UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

PROPUESTA DE PRÁCTICAS PARA EL LABORATORIO DE DISEÑO DE SISTEMAS CON MICROPROCESADORES

TESIS

Que para obtener el título de

INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

P R E S E N T A

JOSÉ ARTURO JIMÉNEZ AGUILAR

DIRECTOR DE TESIS:

Ing. OSCAR GUADALUPE MORENO ESPINOZA

Ciudad Nezahualcóyotl, Estado de México, 2022

Dedicatoria

A mi padre José, por su apoyo y por no dejarme solo en estos 4 años de mi carrera universitaria.

A mi tía Matilde y mis primas Marian y Johana, que me apoyaron en los días más difíciles, brindándome su apoyo incondicional.

A mis amigos de la Facultad Samuel, Uriel, Jorge, Eduardo, Zolache y Darién, por su apoyo día a día en clases, trabajos, etc.

A mi amiga Silvia, sus consejos, apoyo y reprimendas me ayudaron cada día más a superarme y seguir tratando de ser mejor.

A mi amiga Areli, por su apoyo en los momentos más difíciles y su gran entusiasmo me ayudaron a no dejar de seguir intentándolo.

Agradecimientos

A la Universidad Nacional Autónoma de México, en especial a la Facultad de Estudios Superiores Aragón, que me brindo las herramientas necesarias para que pudiera realizarme como profesionista, a los profesores que me apoyaron en todas las dudas que tenía y siempre estuvieron en la disposición de ayudarme.

A mi asesor el Ingeniero Oscar Guadalupe Moreno Espinoza, por darme la oportunidad de realizar este trabajo y por todo su apoyo que me brindo para realizar el mismo.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Resumen

El siguiente trabajo es un compendio de prácticas desarrolladas con el objetivo de que los alumnos del laboratorio de diseño de sistemas con microprocesadores puedan tener un manual para así cursar el laboratorio

Las practicas están basadas en la placa de desarrollo ESP32 usando el IDE de Arduino para su programación, empezando por como instalarlo y configurarlo, después trabajando desde lo más básico como lo son el uso de puertos como entrada y salida de datos, comunicación entre la placa y la computadora, a medida que avanzan las prácticas se pretende ver el potencial que tiene el ESP32 en el tema de comunicación inalámbrica Bluetooth y Wi-Fi, por ejemplo, como conectarlo a un smartphone con Android y recibir o enviar información atraves de él y en las practicas finales se menciona un par de las muchas aplicaciones que puede tener el ESP32, comunicación entre el ESP32 y la aplicación de mensajería Telegram y, comunicación entre sensores usando el ESP32 y un servidor en la nube llamado ThingSpeak.

Capítulo 1

Introducción

El constante avance de circuitos integrados SoC (*System on a Chip*) que incluyen capacidades de conexión inalámbrica como Bluetooth y Wi-Fi, avanza muy rápidamente por la necesidad de intercambiar información en redes computacionales como el Internet. Un claro ejemplo es el surgimiento del internet de las cosas o *Internet of thing* (Iot) demandan cada vez más productos de bajo coste, bajo consumo, pequeños y con funcionalidades inalámbricas, como lo son: bombillas inteligentes, apagadores inteligentes, cerraduras inteligentes, etc. Todos estos aparatos llevan en su interior un chip ya preparado con funciones inalámbricas Wi-Fi o Bluetooth.

El estudio de estos nuevos chips es de vital importancia para el campo de la electrónica, el constante avance del internet de las cosas hace que estos chips sean cada vez más usados para aplicaciones en el hogar, edificios inteligentes o estaciones meteorológicas. Aunque la comunicación por Wi-Fi no cumple al cien por ciento de las especificaciones del IoT, su presencia inherente en casi cualquier lugar ha hecho que sea una de las maneras más populares y de fácil acceso para hacer una comunicación inalámbrica entre varios dispositivos.

Cuando se habla de placas de desarrollo de electrónica para el uso en la domótica o el internet de las cosas casi siempre se nos viene a la mente plataformas como Arduino o Raspberry Pi, dejando de lado otras opciones que pueden ser más viables, por ejemplo, la característica del proyecto, el costo o incluso la complejidad al momento de su programación, no digo que las plataformas anteriormente mencionadas no deban de implementarse, sino más bien, tener conocimiento de que existen otras placas de desarrollo que pueden cumplir con nuestras necesidades de una manera más óptima.

Justificación

Actualmente a la creación de este documento, el laboratorio de Diseño de Sistemas con Microprocesadores de la FES Aragón no se cuenta con un manual de prácticas, por lo cual se decidió en compañía de mi asesor realizar un compendio de prácticas basadas en la placa de desarrollo ESP32 con el fin de que los alumnos y/o profesores que imparten el mismo laboratorio, así como la teoría, tengan una base la cual les pueda ser de ayuda para complementar la enseñanza de la asignatura.

Se opto por realizar este manual en base a la placa de desarrollo ESP32 porque es un dispositivo de bajo coste y de gran accesibilidad disponible en las principales tiendas online en México como Mercado Libre o internacionales como Aliexpress. Dispone de características que lo hace diferenciarse de otras placas de desarrollo como lo son Arduino o RaspberryPi, una de las más importantes es que en su interior contiene un módulo Wi-Fi y Bluetooth haciéndolo ideal para una comunicación inalámbrica entre dispositivos inteligentes como pueden ser smartphones.

