexión a un punto de acceso. Éste parpadeará durante el proceso de conexión y si ésta fue exitosa dejará de parpadear
Nivel de ejecución	ESP 1, 2 o 3
Prototipo	uint8 enable_status_led(uint8 gpio)
Parámetros	uint8 gpio: Número del GPIO utilizado para *status led,* GPIO 4, 5, 9, 10, 12, 13, 14, 15
Retorno	0: Exito De lo contrario: Error

D.26.2 diable_status_led

Firmware	**SDK NONOS V3**
Función	Deshabilita *status led*, permitiendo al GPIO ser utilizado en otra tarea
Nivel de ejecución	ESP 1, 2 o 3
Prototipo	uint8 disable_status_led(uint8 gpio)
Parámetros	uint8 gpio: Número del GPIO utilizado para *status led,* GPIO 4, 5, 9, 10, 12, 13, 14, 15
Retorno	0: Exito De lo contrario: Error

D.27 MDNS

D.27.1 start_mdns

Firmware	**Cualquiera**
Función	Habilita la aplicación, se une al grupo MDNS
Nivel de ejecución	ESP 2 o 3 y MDNS 0 y pasa a MDNS 1
Prototipo	uint8 start_mdns(uint16 ttl, uint8 *hostname) (SDK NONOS) uint8 start_mdns(uint8 *hostname) (ESP-IDF)
Parámetros	uint16 ttl: Tiempo en segundos que se reclamará el *hostname*,[0,65500] uint8 *hostname: *String* del *hostname* que se reclamará, máximo 20 bytes
Retorno	0: Éxito De lo contrario: Error
Nota	En SDK NONOS esta función tarda aproximadamente 2 segundos en retornar un valor, ya que en dicho tiempo verifica que el *hostname* sea único dentro del grupo y realiza el enuncio del mismo.

D.27.2 stop_mdns

Firmware	**Cualquiera**
Función	Deja el grupo MDNS y deshabilita esta aplicación
Nivel de ejecución	ESP 2 o 3 y MDNS 1 y pasa a MDNS 0
Prototipo	uint8 stop_mdns(_bool mode) (SDK NONOS) uint8 stop_mdns(void) (ESP-IDF)
Parámetros	_bool mode: Indica la forma en que dejará el grupo 0: No notifica a los participantes del grupo de su salida y, por tanto, éstos lo eliminan de su memoria cache tras vencerse el TTL 1: Notifica a los participantes del grupo de su salida y, por tanto, éstos lo eliminan de su memoria cache
Retorno	0: Éxito De lo contrario: Error

D.27.3 query_mdns

Firmware	**Cualquiera**
Función	Obtiene la IP y el TTL del *hostname*
Nivel de ejecución	ESP 2 o 3 y MDNS 1
Prototipo	uint8 query_mdns(uint8 *hostname, uint8 *ip, uint8 length ,uint32 *ttl)
Parámetros	uint8 *hostname: *String* del *hostname*, máximo 20 bytes uint8 *ip: Arreglo donde se almacenará la dirección IP uint8 *length: Tamaño del arreglo donde se almacenará la IP uint32 *ttl: Variable que tomará el valor TTL
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño mínimo del arreglo que almacenará la IP es de 16 bytes • Sólo obtendrá direcciones IPv4 • En IDF el valor TTL que se obtiene es una constante definida en el *fimrware*, por lo que no tiene relevancia alguna.

D.27.4 setup_service1_mdns

Firmware	**Cualquiera**
Función	Configura un servicio a ser anunciado en el puerto establecido por el primer servidor
Nivel de ejecución	ESP 3 y MDNS 0
Prototipo	uint8 setup_service1_mdns(uint8 *service_name[, uint8 *description1[, [uint8 *description2]]) (SDK NONOS) setup_service1_mdns(uint8 *service_name, uint8 *description1_key=NULL, uint8 *description1_value=NULL, uint8 *description2_key=NULL, uint8 *description2_value=NULL) (IDF)
Parámetros	uint8 *service_name: *String* del nombre del servicio a ser anunciado, máximo 20 bytes uint8 *description1: Arreglo que contiene la primera descripción del servicio (DNS-SD TXT Record), máximo 30 bytes uint8 *description2: Arreglo que contiene la segunda descripción del servicio (DNS-SD TXT Record), máximo 30 bytes
Retorno	0: Éxito De lo contrario: Error
Nota	• Las descripciones deben presentar generalmente una estructura “key=value” • Requiere habilitar el primer servidor

D.27.5 setup_service2_mdns

Firmware	**Cualquiera**
Función	Configura un servicio a ser anunciado en el puerto establecido por el segundo servidor o servidor SSL
Nivel de ejecución	ESP 3 y MDNS 0
Prototipo	uint8 setup_service2_mdns(uint8 *service_name[, uint8 *description1[, [uint8 *description2]]]) uint8 setup_service2_mdns(uint8 *service_name, uint8 *description1_key=NULL, uint8 *description1_value=NULL, uint8 *description2_key=NULL, uint8 *description2_value=NULL) (IDF)
Parámetros	uint8 *service_name: *String* del nombre del servicio a ser anunciado, máximo 20 bytes uint8 *description1: Arreglo que contiene la primera descripción del servicio (DNS-SD TXT Record), máximo 30 bytes uint8 *description2: Arreglo que contiene la segunda descripción del servicio (DNS-SD TXT Record), máximo 30 bytes
Retorno	0: Éxito De lo contrario: Error
Nota	• Las descripciones deben presentar generalmente una estructura “key=value” • Requiere habilitar el segundo servidor o el servidor SSL

D.28 ESPNOW

- Los términos maestro y esclavo únicamente hacen referencia a la interfaz de salida
  - Maestro-> Interfaz de estación
  - Esclavo->Interfaz de AP
  - Maestro-Esclavo-> Interfaz de AP

D.28.1 up_to_level2

Firmware	**Cualquiera**
Función	Pasa del nivel ESP 1 al nivel ESP 2 configurando al módulo como estación (sin conectarse a un AP) o como punto de acceso dependiendo de la configuración elegida en el archivo de cabecera.
Nivel de ejecución	ESP 1
Prototipo	uint8 up_to_level2(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.28.2 begin_espnow

Firmware	**Cualquiera**
Función	Habilita el uso de *espnow*. Opcionalmente establece una llave para encriptar la llave del *peer*
Nivel de ejecución	ESP 1 o 2 y ESPNOW 0 y pasa a ESP3 y ESPNOW 1
Prototipo	uint8 begin_espnow([uint8 *key,uint8 length])
Parámetros	uint8 *key: Arreglo que contiene la llave, en formato hexadecimal y sin caracteres de separación, utilizada para encriptar la llave del peer uint8 length: Tamaño de la llave
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño de la llave debe ser de 16 bytes • Todos los *peer* deben utilizar la misma llave, si la llave no es especificada, se usa una por defecto.

D.28.3 finish_espnow

Firmware	**Cualquiera**
Función	Deshabilita el uso de *espnow.*
Nivel de ejecución	ESP 3 y ESPNOW 1 y pasa a ESP2* y ESPNOW 0
Prototipo	uint8 finish_espnow(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	• *Pasará a ESP 2 siempre y cuando no haya más servidores y/o clientes activos • Se eliminará cualquier información de los peer que se haya almacenado

D.28.4 set_buffer_espnow

Firmware	**Cualquiera**
Función	Establece el buffer que será utilizado por ESPNOW
Nivel de ejecución	ESP 1 o 2
Prototipo	uint8 set_buffer_espnow(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará los datos recibidos uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más un byte extra que es ocupado internamente • El tamaño mínimo del buffer es de 2 bytes

D.28.5 begin_filter_espnow

Firmware	**Cualquiera**
Función	Habilita la utilización del filtro de direcciones MAC y por tanto, únicamente se recibirá información mediante ESPNOW de los módulos WiFi cuyas direcciones MAC sean especificadas en esta función.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 begin_filter_espnow(uint8 *mac1, [uint8 *mac2, uint8 *mac3])
Parámetros	uint8 *mac1: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del primer *peer* uint8 *mac2: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del segundo *peer* uint8 *mac3: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del tercer *peer*
Formato	Formato de la dirección MAC almacenada en el arreglo: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• Si la función es llamada repetidas veces, únicamente se guardarán las últimas direcciones MAC establecidas • La dirección MAC no es *case sensitive*

D.28.6 finish_filter_espnow

Firmware	**Cualquiera**
Función	Deshabilita la utilización del filtro de direcciones MAC y por tanto, se recibirá información mediante ESPNOW de cualquier módulo WiFi
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 finish_filter_espnow(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.28.7 add_peer_espnow

Firmware	**Cualquiera**
Función	Añade un *peer* a la tabla de comunicación interna ESPNOW
Nivel de ejecución	ESP 2 o 3 y ESPNOW 1
Prototipo	uint8 add_peer_espnow(uint8 *mac, uint8 peer_role, uint8 peer_channel, [uint8 *peer_key,uint8 length])
Parámetros	uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* uint8 peer_role: Indica si el *peer* es un maestro o esclavo* 0: Maestro 1: Esclavo 2: Maestro-esclavo uint8 peer_channel: Canal de comunicación del *peer* [1-13]* uint8 *peer_key: Arreglo que contiene la llave utilizada para encriptar el *payload* durante la comunicación uint8 length: Tamaño de la llave utilizada para encriptar el *payload* durante la comunicación
Formato	Formato de la dirección MAC almacenada en el arreglo: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• Si se establece una llave, el *peer* deberá también establecer la misma llave • Si no se establece la llave, se usará una por defecto • *En SDK NONOS estos parámetros no tienen ningún efecto en las funciones internas del módulo, sólo son almacenados para utilizarlos en la capa de aplicación • Se pueden registrar mediante esta función hasta 6 *peer.* • La información de los *peer* queda almacenada hasta que se deshabilite ESPNOW • La dirección MAC no es *case sensitive*

D.28.8 save_mac_espnow

Firmware	**Cualquiera**
Función	Habilita o deshabilita el añadir los *peers* de los que se recibió un mensaje a la tabla de comunicación interna ESPNOW
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 save_mac_espnow(_bool mode)
Parámetros	_bool mode: 0: Deshabilita el añadir *peers* a la tabla de comunicación ESPNOW 1: Habilita el añadir *peers* a la tabla de comunicación ESPNOW
Retorno	0: Éxito De lo contrario: Error
Nota	• Los *peer* almacenados utilizarán una llave por defecto. • Esta función permite que se puedan registrar hasta 5 *peer* • La información de los *peer* queda almacenada hasta que se deshabilite ESPNOW.

D.28.9 send_espnow

Firmware	**Cualquiera**
Función	Envía los datos al *peer* especificado
Nivel de ejecución	ESP 3 y ESPNOW 1
Prototipo	uint8 send_espnow(uint8 *mac, uint8* message)
Parámetros	uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* al que se le enviará los datos uint8 *message: Arreglo que contiene los datos a enviar
Formato	Formato de la dirección MAC almacenada en el arreglo: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño máximo del arreglo a enviar es de 250 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados. • La dirección MAC no es *case sensitive*

D.28.10 send_all_espnow

Firmware	**Cualquiera**
Función	Envía los datos a todos los *peer* almacenados en la tabla de comunicación interna ESPNOW
Nivel de ejecución	ESP 3 y ESPNOW 1
Prototipo	uint8 send_all_espnow(uint8* message)
Parámetros	uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	El tamaño máximo del arreglo a enviar es de 250 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.28.11 send_espnow_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al *peer* especificado
Nivel de ejecución	ESP 3 y ESPNOW 1

Macro	#define send_espnow_const(a,m)
	{
	presend_espnow_const(m,sizeof(a)-1);
	sending_espnow_const(a);
	}

Prototipo	send_espnow_const(a,m)
Parámetros	m: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* al que se le enviará los datos a: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	Ninguno
Nota	• El tamaño máximo del arreglo a enviar es de 250 bytes • La dirección MAC no es *case sensitive*

D.28.12 ret_send_espnow_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM al *peer* especificado
Nivel de ejecución	ESP 3 y ESPNOW 1

Macro	#define ret_send_espnow_const(a,m,r)
	{
	presend_espnow_const(m,sizeof(a)-1);
	r=sending_espnow_const(a);
	}

Prototipo	send_espnow_const(a,m,r)
Parámetros	m: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* al que se le enviará los datos a: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	Ninguno
Nota	• El tamaño máximo del arreglo a enviar es de 250 bytes • La dirección MAC no es case sensitive

D.28.13 send_espnow_all_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM a todos los *peer* de la tabla de comunicación interna ESPNOW
Nivel de ejecución	ESP 3 y ESPNOW 1

Macro	#define send_espnow_all_const(m)
	{
	presend_espnow_const_all(sizeof(m)-1);
	sending_espnow_const(m);
	}

Prototipo	send_espnow_all_const(m)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar
Retorno	Ninguno
Nota	El tamaño máximo del arreglo a enviar es de 250 bytes

D.28.14 ret_send_espnow_all_const

Firmware	**Cualquiera**
Descripción	Es una macro utilizada para enviar datos almacenados en la ROM a todos los *peer* de la tabla de comunicación interna ESPNOW
Nivel de ejecución	ESP 3 y ESPNOW 1

Macro	#define send_espnow_all_const(m,r)
	{
	presend_espnow_const_all(sizeof(m)-1);
	r=sending_espnow_const(m);
	}

Prototipo	send_espnow_all_const(m,r)
Parámetros	m: Arreglo de tipo uint8 almacenado en la ROM que contiene los datos a enviar r: Variable de tipo uint8 que almacenará el estado final del envío 0: Éxito De lo contrario: Error
Retorno	Ninguno
Nota	El tamaño máximo del arreglo a enviar es de 250 bytes

D.28.15 presend_espnow_const

Firmware	**Cualquiera**
Función	Establece el número de bytes a enviar y la dirección MAC del *peer*
Nivel de ejecución	ESP 3 y ESPNOW 1
Prototipo	void presend_espnow_const(uint8 *mac, uint8 length)
Parámetros	uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* al que se le enviará los datos uint8 length: Número de bytes a enviar
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	Ninguno
Nota	• Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_espnow_const • La dirección MAC no es case sensitive

D.28.16 presend_espnow_const_all

Firmware	**Cualquiera**
Función	Establece el número de bytes a enviar
Nivel de ejecución	ESP 3 y ESPNOW 1
Prototipo	void presend_espnow_const_all(uint8 length)
Parámetros	uint8 length: Número de bytes a enviar
Retorno	Ninguno
Nota	Tras la ejecución de esta función se debe llamar inmediatamente a la función sending_espnow_const

D.28.17 sending_espnow_const

Firmware	**Cualquiera**
Función	Envía el número de bytes establecido en presend_espnow_const o presend_espnow_const_all
Nivel de ejecución	Cualquiera
Prototipo	uint8 sending_espnow_const(uint8 message)
Parámetros	uint8 message: Arreglo almacenado en la memoria ROM que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• Para ejecutar esta función se debió haber llamado previamente presend_espnow_const o presend_espnow_const_all

D.29 Simple pair

D.29.1 up_to_level2

Consultar D.28.1 set_buffer_server1

D.29.2 begin_simple_pair

Firmware	**SDK NONOS**
Función	Habilita el uso de *simple pair*
Nivel de ejecución	ESP 2 o 3 y SIMPLE PAIR 0
Prototipo	uint8 begin_simple_pair(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.29.3 finish_simple_pair

Firmware	**SDK NONOS**
Función	Deshabilita el uso de *simple pair*
Nivel de ejecución	ESP 2 o 3 y SIMPLE_PAIR 1
Prototipo	uint8 finish_simple_pair(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.29.4 simple_pair_check

Firmware	**SDK NONOS**
Función	Verifica que los peer especificados hayan habilitado *simple pair*
Nivel de ejecución	ESP 2 o 3
Requiere	Módulo WiFi como estación
Prototipo	uint8 simple_pair_check(uint8 *mac1[,uint8 *mac2[, uint8 *mac3]])
Parámetros	uint8 *mac1: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del primer *peer* uint8 *mac2: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del segundo *peer* uint8 *mac3: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del tercer *peer*
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	Valor numérico determinado por los tres bits menos significativos. El estado de cada bit representa si el *peer* ha habilitado (1) o no (0) *simple pair.* Bit0 representa el estado del primer *peer* Bit1 representa el estado del segundo *peer* Bit2 representa el estado del tercer *peer* Valor numérico: <8: Éxito De lo contrario: Error
Nota	La dirección MAC no es case sensitive

D.29.5 get_message_simple_pair

Firmware	**SDK NONOS**
Función	Obtiene la información anunciada por un *peer* mediante *simple* pair. No verifica que el peer con el que negociará haya habilitado *simple pair*
Nivel de ejecución	ESP 2 o 3 y SIMPLE_PAIR 1
Requiere	Módulo WiFi como estación
Prototipo	uint8 get_message_simple_pair(_bool mode,_bool repeat, uint8 *mac, uint8 *buffer, uint8 buffer_length, uint8 *key, uint8 key_length)
Parámetros	_bool mode: Tipo de mensaje 0: Mensaje cualquiera 1: Llave hexadecimal _bool repeat: Comportamiento de la función 0: Esta función no será llamada repetidamente* 1: Esta función será llamada repetidamente* uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* uint8 *buffer: Arreglo donde se almacenarán los datos recibidos uint8 buffer_length: Tamaño del arreglo donde se almacenarán los datos recibidos uint8 *key: Arreglo que contiene la llave utilizada para encriptar temporalmente la comunicación uint8 key_length: Tamaño de la llave utilizada para encriptar temporalmente la comunicación
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• Las llaves utilizadas deben ser las mimas en ambos *peer* • El tamaño mínimo del buffer es de 17 bytes, el último byte es ocupado para insertar el \0 • El tamaño de la llave debe ser de 16 bytes • En caso de que el *peer* no habilite *simple pair,* tendrá que esperarse un tiempo para llamar nuevamente a esta función debido al *timeout* en la negociación • La dirección MAC no es case sensitive

D.29.6 get_message_check_simple_pair

Firmware	**SDK NONOS**
Función	Obtiene la información anunciada por un *peer* mediante *simple* pair. Antes de iniciar la negociación verifica que el peer con el que negociará ha habilitado *simple pair*
Nivel de ejecución	ESP 2 o 3 y SIMPLE_PAIR 1
Requiere	Módulo WiFi como estación
Prototipo	uint8 get_message_check_simple_pair(_bool mode,_bool repeat, uint8 *mac, uint8 *buffer, uint8 buffer_length,uint8 *key, uint8 key_length)
Parámetros	_bool mode: Tipo de mensaje 0: Mensaje cualquiera 1: Llave hexadecimal _bool repeat: Comportamiento de la función 0: Esta función no será llamada repetidamente* 1: Esta función será llamada repetidamente* uint8 *mac: *string* de la dirección MAC, en formato hexadecimal con caracteres de separación, del *peer* uint8 *buffer: Arreglo donde se almacenarán los datos recibidos uint8 buffer_length: Tamaño del arreglo donde se almacenarán los datos recibidos uint8 *key: Arreglo que contiene la llave utilizada para encriptar temporalmente la comunicación uint8 key_length: Tamaño de la llave utilizada para encriptar temporalmente la comunicación
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• Las llaves utilizadas deben ser las mimas en ambos *peer* • El tamaño mínimo del buffer es de 17 bytes, el último byte es ocupado para insertar el \0 • El tamaño de la llave debe ser de 16 bytes • En caso de que el *peer* no habilite *simple pair* esta función regresará inmediatamente error • La dirección MAC no es case sensitive

D.29.7 set_message_simple_pair

Firmware	**SDK NONOS**
Función	Establece el mensaje a compartir mediante *simple pair*
Nivel de ejecución	ESP 2 o 3 y SIMPLE_PAIR 1
Requiere	Módulo WiFi como AP
Prototipo	uint8 set_message_simple_pair(_bool mode, uint8 *message, uint8 message_length)
Parámetros	_bool mode: Tipo de mensaje 0: Mensaje cualquiera 1: Llave hexadecimal uint8 *message: Arreglo que contiene los datos a enviar uint8 message_length: Tamaño del mensaje a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• Si el tipo de mensaje es una llave hexadecimal, el tamaño de ésta debe ser de 16 bytes • Si el tipo de mensaje es un mensaje cualquiera, el tamaño máximo de éste debe ser de 16 bytes

D.29.8 announcing_message_simple_pair

Firmware	**SDK NONOS**
Función	Anuncia la disponibilidad del mensaje para que éste pueda ser solicitado mediante *simple pair.*
Nivel de ejecución	ESP 2 o 3 y SIMPLE_PAIR 1
Requiere	Módulo WiFi como AP
Prototipo	uint8 announcing_message_simple_pair(_bool mode, uint8 *key, uint8 key_length)
Parámetros	_bool mode: Funcionamiento tras la comunicación 0: Tras un intercambio exitoso de la información se deberá llamar nuevamente a esta función. Este modo es para peticiones esporádicas. 1: Tras un intercambio exitoso de la información no se requiere llamar nuevamente a esta función. Este modo es para peticiones constantes uint8 *key1: Arreglo que contiene la llave utilizada para encriptar temporalmente la comunicación uint8 key_length: Tamaño de la llave para encriptar temporalmente la comunicación
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño de la llave debe ser de 16 bytes • Previamente se debió haber llamado a la función set_message_simple_pair • Las llaves utilizadas deben ser las mimas en ambos *peer*

D.29.9 begin_filter_simple_pair

Firmware	**SDK NONOS**
Función	Habilita la utilización del filtro de direcciones MAC y por tanto, únicamente las direcciones MAC de los módulos WiFi especificadas en esta función y que habiliten *simple-pair* se les suministrará la información
Nivel de ejecución	ESP 2 o 3
Requiere	Módulo WiFi como AP
Prototipo	uint8 begin_filter_simple_pair(uint8* mac1, uint8 key1,uint8 key1_length[, uint8 mac2, uint8 *key2, uint8 key2_length])
Parámetros	uint8 *mac1: *string* de la primer dirección MAC, en formato hexadecimal con caracteres de separación uint8 *key1: Arreglo que contiene la llave temporal, utilizada para encriptar la comunicación, para la primer MAC uint8 key_length1: Tamaño de la llave para la primer MAC uint8 *mac2: *string* de la segunda dirección MAC, en formato hexadecimal con caracteres de separación uint8 *key2: Arreglo que contiene la llave temporal, utilizada para encriptar la comunicación, para la segunda MAC uint8 key_length2: Tamaño de la llave para la segunda MAC
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño de las llaves debe ser de 16 bytes • Las llaves utilizadas deben ser las mimas en ambos *peer* • Si la función es llamada repetidas veces, únicamente se guardarán las últimas direcciones MAC establecidas • La dirección MAC no es *case sensitive*

D.29.10 finish_filter_simple_pair

Firmware	**SDK NONOS**
Función	Deshabilita la utilización del filtro de direcciones MAC y por tanto, cualquier módulo WiFi que habilite *simple-pair* puede solicitar la información
Nivel de ejecución	ESP 2 o 3
Requiere	Módulo WiFi como AP
Prototipo	uint8 finish_filter_simple_pair(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.30 Tracking

- El módulo WiFi necesita ser estación

D.30.1 up_level2_to_level3

Consultar D.25.1 up_level2_to_level3

D.30.2 up_to_level2

Consultar D.28.1 up_to_level2

D.30.3 enable_specific_channel_traking

Firmware	**Cualquiera**
Función	Habilita la detección de paquetes *probe request,* los cuales serán buscados en un canal de frecuencia en específico. En seste caso, únicamente se detectarán los *probe request* cuyo SSID sea ESP8266.
Nivel de ejecución	ESP 3
Prototipo	uint8 enable_specific_channel_traking(uint8 channel)
Parámetros	uint8 channel: ID del canal donde se buscarán los paquetes *probe request*
Retorno	0: Éxito De lo contrario: Error
Nota	Las interfaces AP y estación serán desactivadas. Utilizar espnow como protocolo de comunicación.

D.30.4 enable_var_channel_traking

Firmware	**Cualquiera**
Función	Habilita la detección de paquetes *probe request,* los cuales serán buscados en cada uno de los canales de frecuencia [1-14]. En este caso, únicamente se detectarán los *probe request* cuyo SSID sea ESP8266.
Nivel de ejecución	ESP 3
Prototipo	uint8 enable_var_channel_traking(uint16 time)
Parámetros	uint16 time: Tiempo en milisegundos que pasará en un canal buscando los paquetes *probe request* [10-6000]
Retorno	0: Éxito De lo contrario: Error
Nota	Los canales se irán recorriendo de manera secuencial del 1 al 14. Las interfaces AP y estación serán desactivadas. Utilizar espnow como protocolo de comunicación.

D.30.5 disable_traking

Firmware	**Cualquiera**
Función	Deshabilita la detección de paquetes *probe request*
Nivel de ejecución	ESP 2 y 3
Prototipo	uint8 disable_traking(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.30.6 insert_mac_tracking

Firmware	**Cualquiera**
Función	Registra la dirección MAC del módulo WiFi del cual se trata de detectar su presencia
Nivel de ejecución	ESP 2 y 3
Prototipo	uint8 insert_mac_traking(uint8 id, uint8 *mac)
Parámetros	uint8 id: Identificador a asignar a la dirección MAC [1-5] uint8 *mac: *String* de la dirección MAC en formato hexadecimal con caracteres de separación.
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	• El ID se asignará a la última MAC a la que se a*Firmware*ie • Los módulos WiFi a detectar deben emitir paquetes *probe request* • La dirección MAC no es *case sensitive* • Se podrán almacenar hasta 6 direcciones MAC

D.30.7 delete_mac_tracking

Firmware	**Cualquiera**
Función	Deja de detectar la MAC con el ID asociado.
Nivel de ejecución	ESP 2 y 3
Prototipo	_bool check_mac_traking(uint8 id_mac)
Parámetros	uint8 id_mac: ID asignada a la dirección MAC que se dejará de detectar
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	La dirección MAC no es *case sensitive*

D.30.8 check_mac_tracking

Firmware	**Cualquiera**
Función	Deja de detectar la MAC con el ID asociado.
Nivel de ejecución	ESP 2 y 3
Prototipo	_bool check_mac_traking(uint8 id_mac)
Parámetros	uint8 id_mac: ID asignada a la dirección MAC que se dejará de detectar
Formato	Formato de la dirección MAC: 00:01:00:00:0a:01
Retorno	0: Éxito De lo contrario: Error
Nota	La dirección MAC no es *case sensitive*

D.30.9 tracking_print

Firmware	**Cualquiera**
Función	Imprime en el segundo serial creado por *software,* durante el tiempo especificado, todas las direcciones de los módulos WiFi que emiten paquetes *probe request*
Nivel de ejecución	ESP 3
Prototipo	uint8 tracking_print(uint16 probe_time)
Parámetros	uint16 probe_time: Tiempo en segundos que estará imprimiendo en el segundo serial creado por *software* las direcciones MAC de los módulos WiFi que emiten los paquetes *probe request* [1,600]
Retorno	0: Éxito De lo contrario: Error
Nota	El programa se detiene en esta función el probe_time especificado.

D.30.10 tracking_counter

Firmware	**Cualquiera**
Función	Inicia el conteo de los módulos WiFi que emiten paquetes *probe request*
Nivel de ejecución	ESP 3
Prototipo	uint8 tracking_counter(uint16 probe_time)
Parámetros	uint16 probe_time: Tiempo en segundos que estará detectando a los diferentes módulos WiFi que emiten los paquetes *probe request* para su conteo [1,600]
Retorno	0: Éxito De lo contrario: Error
Nota	El número máximo de módulos WiFi que puede detectar es de 30.

D.30.11 get_tracking_counter

Firmware	**Cualquiera**
Función	Obtiene el número de módulos WiFi que emiten paquetes *probe request*
Nivel de ejecución	Ninguno
Prototipo	_bool get_tracking_counter(uint8 *var)
Parámetros	uint8 *var: Dirección de la variable que almacenará el número de módulos WiFi, que emiten paquetes *probe request,* detectados.
Retorno	1: Se ha obtenido el conteo 0: Aún no se ha obtenido el conteo
Nota	El número máximo de módulos WiFi que puede detectar es de 30. Para que esta función obtenga un número se debe llamar previamente a la función *tracking_counter*

D.31 Funciones criptográficas

D.31.1 Cryptographic

Firmware	**Cualquiera**
Función	Aplica el algoritmo especificado a los datos suministrados
Nivel de ejecución	ESP 1, 2 o 3
Prototipo	uint8 cryptographic(uint8 method, uint16 length, uint8 *input, uint8 *buffer)
Parámetros	uint8 method: Algoritmo a aplicar: SHA1 SHA254 SHA256 SHA384 SHA512 MD5 AES ECB Encriptación (128,192,256) AES ECB Desencriptación (128,192,256) AES CBC Encriptación (128,192,256) AES CBC Desencriptación (128,192,256) ARC4 BLOWFISH ECB Encriptación BLOWFISH ECB Desencriptación BLOWFISH CBC Encriptación BLOWFISH CBC Desencriptación XTEA ECB Encriptación XTEA ECB Desencriptación XTEA CBC Encriptación XTEA CBC Desencriptación CAMELLIA ECB Encriptación (128,192,256) CAMELLIA ECB Desencriptación (128,192,256) CAMELLIA CBC Encriptación (128,192,256) CAMELLIA CBC Desencriptación (128,192,256) SDK NONOS V3: MD4 AES CFB128 Encriptación AES CFB28 Desencriptación AES CFB8 Encriptación AES CFB8 Desencriptación IDF: AES CFB128 Encriptación AES CFB28 Desencriptación AES CFB8 Encriptación AES CFB8 Desencriptación uint16 length: Número de datos a los que se les aplicará algún algoritmo. uint8 *input: Arreglo que contiene los datos a los que se les aplicará algún algoritmo uint8 *buffer: Arreglo donde se almacenará el resultado de aplicar el método seleccionado a los datos suministrados
Retorno	0: Éxito De lo contrario: Error
Nota	Algoritmos validos [1-28] SDK NONOS V3 [1-19] SDK NONOS V2 [1-27] IDF

D.31.2 set_key

Firmware	**Cualquiera**
Función	Establece la llave de encriptación o desencriptación de tamaño estándar (16,24,32 bytes)
Nivel de ejecución	ESP 1, 2 o 3
Prototipo	uint8 set_key(uint8 *key, uint8 length)
Parámetros	uint8 *key: Arreglo que contiene la llave en formato hexadecimal y sin caracteres de separación uint8 length: Tamaño de la llave, [16,24,32] bytes
Retorno	0: Éxito De lo contrario: Error
Nota	• Estás llaves son compartidas por todos los métodos que las requieran. • Sólo se conservará la última llave de encriptación establecida • Sólo se conservará la última llave de desencriptación establecida • Algunos métodos, debido a su naturaleza, utilizan la llave de encriptación para desencriptar • 16 bytes==128 bits, 24 bytes ==192 bits, 32 bytes==256 bits

D.31.3 set_var_key

Firmware	**Cualquiera**
Función	Establece la llave utilizada para encriptar o desencriptar información para todos los métodos cuya llave no es de un tamaño fijo.
Nivel de ejecución	ESP 1, 2 o 3
Prototipo	uint8 set_var_key(uint8 *key, uint8 length)
Parámetros	uint8 *key: Arreglo que contiene la llave en formato hexadecimal y sin caracteres de separación uint8 length: Tamaño de la llave, [1-58] bytes
Retorno	0: Éxito De lo contrario: Error
Nota	• Esta llave es compartida por todos los métodos que la requieran • Sólo se conservará la última llave establecida

D.31.4 set_iv_key

Firmware	**Cualquiera**
Función	Establece el vector de inicialización (IV) para los métodos que realizan un cifrado en bloques.
Nivel de ejecución	ESP 1, 2 o 3
Prototipo	uint8 set_iv_key(uint8 *key, uint8 length)
Parámetros	uint8 *key: Arreglo que contiene el vector de inicialización en formato hexadecimal y sin caracteres de separación uint8 length: Tamaño del vector de inicialización, el cual sólo puede ser de 8 o 16 bytes
Retorno	0: Éxito De lo contrario: Error
Nota	• Esta llave es compartida por todos los métodos que la requieran • Sólo se conservará la última llave establecida

D.32 ADC

- Estas funciones se refieren al módulo ADC que incorporan los módulos WiFi.
- En la versión IDF ADC está basado sobre ADC1, para más detalles consultar https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html

D.32.1 read_mod_adc

Firmware	**SDK NONOS V3 y ESP-IDF**
Función	Realiza una lectura del pin ADC
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 read_mod_adc(uint16 *user_adc_value) (SDK NONOS e IDF ESP8266) uint8 read_mod_adc(uint8 channel, uint16 *user_adc_value) (IDF ESP32)
Parámetros	uint16 *user_adc_value: Dirección de la variable donde se almacena el valor obtenido uint8 channel: Canal en el cual se realizará una lectura [0-7]
Retorno	0: Éxito De lo contrario: Error
Nota	ADC1 (8 canales correspondientes a los GPIOs 32 - 39)

D.32.2 enable_adc

Firmware	**ESP-IDF ESP32**
Función	Habilita ADC y establece la resolución
Nivel de ejecución	ESP 2 o 3
Prototipo	enable_adc(uint8 bits)
Parámetros	uint8 bits: 0: ADC_WIDTH_BIT_9 1: ADC_WIDTH_BIT_10 2: ADC_WIDTH_BIT_11 3: ADC_WIDTH_BIT_12
Retorno	0: Éxito De lo contrario: Error
Nota	Esta función debe ser llamada primero para habilitar el resto de las funciones

D.32.3 disable_adc

Firmware	**ESP-IDF ESP32**
Función	Deshabilita ADC
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 disable_adc(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error

D.32.4 config_channel_adc

Firmware	**ESP-IDF ESP32**
Función	Hablita el canal ADC especificado y establece la atenuación de éste
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 config_channel_adc(uint8 channel, uint8 attenuation)
Parámetros	uint8 channel: Canal ADC [0-7] uint8 attenuatión: 0: ADC_ATTEN_DB_0 1: ADC_ATTEN_DB_2_5 2: ADC_ATTEN_DB_6 3: ADC_ATTEN_DB_11
Retorno	0: Éxito De lo contrario: Error
Nota	Consultar: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html

D.33 PWM

- Estas funciones se refieren a los pines PWM del módulo WiFi implementado
- En IDF PWM está desarrollado sobre la API LED Control en modo LOW SPEED, para más detalles sobre el funcionamiento de ésta, consultar https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html

D.33.1 setup_timer_pwm

Firmware	**ESP-IDF ESP32**
Función	Establece el temporizador (LOW_SPEED) que será la fuente de oscilación con la resolución y la frecuencia dadas
Nivel de ejecución	PWM 0
Prototipo	uint8 setup_timer_pwm(uint8 timer, uint32 duty_re,uint32 frequency)
Parámetros	uint8 timer: Temporizador del módulo WiFi que será la fuente de oscilación para los canales PWM [0-3] uint32 duty_re: resolución (bits) para los ciclos de trabajo PWM [1-20] uint32 frequency: Frecuencia en Hertz (Max 40 MHz con 1 bit de resolución de ciclo de trabajo)
Retorno	0: Éxito De lo contrario: Error
Nota	• Los temporizadores pueden ser configurados de maneras diferentes y asignados a los diferentes canales PWM • Si la frecuencia y la resolución no son soportadas regresará error

D.33.2 setup_gpio_pwm

Firmware	**SDK NONOS y ESP-IDF ESP8266**
Función	Cambia los GPIO asociados a los canales PWM. Los GPIO deberán listarse conforme al canal PWM que se desea que ocupen.
Nivel de ejecución	PWM 0 y pasa a PWM 1
Prototipo	uint8 setup_gpio_pwmt8 setup_gpio_pwm(uint8 gpio1,uint8 gpio2,uint8 gpio3,uint8 gpio4);
Parámetros	uitn8 gpio0: Número del gpio que ocupará el canal pwm 0 uint8 gpio1: Número del gpio que ocupará el canal pwm 1 uint8 gpio2: Número del gpio que ocupará el canal pwm 2 uint8 gpio3: Número del gpio que ocupará el canal pwm 3
Retorno	0: Éxito De lo contrario: Error
Nota	• Orden por defecto: GPIO12 canal 0, GPIO 15 canal 1, GPIO 14 canal 2 y GPIO 4 canal 3 • Validos sólo GPIO 12, 15, 14, 4

D.33.3 set_frequency_pwm

Firmware	**ESP_IDF ESP32**
Función	Cambia la frecuencia para el temporizador especificado
Nivel de ejecución	PWM 1
Prototipo	uint8 set_frequency_pwm(uint32 frequency, uint8 timer)
Parámetros	uint8: Temporizador del módulo WiFi que será la fuente de oscilación PWM [0-3] uint32 frequency: Frecuencia en Hertz (Max 40 MHz con 1 bit de resolución de ciclo de trabajo)
Retorno	0: Éxito De lo contrario: Error

D.33.4 begin_pwm (IDF ESP32)

Firmware	**IDF ESP32**
Función	Activa el o los PWM con los parámetros deseados
Nivel de ejecución	Pasa a PWM1
Prototipo	begin_pwm(uint8 channel,uint8 gpio, uint8 timer, uint32 duty);
Parámetros	uint8 channel: Canal a asignar al GPIO uint8 gpio: Número del GPIO que ocupará el canal PWM dado uint8 timer: Fuente de oscilación del GPIO [0-3] uint32 duty: Ciclo de trabajo
Retorno	0: Éxito De lo contrario: Error
Nota	Esta función debe llamarse para cada GPIO

D.33.5 begin_pwm (SDKV3 e IDF ESP8266)

Firmware	**SDK NONOS V3 e IDFSP8266**
Función	Activa el o los PWM con los parámetros deseados
Nivel de ejecución	PWM 0 y pasa a PWM 1
Prototipo	uint8 begin_pwm(uint16 period, uint32 duty0,[uint32 duty1,[uint32 duty2,[uint32 duty3]]])
Parámetros	uint16 period: Periodo de la señal PWM en microsegundos [1000-10000] uint32 duty0: Tiempo en alto del periodo del canal pwm 0 uint32 duty1: Tiempo en alto del periodo del canal pwm 1 uint32 duty2: Tiempo en alto del periodo del canal pwm 2 uint32 duty3: Tiempo en alto del periodo del canal pwm 3
Retorno	0: Éxito De lo contrario: Error
Nota	• El periodo es de 1000 *µs* (1 *KHz*) ~ 10000 *µs* (100 *Hz*) • Duty puede alcanzar como máximo period*1000/45 • Todos los canales comparten el mismo periodo • Esta función debe llamarse una única vez

D.33.6 set_period_pwm

Firmware	**SDK NONOS V3 e ESP-IDF ESP8266**
Función	Restablece el periodo para todos los canales PWM
Nivel de ejecución	PWM 1
Prototipo	uint8 set_period_pwm(uint16 period)
Parámetros	uint16 period: Periodo de la señal PWM en microsegundos [1000-10000]
Retorno	0: Éxito De lo contrario: Error
Nota	• El periodo es de 1000 *µs* (1 *KHz*) ~ 10000 *µs* (100 *Hz*) • Todos los canales comparten el mismo periodo

D.33.7 set_phase_pwm

Firmware	**ESP-IDF ESP8266**
Función	Establece la phase de un canal PWM
Nivel de ejecución	PWM 1
Prototipo	uint8 set_phase_pwm(uint32 phase, uint8 channel)
Parámetros	uint32 duty: pase [-180,180] uint8 channel: Canal PWM al que se asignará el duty [0-7]
Retorno	0: Éxito De lo contrario: Error

D.33.8 set_duty_pwm

Firmware	**SDK NONOS V3 y ESP-IDF**
Función	Establece el *duty* de un canal PWM
Nivel de ejecución	PWM 1
Prototipo	uint8 set_duty_pwm(uint32 duty, uint8 channel)
Parámetros	uint32 duty: Tiempo en alto de la señal uint8 channel: Canal PWM al que se asignará el duty [0-3]SDK NONOS, [0-7] ESP-IDF
Retorno	0: Éxito De lo contrario: Error
Nota	Se apaga PWM con *duty* igual a cero Duty puede alcanzar como máximo *period**1000/45 NONOS

D.34 Software upgrade

D.34.1 start_upgrade_pic

Firmware	**Cualquiera**
Función	Inicia la actualización del programa del PIC mediante WiFi, para esto el módulo utilizado mantendrá inicialmente al uC PIC en estado de reinicio.
Nivel de ejecución	Cualquier nivel
Prototipo	void start_upgrate_pic(void)
Parámetros	Ninguno
Retorno	Ninguno
Nota	En caso de existir un error durante la actualización, tanto el uC PIC como el módulo se reiniciarán y se ejecutará el último programa guardado en el uC PIC.

D.35 Cliente especial

- Sólo se puede crear un *special client*
- Dependiendo del filtro aplicado se obtendrá una parte de los datos recibidos
    - El filtro HTTP obtiene el contenido de la línea de estado inicial, sin la parte que especifica la versión del protocolo, de una respuesta HTTP.
    - El filtro HTML obtiene el contenido del cuerpo de una respuesta HTTP que esté entre las etiquetas <html> y </html>.

D.35.1 set_buffer_special_client

Firmware	**SDK NONOS V3**
Función	Establece el buffer que será utilizado por el *special client*
Nivel de ejecución	ESP 2
Prototipo	uint8 set_buffer_special_client(uint8 *buffer,uint8 buffer_length)
Parámetros	uint8 *buffer: Arreglo donde se almacenará la información recibida uint8 buffer_length: Tamaño del arreglo donde se almacenará los datos recibidos
Retorno	0: Éxito De lo contrario: Error
Nota	• El tamaño del buffer debe ser igual al tamaño deseado más un byte extra, que será en el que se inserte el '\0' • El tamaño mínimo del buffer es de 2 bytes

D.35.2 create_special_pclient

Firmware	**SDK NONOS V3**
Función	Crea un cliente TCP o SSL persistente con el filtro establecido
Nivel de ejecución	ESP 2 y pasa a ESP 3
Prototipo	uint8 create_special_pclient(uint8 filter,uint8 type, uint8 *server, uint16 port)
Parámetros	uint8 filter: Filtro aplicado a los mensajes recibidos 0: Ninguno 1: HTTP 2: HTML uint8 type: Tipo de conexión 0: TCP 1: SSL uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor
Retorno	0: Éxito De lo contrario: Error
Nota	El filtro será aplicado a todos los mensajes que se reciban del servidor Sólo puede crearse un único cliente con este comando

D.35.3 send_special_client

Firmware	**SDK NONOS V3**
Función	Envía datos al servidor.
Nivel de ejecución	ESP 3
Prototipo	uint8 send_special_client(uint8 filter, uint8 *message)
Parámetros	uint8 filter: Filtro aplicado a los mensajes recibidos 0: Ninguno 1: HTTP 2:HTML uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• El filtro seleccionado sólo se aplicará al primer mensaje recibido tras la ejecución de esta función. • El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados. • Aplicable a conexiones persistentes

D.35.4 nsend_spcial_client

Firmware	**SDK NONOS V3**
Función	Se conecta y envía datos a un servidor con una conexión no persistente.
Nivel de ejecución	ESP 2 o 3
Prototipo	uint8 nsend_special_client(uint8 type, uint8 filter, uint8 *server, uint16 port, uint8 *message)
Parámetros	uint8 type: Tipo de conexión 0: TCP 1: SSL uint8 filter: Filtro aplicado a los mensajes recibidos 0: Ninguno 1: HTTP 2:HTML uint8 *server: Arreglo que contiene la dirección IP o el nombre de dominio del servidor uint16 port: Puerto habilitado por el servidor uint8 *message: Arreglo que contiene los datos a enviar
Retorno	0: Éxito De lo contrario: Error
Nota	• Esta función debe utilizarse cuando el *timeout* del servidor es muy pequeño • El tamaño máximo del arreglo a enviar es de 2048 bytes y éste debe estar almacenado en la memoria RAM. Arreglos almacenados en la memoria ROM no son soportados.

D.35.5 delete_special_client

Firmware	**SDK NONOS V3**
Función	Desconecta al *special client* del servidor
Nivel de ejecución	ESP 3 y pasa a ESP 2
Prototipo	uint8 delete_special_client(void)
Parámetros	Ninguno
Retorno	0: Éxito De lo contrario: Error
Nota	Para ejecutar esta función, el servidor no debe tener conexiones abiertas

D.36 Banderas

Se desarrollaron las banderas:

- new_message_server1
- new_message_server2
- new_message_espnow
- new_message_ssl (Para servidor SSL)

Estas tomarán el valor de uno cuando se recibe un nuevo mensaje, por lo que pueden ser utilizadas para procesar únicamente los nuevos mensajes. Una vez procesadas el usuario debe poner en cero la bandera que haya utilizado para detectar el próximo mensaje. Para acceder a estas se debe declarar:

- *extern _bool nombre_de_la_bandera.*

Así mismo, el primer servidor TCP y clientes cuentan con una variable llamada *rda_message_limit* que tomará el valor de la longitud del último mensaje recibido, ya sea del servidor o cliente. El acceso a esta variable puede ser a partir de sus macros *message_server_length* y *message_client_length,* creadas para brindar claridad a la documentación de los programas, de la siguiente manera:

- *extern uint16 nombre_de_la_macro*

Apéndice E

Comandos AT creados para los diversos *firmware*

E.1 RSSI

- El código de estos comandos están en el archivo *user_rssi.c* tanto en la versión SDK NONOS como ESP-IDF.
- No pueden llamarse un comando RSSI hasta que un comando RSSI previamente llamado haya terminado su ejecución.

E.1.1 AT+RSSI

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+RSSI Función: Obtiene un valor RSSI
Respuesta	+RSSI:<rssi>\r\n
Parámetros	Ninguno
Nota	<15 Valor RSSI De lo contrario: Error

E.1.2 AT+RSSIP

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+RSSIP=<samples>,<Interval_time>[,<e_print>] Función: Obtiene el RSSI promedio de n lecturas realizadas
Respuesta	/r/nOK/r/n Al obtener los valores regresará: +RSO:<rssi>/r/n En caso de que ocurra un error durante la obtención de los valores regresará: +RSF\r\n
Parámetros	<samples>: Número de muestras para obtener el valor RSSI promedio [1-600] <Interval_time>: Intervalo de tiempo en milisegundos para obtener un nuevo valor RSSI [5-70000] <e_print>: 0: Deshabilita la impresión de cada valor RSSI 1: Habilita la impresión de cada valor RSSI
Nota	Si se habilita la impresión cada valor RSSI será impreso: <rssi>, <15 Valor RSSI De lo contrario: Error
Ejemplo	AT+RSSIP=100,20,1

E.1.3 AT+RSSIM

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+RSSIM=<mode>,<interval_time>[,<e_print>] Función: Realiza 99 lecturas del valor RSSI en el intervalo de tiempo especificado y obtiene el valor máximo, mínimo y promedio.
Respuesta	/r/nOK/r/n Al obtener los valores regresará: +RSO:<rssi_max>,<rssi_min>,<rssi_mean>/r/n En caso de que ocurra un error durante la obtención de los valores regresará: +RSF\r\n
Parámetros	<mode>: Método para obtener los valores máximo, mínimo y promedio 0: Desviación estándar 1: Quartiles interval_time: Intervalo de tiempo en milisegundos para obtener un nuevo valor RSSI [5-70000] <e_print>: 0: Deshabilita la impresión de cada valor RSSI 1: Habilita la impresión de cada valor RSSI
Nota	Si se habilita la impresión cada valor RSSI será impreso: <rssi>, <15 Valor RSSI De lo contrario: Error
Ejemplo	AT+RSSIM=1,60,0

E.1.4 AT+RSSIC

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+RSSIC=<mode>[,<samples>,<interval_time>,<max>,<min>[,<e_print>]] Función: Habilita o deshabilita la obtención constante de valores RSSI, verificando que éstos se encuentren dentro del intervalo dado. Cuando un valor sale del intervalo, imprime una notificación.
Respuesta	/r/nOK/r/n Al obtener un valor fuera del intervalo: +RSL/r/n En caso de que ocurra un error durante la obtención de los valores regresará: +RSF\r\n
Parámetros	<mode>: 0: Deshabilita 1: Habilita <samples>: Número de muestras para obtener el valor que se comprobará en el intervalo [1-10] (promedios móviles) <Interval_time> Intervalo de tiempo en milisegundos para obtener un nuevo valor RSSI [5-70000] <min>: Valor RSSI mínimo del intervalo <max>: Valor RSSI máximo del intervalo <e_print>: 0: Deshabilita la impresión de cada valor RSSI 1: Habilita la impresión de cada valor RSSI
Nota	Si simples>2 el tiempo para obtener un valor que será comparando en el intervalo será de Interval_time*smaples Si se habilita la impresión cada valor RSSI será impreso: <rssi>, <15 Valor RSSI De lo contrario: Error
Ejemplo	AT+RSSIC=1,2,100,-20,-80,0 (Habilitar) (NONOS e IDF) AT+RSSIC=1,2,100,-20.213,-80.475,0 (Habilitar) (NONOS) AT+RSSIC=0 (Deshabilitar)

E.1.5 AT+RSSISS

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+RSSISS=<ssid1>[,<ssid2>[,<ssid3>[,<ssid4>[,<ssid5>]]]] Función: Especifica los SSID de los cuales se obtendrá su RSSI
Respuesta	/r/nOK/r/n
Parámetros	ssid1: String del nombre del primer SSID ssid2: String del nombre del segundo SSID ssid3: String del nombre del tercer SSID ssid4: String del nombre del cuarto SSID ssid5: String del nombre del quinto SSID
Nota	El tamaño de los nombres deber ser máximo de 20 caracteres
Ejemplo	AT+RSSISS=”red_1”,”red_ab”

E.1.6 AT+RSSISC

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+RSSISC= =<samples>[,<e_print>] Función: Obtiene el promedio de los valores RSSI para cada uno de los SSID registrados conforme al número de muestras especificado
Respuesta	/r/nOK/r/n Al obtener los valores regresará: +RSO:<rssi_s1>[,<rssi_s2>[,<rssi_s3>[,<rssi_s4>[,<rssi_s5>]]]]/r/n En caso de que ocurra un error durante la obtención de los valores regresará: +RSF\r\n
Parámetros	<samples>: Número de muestras con las que se obtendrá el promedio de cada SSID, [1-10] <e_print>: 0: Deshabilita la impresión de cada valor RSSI 1: Habilita la impresión de cada valor RSSI
Nota	<15 Valor RSSI De lo contrario: Error Si se habilita la impresión cada valor RSSI será impreso: <ID>,<rssi>,
Ejemplo	AT+RSSISC=”RED1”,”red2”

E.2 Servidor *upgrade*

- El código de este comando y del servidor creado para este fin, están en el archivo *user_upgrade_server.c* tanto en la versión SDK NONOS como ESP-IDF.

E.2.1 AT+UPG

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+UPG=<pwd> Función: Crear un punto de acceso y un servidor para realizar la actualización de un programa del microcontrolador PIC
Respuesta	Upgrade ready /r/nOK/r/n
Parámetros	<pwd>: Contraseña para conectarse al AP creado por el módulo [8-12] caracteres.
Nota	La dirección IP del servidor creado es 192.168.1.0 y habilita el puerto 100. Previamente se debe ejecutar el comando AT+RST
Ejemplo	AT+UPG=”ESP8266”

E.3 Segundo servidor TCP

- El código de estos comandos para este fin están en el archivo *user_server.c* tanto en la versión SDK NONOS como ESP-IDF.
- La información se recibe: /r/n+IPS,<id>,<len>:<datos>

E.3.1 AT+SV

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SV=<mode>,<port>,<connection>,<timeout> Función: Crea un segundo servidor TCP en el puerto especificado.
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Elimina el servidor 1: Crea el servidor <port>: Puerto que habilitará el servidor [1- 65535] <max_conn>: Número máximo de clientes que podrán conectarse al servidor [1-2] <timeout>: Tiempo en segundos que el servidor mantendrá conectado a un cliente inactivo [0-7200]
Nota	Para eliminar el servidor, éste no debe tener clientes conectados Si <mode>==0 lo demás parámetros no deben ser declarados No usar más de 2 clientes para el primer servidor, cuando se crea el segundo servidor. Los clientes tendrán el ID 2 y 3
Ejemplo	AT+SV=1,100,1,200 (Crear) AT+SV=0 (Eliminar)

E.3.2 AT+SVSEND

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SVSEND =<id_channel>,<length> Función: Envía datos del segundo servidor al cliente especificado
Respuesta	Regresa > después de establecer el comando. Inicia a recibir datos a través del serial. Cuando la longitud de los datos definida es alcanzada, la transmisión de datos empieza. Si los datos son enviados de manera exitosa, regresa: /r/nSEND OK/r/n En caso contrario regresa: /r/nSEND FAIL/r/n
Parámetros	<id_channel>: ID asignado al cliente <length>: Longitud de datos a enviar [1-1000]
Nota	Ninguna

E.3.3 AT+SVC

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SVC=<id_channel> Función: Cierra la conexión con el cliente TCP especificado
Respuesta	/r/nOK/r/n
Parámetros	<id_channel>: ID del cliente a desconectar [2,3,5]
Nota	Si el <id_channel>==5 se cerrarán todos los canales
Ejemplo	AT+SVC=2

E.4 MDNS

- El código de estos comandos están en el archivo *user_mdns.c* en la versión SDK NONOS. Para la creación de estos comandos se modificó el archivo mdns.c de la biblioteca lwip proporcionada por Espressif Systems.

E.4.1 AT+NMDNS

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+NMDNS=<mode>,<ttl>,<hostname> (NONOS) AT+NMDNS=<mode>,<hostname> (IDF) Función: Habilita o deshabilita el uso de MDNS.
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deja el grupo MDNS y deshabilita esta aplicación. No notifica a los participantes del grupo de su salida y, por tanto, éstos lo eliminan de su memoria cache tras vencerse el TTL. 1: Deja el grupo MDNS y deshabilita esta aplicación. Notifica a los participantes del grupo de su salida y, por tanto, éstos lo eliminan de su memoria cache. 0 o 1 deshabilitan MDNS en IDF 2: Habilita la aplicación, se une al grupo MDNS <ttl>: Tiempo en segundos que se reclamará el hostname,[0,65500] <hostname>: String del hostname que se reclamará, máximo 20 caracteres.
Nota	Para habilitar MDNS tarda aproximadamente 3 segundos en retornar un valor, ya que en dicho tiempo verifica que el hostname sea único dentro del grupo y realiza el enuncio del mismo. Si <mode>==0 o 1, no se deben declarar el resto de parámetros En IDF <mode>==0 o1 realiza la misma acción
Ejemplo	AT+NMDNS=2,500,”esp” (Habilitar) AT+NMDNS=1 (Deshabilitar) AT+NMDNS=0 (Deshabilitar)

E.4.2 AT+NMDNSS

Firmware	**SDK NONOS V2, V3 y ESP-IDF**
Comando	AT+NMDNSS=<service_id>,<port>,<service_name> Función: Configura un servicio a ser anunciado
Respuesta	/r/nOK/r/n
Parámetros	<service_id>: ID del servicio a anunciar [1,2] <port>: Puerto habilitado por el servidor, al cual se le anuncia un servicio [1-65535] <service_name>: String del nombre del servicio a ser anunciado, máximo 20 caracteres
Nota	Este comando debe ser llamado antes de AT+NMDNS Este comando se utiliza para configurar hasta dos servicios
Ejemplo	AT+NMDNS=1,100,”luces”

E.4.3 AT+NMDNSST

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+NMDNSST=<service>,<id_tag>,<description> (SDK NONOS) AT+NMDNSST=<service>,<id_tag>,< description _key>,< description _value> (IDF) Función: Establece las etiquetas TXT que describirán el servicio anunciado
Respuesta	/r/nOK/r/n
Parámetros	<service>: ID del servicio al cual se le asignarán las etiquetas TXT, [1,2]. <id_tag>: ID de la etiqueta [0,1] <description>: String de la descripción del servicio,máximo 30 caracteres
Nota	Las descripciones deben presentar generalmente una estructura “clave=valor” Se requiere previamente haber configurado el servicio con AT+NMDNSS Se debe haber establecido previamente una etiqueta TXT con el ID 1 para establecer otra con el ID 2
Ejemplo	AT+NMDNSST=1,0,”version=3.0”

E.4.4 AT+NMDNSQ

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+NMDNSQ=<hostname> Función: Obtiene la IP y el TTL del hostname
Respuesta	+QMDNS:<ttl>,<ip>\r\n
Parámetros	<hostname>: String del hostname, máximo 20 caracteres
Nota	Sólo obtendrá direcciones IPv4 Previamente se debió llamar el comando AT+NMDNS En IDF TTL es una constante definida en el firmwae que no debe ser considerada
Ejemplo	AT+NMDNSSQ=”luces”

E.5 Simple pair

- El código de estos comandos están en el archivo *user_simple_pair.c* en la versión SDK NONOS.
- La información se recibe:+SPM,<len>:<datos>/r/n

E.5.1 AT+SP

Firmware	**SDK NONOS**
Comando	AT+SP=<mode> Función: Habilita o deshabilita simple pair
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita
Nota	En modo estación, es cuando el módulo solicitará información En modo AP, es cuando el módulo suministrará información
Ejemplo	AT+SP=1

E.5.2 AT+SPS

Firmware	**SDK NONOS**
Comando	AT+SPS=<mac1>[,<mac2>[,<mac3>]] (Sólo en modo estación) Función: Verificará si los dispositivos con las MAC establecidas han habilitado simple pair
Respuesta	<state_mac1>,<state_mac2>,<state_mac3>/r/n state_mac: 0: Deshabilitado 1: Habilitado
Parámetros	<mac1>: Primera dirección MAC <mac2>: Segunda dirección MAC <mac3>: Tercera dirección MAC
Nota	El módulo debe estar en modo estación
Ejemplo	AT+SPS=”00:01:00:01:00:01”

E.5.3 AT+SPGET

Firmware	**SDK NONOS**
Comando	AT+SPGET=<type>,<mac>,<temkey> (Sólo en modo estación) Función: Obtiene la información anunciada por un peer mediante simple pair. No verifica que el peer con el que negociará haya habilitado simple pair.
Respuesta	/r/nOK/r/n
Parámetros	<type>: Tipo de mensaje 0: Mensaje cualquiera 1: Llave hexadecimal <mac>: Dirección MAC del peer <temkey>: Llave hexadecimal utilizada para encriptar temporalmente la comunicación, string de 32 caracteres
Nota	El módulo debe estar en modo estación
Ejemplo	AT+SPGET=1,”00:01:00:01:00:01”,”00000000000000000000000000000000”

E.5.4 AT+SPGETS

Firmware	**SDK NONOS**
Comando	AT+SPGETS=<mode>,<mac>,<temkey> (Sólo en modo estación) Función: Obtiene la información anunciada por un peer mediante simple pair. Antes de iniciar la negociación verifica que el peer con el que negociará ha habilitado simple pair.
Respuesta	/r/nOK/r/n
Parámetros	<type>: Tipo de mensaje 0: Mensaje cualquiera 1: Llave hexadecimal <mac>: Dirección MAC del peer <temkey>: Llave hexadecimal utilizada para encriptar temporalmente la comunicación, string de 32 caracteres
Nota	El módulo debe estar en modo estación Las llaves utilizadas deben ser las mimas en ambos peer
Ejemplo	AT+SPGETS=1,”00:01:00:01:00:01”,”00000000000000000000000000000000”

E.5.5 AT+SPSET

Firmware	**SDK NONOS**
Comando	AT+SPSET=<type>,<message> (Solo en modo punto de acceso) Función: Establece el mensaje a compartir mediante simple pair
Respuesta	/r/nOK/r/n
Parámetros	<type>: Tipo de mensaje 0: Mensaje cualquiera, máx16 caracteres 1: Llave hexadecimal, deben ser 32 caracteres <message>: Mensaje a compartir
Nota	El módulo debe estar en modo punto de acceso
Ejemplo	AT+SPSET=0,”holamundo”

E.5.6 AT+SPANU

Firmware	**SDK NONOS**
Comando	AT+SPANU=<mode>,<key> (Solo en modo punto de acceso) Función: Anuncia la disponibilidad del mensaje para que éste pueda ser solicitado mediante simple pair.
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Tras un intercambio exitoso de la información se deberá llamar nuevamente a esta función. Este modo es para peticiones esporádicas. 1: Tras un intercambio exitoso de la información no se requiere llamar nuevamente a esta función. Este modo es para peticiones constantes <key>:>: Llave hexadecimal utilizada para encriptar temporalmente la comunicación, string de 32 caracteres
Nota	Previamente se debió haber llamado al comando AT+SPSET Las llaves utilizadas deben ser las mimas en ambos peer
Ejemplo	AT+SPANU=1,”00000000000000000000000000000000”

E.5.7 AT+SPFILTER

Firmware	**SDK NONOS**
Comando	AT+SPFILTER=<mode>,<mac>,<key>, [,<mac>,<key>] Función: Habilita o deshabilita la utilización del filtro de direcciones MAC y por tanto, únicamente las direcciones MAC de los módulos WiFi especificadas en esta función y que habiliten simple-pair se les suministrará la información
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita <mac>: Dirección MAC del peer <key>: Llave hexadecimal utilizada para encriptar temporalmente la comunicación, string de 32 caracteres
Nota	Las llaves utilizadas deben ser las mimas en ambos peer Máximo se pueden registrar hasta 2 peer Si se deshabilita, el resto de los parámetros no deben ser declarados
Ejemplo	AT+SPFILTER=1,”00:01:00:01:00:01”,”00000000000000000000000000000000” (Habilitar) AT+SPFILTER=0 (Deshabilita)

E.6 Espnow

- El código de estos comandos están en el archivo *user_espnow.c* en la versión SDK NONOS e ESP-IDF.
- La información se recibe: /r/n+ESP,<len>:<datos>

E.6.1 AT+EN

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+EN=<mode>[,<kok>] Función: Habilita o deshabilita espnow. Opcionalmente establece una llave para encriptar la llave del peer
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita <kok>: Llave, en formato hexadecimal y sin caracteres de separación, utilizada para encriptar la llave del peer, string de 32 caracteres
Nota	Todos los peer deben utilizar la misma llave, si la llave no es especificada, se usa una por defecto.
Ejemplo	AT+EN=1

E.6.2 AT+ENFILTER

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+ENFILTER=<mode>,<mac1>[,<mac2>[,<mac3>]] Función: Habilita o deshabilita la utilización del filtro de direcciones MAC. Si se habilita únicamente se recibirá información mediante ESPNOW de los módulos WiFi cuyas direcciones MAC sean especificadas en esta función
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita <mac1>: string de la dirección MAC, en formato hexadecimal con caracteres de separación, del primer peer <mac2>: string de la dirección MAC, en formato hexadecimal con caracteres de separación, del segundo peer <mac3>: string de la dirección MAC, en formato hexadecimal con caracteres de separación, del tercer peer
Nota	Si la función es llamada repetidas veces, únicamente se guardarán las últimas direcciones MAC establecidas
Ejemplo	AT+ENFILTER=1,”00:01:00:01:00:01”

E.6.3 AT+ENADD

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+ENADD=<mac>,<role>,<channel>[,<key>]
Respuesta	/r/nOK/r/n
Parámetros	<mac>: Dirección MAC del peer <role>: Indica si el peer es un maestro o esclavo* 0: Maestro 1: Esclavo 2: Maestro-esclavo <channel>: Canal de comunicación del peer [1-13] <key>: Llave utilizada para encriptar la carga (payload) durante la comunicación con el peer, string de 32 caracteres
Nota	Si se establece una llave, el peer deberá también establecer la misma llave Si no se establece la llave, se usará una por defecto Se pueden registrar mediante esta función hasta 6 peer. La información de los peer queda almacenada hasta que se deshabilite ESPNOW
Ejemplo	AT+ENADD=”00:01:00:01:00:01”,0,2

E.6.4 AT+ENSEND

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+ENSEND=<mac>,<length> Función: Envía los datos al peer especificado
Respuesta	Regresa > después de establecer el comando. Inicia a recibir datos a través del serial. Cuando la longitud de los datos definida es alcanzada, la transmisión de datos empieza. Si los datos son enviados de manera exitosa, regresa: /r/nSEND OK/r/n En caso contrario regresa: /r/nSEND FAIL/r/n
Parámetros	<mac>: Dirección MAC del peer <length>: Longitud de datos a enviar [1-250]
Nota	Ninguna
Ejemplo	AT+SEND=”00:01:00:01:00:01”,3

E.6.5 AT+ENSENDA

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+ENSENDA=<length> Función: Envía los datos a todos los peer almacenados en la tabla de comunicación interna ESPNOW
Respuesta	Regresa > después de establecer el comando. Inicia a recibir datos a través del serial. Cuando la longitud de los datos definida es alcanzada, la transmisión de datos empieza. Si los datos son enviados de manera exitosa, regresa: /r/nSEND OK/r/n En caso contrario regresa: /r/nSEND FAIL/r/n
Parámetros	<length>: Longitud de datos a enviar [1-250]
Nota	Ninguna
Ejemplo	AT+ENSENDA=2

E.6.6 AT+ENSMAC

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+ENSMAC=<mode> Función: Habilita o deshabilita el añadir los peers de los que se recibió un mensaje a la tabla de comunicación interna ESPNOW
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita
Nota	Los peer almacenados utilizarán una llave por defecto. Esta función permite que se puedan registrar hasta 5 peer La información de los peer queda almacenada hasta que se deshabilite ESPNOW.
Ejemplo	AT+ENSMAC=1

E.7 Tracking

- El código de estos comandos están en el archivo *user_tracking.c* en la versión SDK NONOS.
- Cuando una dirección MAC es detectada se recibe el mensaje: +SNMAC:<id_mac>\r\n

E.7.1 AT+SNIF

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SNIF=<mode>[,<aux>] Función: Habilita o deshabilita el modo promiscuo del módulo para detectar paquetes 802.11. En seste caso, únicamente se detectarán los probe request cuyo SSID sea ESP8266.
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita la detección de paquetes probe request, los cuales serán buscados en cada uno de los canales de frecuencia 2: Habilita la detección de paquetes probe request, los cuales serán buscados en un canal de frecuencia en especifico <aux>: Si mode==1, Tiempo en milisegundos que pasará en un canal buscando los paquetes probe request [50-6000] Si mode==2, ID del canal donde se buscarán los paquetes probe request [1-13].
Nota	Las interfaces AP y estación serán desactivadas. Utilizar espnow como protocolo de comunicación. Si mode==0, el resto de los parámetros no deben ser establecidos.
Ejemplo	AT+SNIF=1,1

E.7.2 AT+PACKET

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+PACKET=<mode>[,<probe_time>] Función: Habilita o deshabilita la emisión de paquetes probe request el tiempo especificado
Respuesta	/r/nOK/r/n
Parámetros	<mode> 0: Deshabilita 1: Habilita <probe_time>: Tiempo en milisegundos entre el envío de cada paquete [5-7200]
Nota	Ninguna
Ejemplo	AT+PACKET=1,20

E.7.3 AT+SNIFREG

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SNIFREG=<mode>,<mac_id>,<mac> Función: Registra o elimina las direcciones MAC de los módulos que emiten probe request y que son las que se interesa detectar.
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Eliminar 1: Registrar <mac_id>: ID con la cual la dirección MAC será almacenada [0-5] <mac>: Dirección MAC del peer
Nota	Se pueden registrar hasta 6 direcciones MAC
Ejemplo	AT+SNIFREG=1,0,”00:01:00:01:00:01” (agregar) AT+SNIFREG=0,0 (eliminar)

E.7.4 AT+SNIFPR

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SNIFPR=<time> Función: Imprime todas las direcciones MAC de los módulos WiFi que emiten los paquetes probe request durante el tiempo establecido
Respuesta	/r/nOK/r/n
Parámetros	<time>: Tiempo en segundos en el que estará imprimiendo las direcciones MAC de los módulos WiFi
Nota	Ninguna
Ejemplo	AT+SNIFPR=5

E.7.5 AT+SNIFCT

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SNIFCT=<time> Función: Imprime el número de módulos WiFi que han emitido paquetes probe request durante el tiempo especificado.
Respuesta	/r/nOK/r/n
Parámetros	<time>: Tiempo en segundos en el que se detectarán los módulos WiFi que emiten los paquetes probe request y se realizará su contabilización.
Nota	Se puede detectar hasta 30 dispositivos.
Ejemplo	AT+SNIFCT=20

E.8 Servidor SSL

- El código de estos comandos están en el archivo *user_server.c* de la versión SDK NONOS.
- La información se recibe: /r/n+SSL,<len>:<datos>

E.8.1 AT+SSLSV

Firmware	**SDK NONOS**
Comando	AT+SSLSV= =<mode>,<port>,<timeout> Función: Crea un servidor SSL en el puerto especificado
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Elimina el servidor 1: Crea el servidor <port>: Puerto que habilitará el servidor [1- 65535] <timeout>: Tiempo en segundos que el servidor mantendrá conectado a un cliente inactivo [0-7200]
Nota	El servidor sólo puede tener un cliente Para eliminar el servidor, éste no debe tener clientes conectados Si <mode>==0 lo demás parámetros no deben ser declarados Si el <timeout> es cero, el cliente será quien deberá cerrar la conexión.
Ejemplo	AT+SSLSV=1,200,10 (Crear) AT+SSLSV=0 (Eliminar)

E.8.2 AT+SSLSEND

Firmware	**SDK NONOS**
Comando	AT+SVSEND =<length> Función: Envía datos del servidor SSL al cliente que esté conectado
Respuesta	Regresa > después de establecer el comando. Inicia a recibir datos a través del serial. Cuando la longitud de los datos definida es alcanzada, la transmisión de datos empieza. Si los datos son enviados de manera exitosa, regresa: /r/nSEND OK/r/n En caso contrario regresa: /r/nSEND FAIL/r/n
Parámetros	<length>: Longitud de datos a enviar [1-1000]
Nota	Ninguna

E.8.3 AT+SSLC

Firmware	**SDK NONOS**
Comando	AT+SSLC Función: Desconecta al cliente que éste conectado
Respuesta	/r/nOK/r/n
Parámetros	Ninguno
Nota	Ninguna
Ejemplo	AT+SSLC

E.9 Fingerprint

- El código de estos comandos están en el archivo *user_cripto.c* de la versión SDK NONOS.
- Si la conexión no se pudo llevar a cabo debido a un incorrecto *fingerprint,* se recibirá el mensaje "*wrong fingerprint*".

E.9.1 AT+FINGER

Firmware	**SDK NONOS**
Comando	AT+FINGER=<mode>[,<fingerprint>] Función: Habilita o deshabilita el uso del fingerprint para autentificar durante la conexión. Conforme a la longitud del fingerprint dado, se establecerá el método para calcular el fingerprint del certificado recibido durante la conexión.
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilitar 1: Habilitar <fingerprint>: String del fingerprint en formato hexadecimal sin caracteres de separación. El tamaño del fingerprint es el que determina el método que se aplicará al certificado recibido desde el servidor durante la negociación. String de 32 bytes será MD5 String de 40 bytes será SHA1 String de 64 bytes será SHA256
Nota	Si <mode>==0, el resto de los parámetros no deben ser establecidos
Ejemplo	AT+FINGER=1,”4e92bde81ab27334ec2c69b9b083ef7e422a5c71”

E.10 Criptográficos

- El código de estos comandos están en el archivo *user_cripto.c* de la versión SDK NONOS y user_cryto.c en ESP-IDF

E.10.1 AT+ENC

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+EN=<method>,<length> Función: Aplica el algoritmo especificado a los datos suministrados
Respuesta	Regresa > después de establecer el comando. Inicia a recibir datos a través del serial. Cuando la longitud de los datos definida es alcanzada, la aplicación del método empieza. Una vez terminado la aplicación del método: \r\n+ENC,<len>:<datos>
Parámetros	<method>: Algoritmo a aplicar: SHA1 SHA254 SHA256 SHA384 SHA512 MD5 AES ECB Encriptación (128,192,256) AES ECB Desencriptación (128,192,256) AES CBC Encriptación (128,192,256) AES CBC Desencriptación (128,192,256) ARC4 BLOWFISH ECB Encriptación BLOWFISH ECB Desencriptación BLOWFISH CBC Encriptación BLOWFISH CBC Desencriptación XTEA ECB Encriptación XTEA ECB Desencriptación XTEA CBC Encriptación XTEA CBC Desencriptación CAMELLIA ECB Encriptación (128,192,256) CAMELLIA ECB Desencriptación (128,192,256) CAMELLIA CBC Encriptación (128,192,256) CAMELLIA CBC Desencriptación (128,192,256) SDK NONOS V3: MD4 AES CFB128 Encriptación AES CFB128 Desencriptación AES CFB8 Encriptación AES CFB8 Desencriptación IDF: AES CFB128 Encriptación AES CFB128 Desencriptación AES CFB8 Encriptación AES CFB8 Desencriptación <length>: Número de datos a los que se les aplicará algún algoritmo.
Nota	Algoritmos validos [1-28] SDK NONOS V3 [1-19] SDK NONOS V2 [1-27] IDF

E.10.2 AT+IV

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+IV=<key> Función: Establece el vector de inicialización (IV) para los métodos que realizan un cifrado en bloques.
Respuesta	/r/nOK/r/n
Parámetros	<key>:String del vector de inicialización en formato hexadecimal y sin caracteres de separación,16 o 32 caracteres.
Nota	Esta llave es compartida por todos los métodos que la requieran Sólo se conservará la última llave establecida
Ejemplo	AT+IV=”00010001000100010001000100010001”

E.10.3 AT+VKEY

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+VKEY=<key> Función: Establece la llave utilizada para encriptar o desencriptar información para todos los métodos cuya llave no es de un tamaño fijo.
Respuesta	/r/nOK/r/n
Parámetros	<key>: String de la llave en formato hexadecimal y sin caracteres de separación, [2-116] caracteres
Nota	Esta llave es compartida por todos los métodos que la requieran Sólo se conservará la última llave establecida El tamaño de la llave debe ser múltiplo de 2 al ser hexadecimal.
Ejemplo	AT+VKEY=”0001”

E.10.4 AT+SKEY

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+SKEY=<key> Función: Establece la llave de encriptación o desencriptación de tamaño estándar (16,24,32 bytes)
Respuesta	/r/nOK/r/n
Parámetros	<key>: String de la llave en formato hexadecimal y sin caracteres de separación, [32,48,64] caracteres
Nota	Estás llaves son compartidas por todos los métodos que las requieran. Sólo se conservará la última llave de encriptación establecida Sólo se conservará la última llave de desencriptación establecida Algunos métodos, debido a su naturaleza, utilizan la llave de encriptación para desencriptar
Ejemplo	AT+SKEY=”0000000000000000000000000000000000000000000000000000000000000000”

E.11 PWM

- El código de estos comandos están en el archivo *user_pwm.c* de la versión SDK NONOS e IDF.
- En IDF ESP32 PWM está desarrollado sobre la API LED Control en modo LOW SPEED

E.11.1 AT+SCPWM

Firmware	**SDK NONOS V3 y ESP-IDF ESP8266**
Comando	AT+SCPWM=<pwm1>,<pwm1>,<pwm2>,<pwm3> Función: Establece los GPIO asociados a los canales PWM. Los GPIO deberán listarse conforme al canal PWM que se desea que ocupen
Respuesta	/r/nOK/r/n
Parámetros	<pwm1>: Número del GPIO que ocupará el canal PMW 0 <pwm2>: Número del GPIO que ocupará el canal PMW 1 <pwm3>: Número del GPIO que ocupará el canal PMW 2 <pwm4>: Número del GPIO que ocupará el canal PMW 3
Nota	GPIO válidos 4,12,14 y 15 Orden por defecto: GPIO12 canal 0, GPIO 15 canal 1, GPIO 14 canal 2 y GPIO 4 canal 3
Ejemplo	AT+SCPWM=4,12,14,15

E.11.2 AT+SPWM

Firmware	**SDK NONOS V3 y ESP-IDF ESP8266**
Comando	AT+SPWM=<period>,<channels>,<duty1>,<duty2>,<duty3>,<duty4> Función: Activa el o los PWM con los parámetros deseados
Respuesta	/r/nOK/r/n
Parámetros	<period>: Periodo en microsegundos de la señal (1000-10000) equivalente a (1KHz-100Hz) <channels>: Número de canales PWM a habilitar. Máximo 4. <duty1> Tiempo en alto del periodo PWM del canal 0 <duty2> Tiempo en alto del periodo PWM del canal 1 <duty3> Tiempo en alto del periodo PWM del canal 2 <duty4> Tiempo en alto del periodo PWM del canal 3
Nota	El periodo es de 1000 µs (1 KHz) ~ 10000 µs (100 Hz) Duty puede alcanzar como máximo period*1000/45 Todos los canales comparten el mismo periodo
Ejemplo	AT+SPWM=1000,2,100,85 AT+SPWM=1000,1,100

E.11.3 AT+PER

Firmware	**SDK NONOS V3 y ESP-IDF ESP8266**
Comando	AT+PER=<period> Función: Restablece el periodo para todos los canales PWM
Respuesta	/r/nOK/r/n
Parámetros	<period>: Periodo de la señal PWM en microsegundos [1000-10000]
Nota	Todos los canales comparten el mismo periodo
Ejemplo	AT+PER=1000

E.11.4 AT+DUTY

Firmware	**SDK NONOS V3 y ESP-IDF ESP8266**
Comando	AT+DUTY =<duty>,<channel> Función: Establece el duty de un canal PWM
Respuesta	/r/nOK/r/n
Parámetros	<duty>: Tiempo en alto de la señal <channel>: Canal PWM al que se asignará el duty [0-3]
Nota	Se apaga PWM con duty igual a cero Duty puede alcanzar como máximo period*1000/45
Ejemplo	AT+DUTY=100,0

E.11.5 AT+PHA

Firmware	**ESP-IDF ESP8266**
Comando	AT+PHA =<phase>,<channel> Función: Establece la phase de un canal PWM
Respuesta	/r/nOK/r/n
Parámetros	<phase>: [-180,180] <channel>: Canal PWM al que se asignará el duty [0-3]
Nota	Ninguna
Ejemplo	AT+PHA=100,0

E.11.6 AT+PWMTIMER

Firmware	**ESP-IDF ESP32**
Comando	AT+PWMTIMER =<timer>,<duty_re>,<frequency> Función: Establece el temporizador (LOW_SPEED) que será la fuente de oscilación con la resolución y la frecuencia dadas
Respuesta	/r/nOK/r/n
Parámetros	timer: Temporizador del módulo WiFi que será la fuente de oscilación para los canales PWM [0-3] duty_re: resolución (bits) para los ciclos de trabajo PWM [1-15] frequency: Frecuencia en Hertz (Max 40 MHz con 1 bit de resolución de ciclo de trabajo)
Nota	Los temporizadores pueden ser configurados de maneras diferentes y asignados a los diferentes canales PWM
Ejemplo	AT+PWMTIMER=1,5,4000

E.11.7 AT+PWMCHAN

Firmware	**ESP-IDF ESP32**
Comando	AT+PWMCHAN =<channel>,<GPIO>,<timer>, duty Función: Asocia el GPIO con el canal, temporizador y ciclo de trabajo especificado
Respuesta	/r/nOK/r/n
Parámetros	timer: Temporizador del módulo WiFi que será la fuente de oscilación PWM [0-3] channel: Canal PWM [0-7] gpio: GPIO que presentará la salida PWM duty: Establece el duty [0, (2**duty_resolution)]
Nota	Varios canales pueden tener el mismo temporizador Al ejecutarse esta función se activa la salida PWM
Ejemplo	AT+PWMCHAN=1,29,1,100

E.11.8 AT+PWMFRE

Firmware	**ESP-IDF ESP32**
Comando	AT+PWMFRE=<frequency>,<timer> Función: Cambia la frecuencia para el temporizador especificado
Respuesta	/r/nOK/r/n
Parámetros	<timer>: Temporizador del módulo WiFi que será la fuente de oscilación PWM [0-3] <frequency>: Frecuencia en Hertz (Max 40 MHz con 1 bit de resolución de ciclo de trabajo)
Nota	Ninguna
Ejemplo	AT+PWMFRE=10000,1

E.11.9 AT+PWMDUTY

Firmware	**ESP-IDF ESP32**
Comando	AT+PWMDUTY=<channel>,<duty> Función: Cambia el duty de un canal PWM
Respuesta	/r/nOK/r/n
Parámetros	duty: Tiempo en alto de la señal channel: Canal PWM al que se asignará el duty [0-7]
Nota	Ninguna
Ejemplo	AT+PWMDUTY=1, 200

E.12 Status led

- El código de estos comandos están en el archivo *user_status_led.c* de la versión SDK NONOS.

E.12.1 AT+STALED

Firmware	**SDK NONOS V3**
Comando	AT+STALED=<mode>[,<gpio>] Función: Habilita o deshabilita un GPIO del módulo WiFi para indicar mediante un LED el estado de la conexión a un punto de acceso. Éste parpadeará durante el proceso de conexión y si ésta fue exitosa dejará de parpadear
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 0: Deshabilita 1: Habilita <gpio>: Número del GPIO utilizado para status led, GPIO 4, 5, 9, 10, 12, 13, 14, 15
Nota	Si <mode>==0, el resto de los parámetros no deben ser declarados
Ejemplo	AT+STALED=1,9 (Habilitar) AT+STALED=0

E.13 Cliente especial

- El código de estos comandos están en el archivo *user_client.c* de la versión SDK NONOS.

E.13.1 AT+SPEC

Firmware	**SDK NONOS V3**
Comando	AT+SPEC=<type>,<filter>,<ip>,<port> Función: Crea un cliente TCP o SSL persistente con el filtro establecido
Respuesta	/r/nOK/r/n
Parámetros	<filter>: Filtro aplicado a los mensajes recibidos 0: Ninguno 1: GET <type>: Tipo de conexión 0: TCP 1: SSL <ip>:String de la dirección IP o el nombre de dominio del servidor <port>: Puerto habilitado por el servidor
Nota	Sólo puede crearse un único cliente con este comando
Ejemplo	AT+SPEC=1,0,”192.168.4.1”,20

E.13.2 AT+SPECS

Firmware	**SDK NONOS V3**
Comando	AT+SPECS=<filter>,<length> Función: Envía datos al servidor. El filtro seleccionado sólo se aplicará al primer mensaje recibido tras la ejecución de esta función
Respuesta	Regresa > después de establecer el comando. Inicia recibir datos a través del serial. Cuando la longitud de los datos definida es alcanzada, la transmisión de datos empieza. Si los datos son enviados de manera exitosa, regresa: /r/nSEND OK/r/n En caso contrario regresa: /r/nSEND FAIL/r/n
Parámetros	filter: Filtro aplicado a los mensajes recibidos 0: Ninguno 1: GET <length>: Longitud del mensaje a enviar [1-1000]
Nota	Ninguna
Ejemplo	AT+SPECS=1,4

E.13.3 AT+SPECC

Firmware	**SDK NONOS V3**
Comando	AT+SPECC Función: Cierra la conexión con el servidor
Respuesta	/r/nOK/r/n
Parámetros	Ninguno
Nota	Ninguna
Ejemplo	AT+SPECC

E.14 GPIO

- El código de estos comandos están en el archivo *user_gpio.c* de la versión ESP-IDF.

E.14.1 AT+CFGGPIO

Firmware	**ESP-IDF**
Comando	AT+CFGGPIO=<pin>,<mode>,<pull> Función: Configura el modo de trabajo del GPIO
Respuesta	/r/nOK/r/n
Parámetros	<pin>: Número del GPIO <mode>: 0: input 1: output <pull>: 0: pull-up 1: pull.down 2: ninguno
Nota	Ninguna
Ejemplo	AT+CFGGPIO=3,0,0

E.14.2 AT+SYSGPIOWRITE

Firmware	**ESP-IDF**
Comando	AT+SYSGPIOWRITE=<pin>,<level> Función: Configura el nivel de salida del GPIO
Respuesta	/r/nOK/r/n
Parámetros	<pin>: Número del GPIO <level>: 0: low 1: high
Nota	Ninguna
Ejemplo	AT+SYSGPIOWRITE=3,0

E.14.3 AT+SYSGPIOREAD

Firmware	**ESP-IDF**
Comando	AT+SYSGPIOREAD=<pin> Función: Lee el nivel de voltaje presente en el GPIO
Respuesta	+SYSGPIOREAD:<gpio>,<dir>,<state>
Parámetros	<pin>: Número del GPIO
Nota	Ninguno
Ejemplo	AT+SYSGPIOREAD=3

E.15 Utilidades

E.15.1 AT+APS

- El código de este comando está en el archivo *user_aps.c* de la versión SDK NONOS y ESP-IDF.

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+APS=<ssid1>,<ssid2>[,<ssid3>] Función: Realiza un escaneo de los AP cercanos y regresa el ID del que tiene el RSSI más fuerte.
Respuesta	+STA:<id>
Parámetros	<ssid1>: Nombre del primer AP (máx 20 caracteres) <ssid2>: Nombre del segundo AP (máx 20 caracteres) <ssid3>: Nombre del tercer AP (máx 20 caracteres)
Nota	Ninguna
Ejemplo	AT+APS=”red1”,”red2”

E.15.2 AT+DSTA

- El código de este comando está en el archivo user_sniffer.c de la versión SDK NONOS

Firmware	**SDK NONOS**
Comando	AT+DSTA=<mac> Función: Desconecta la estación con la dirección MAC especificada del AP creado por el módulo
Respuesta	/r/nOK/r/n
Parámetros	<mac>: Dirección MAC de la estación a desconectar
Nota	Ninguna
Ejemplo	AT+DSTA=”00:01:00:01:00:01”

E.15.3 AT+POWER

- El código de este comando está en el archivo user_rssi.c de la versión SDK NONOS y ESP-IDF.

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+POWER=<power> Función: Establece el valor máximo de potencia RF TX
Respuesta	/r/nOK/r/n
Parámetros	<power>: valor [0,82] Unidades 0.25 dBm (SDK NONOS) <power>: valor [40,82] Unidades 0.25 dBm (ESP_IDF) correspondiente a un rango de 10dBm - 20.5dBm
Nota	Ninguna
Ejemplo	AT+POWER=25

E.15.4 AT+MODE

- El código de este comando está en el archivo user_rssi.c de la versión SDK NONOS e ESP-IDF.

Firmware	**SDK NONOS y ESP-IDF**
Comando	AT+MODE=<mode> Función: Establece el modo WiFi (802.11b/g/n)
Respuesta	/r/nOK/r/n
Parámetros	<mode>: 1==802.11b 2==802.11g 3==802.11n (SDK NONOS) 1==802.11b 2==802.11bg 3==802.11bgn (ESP-IDF)
Nota	Ninguna
Ejemplo	AT+MODE=2

E.16 ADC

- El código de este comando está en el archivo user_adc.c de la versión ESP-IDF

E.16.1 AT+ADC

Firmware	**ESP-IDF ESP32**
Comando	AT+ADC=<state>,<bits> Función: Habilita o deshabilita ADC y establece la resolución
Respuesta	/r/nOK/r/n
Parámetros	<state> 0: Deshabilita 1: Habilita <bits>: 0: ADC_WIDTH_BIT_9 1: ADC_WIDTH_BIT_10 2: ADC_WIDTH_BIT_11 3: ADC_WIDTH_BIT_12
Nota	Está basado sobre ADC1 (8 canales correspondientes a los GPIOs 32 - 39)
Ejemplo	AT+ADC=1,1

E.16.2 AT+SYSADCO

Firmware	**ESP-IDF ESP32**
Comando	AT+SYSADCO=<channel>,<attenuation> Función: Configura el canal ADC para ser leído y establece la atenuación de éste
Respuesta	/r/nOK/r/n
Parámetros	<channel > Canal ADC [0-7] <bits>: Atenuación 0: ADC_WIDTH_BIT_9 1: ADC_WIDTH_BIT_10 2: ADC_WIDTH_BIT_11 3: ADC_WIDTH_BIT_12
Nota	Ninguna
Ejemplo	AT+SYSADCO=1,1

E.16.3 AT+SYSADC

Firmware	**ESP-IDF**
Comando	AT+SYSADC=<channel> Función: Obtiene valor ADC del canal especificado
Respuesta	+SYSADC:<value>\r\n
Parámetros	<channel > Canal ADC [0-7]
Nota	Ninguna
Ejemplo	AT+SYSADC=1