Cumple con lo estipulado en el plan de estudios de la asignatura, concretamente en el último tema llamado “Nuevas tecnologías con microprocesadores” porque esta placa de desarrollo tiene en su interior un circuito integrado SoC (*System on a Chip*), una tecnología con un potencial para el entorno del IoT que se espera sea el futuro de los próximos dispositivos inteligentes.

Objetivo

Desarrollar una serie de prácticas que ayuden al alumno y/o profesores a tener una base para complementar el temario del laboratorio de Diseño de Sistemas con Microprocesadores de la carrera de Ingeniera Eléctrica y Electrónica utilizando la placa de desarrollo ESP32.

Objetivos específicos

- Elaborar un compendio de prácticas a manera de guía para el laboratorio de Diseño de Sistemas con Microprocesadores
- Fomentar el uso de la placa de desarrollo ESP32 para futuros proyectos de electrónica
- Promover el uso de las tecnologías de comunicación inalámbrica Bluetooth y Wi-Fi para el internet de las cosas
- Observar las aplicaciones, limitaciones y alcances de las comunicaciones inalámbricas Bluetooth y Wi-Fi en la transmisión de datos para el internet de las cosas.
- Aplicar técnicas de programación usando el IDE de Arduino

Capítulo 2

Marco Teórico

Antes de que la placa de desarrollo del ESP32 fuera creada y usada como lo es ahora, existían otras alternativas que utilizaban una versión anterior al ESP32, era el ESP8266, una versión muy inferior con características limitadas y que era comúnmente usado junto con Arduino, ambos dispositivos se conectaban y configuraban para dotar al Arduino una conectividad inalámbrica Wi-Fi, pero si se requería tener también tener conectividad Bluetooth se necesitaba de otro dispositivo, un módulo Bluetooth que podría ser el HC-05, HC-06 o At-09.

A continuación, se mostrarán propuestas, ejercicios o proyectos que muestran cómo es que todos los dispositivos anteriormente descritos eran utilizados, programados y configurados.

Home automation using arduino Wi-Fi module ESP8266

"This project presents a design and prototype implementation of new home automation system that uses Wi-Fi technology as a network infrastructure connecting its parts. The proposed system consists of two main components; the first part is the server (web server), which presents system core that manages, controls, and monitors users' home. Users and system administrator can locally (LAN) or remotely (internet) manage and control system code. Second part is hardware interface module, which provides appropriate interface to sensors and actuator of home automation system. Unlike most of available home automation system in the market the proposed system is scalable that one server can manage many hardware interface modules as long as it exists on Wi-Fi network coverage. System supports a wide range of home automation devices like power management components, and security components. The proposed system is better from the scalability and flexibility point of view than the commercially available home automation systems."[1]

En español: “Este proyecto presenta un diseño y prototipo de implementación de un nuevo sistema domótico que utiliza la tecnología Wi-Fi como infraestructura de red conectando sus partes. El sistema propuesto consta de dos componentes principales; la primera parte es el servidor (servidor web), que presenta el núcleo del sistema que administra, controla y monitorea el hogar de los usuarios. Los usuarios y el administrador del sistema pueden administrar y controlar el código del sistema localmente (LAN) o remotamente (internet). La segunda parte es el módulo de interfaz de hardware, que proporciona una interfaz adecuada para los sensores y actuadores del sistema de automatización del hogar. A diferencia de la mayoría de los sistemas de automatización del hogar disponibles en el mercado, el sistema propuesto es escalable y un servidor puede administrar muchos módulos de interfaz de hardware siempre que exista cobertura de red Wi-Fi. El sistema es compatible con una amplia gama de dispositivos de automatización del hogar, como componentes de administración de energía y componentes de seguridad. El sistema propuesto es mejor desde el punto de vista de escalabilidad y flexibilidad que los sistemas domóticos disponibles comercialmente”

El anterior trabajo plantea una propuesta de cómo usar el ESP8266 y el Arduino con el propósito de diseñar un nuevo sistema domótico, en este caso solo exploran la posibilidad de como conectar a la salida del Arduino un control de relevadores para prender y apagar focos o que se pueda utilizar para prender o apagar aparatos, también da una breve introducción de cómo manejarlo mediante una página web en HTML y da una base de como este proyecto puede ser fácilmente escalable para usar otro tipo de actuadores o sensores, al ser un trabajo de investigación del 2015 presenta un poco de complicación a la hora de programar todo, una de las principales que note fue que el módulo ESP8266 tiene que ser cargado con un Firmware para que pueda ser usado en su modalidad Wi-Fi y que se pueda comunicar con el Arduino, haciendo necesario adquirir un programador para el mismo, si bien no es complicado, presenta una doble programación y más tiempo de preparación para que todo funcione correctamente.

Comunicación Bluetooth entre Arduino UNO y Android aplicado a un detector de mentiras

“Vamos a realizar un proyecto en el que utilizaremos una placa Arduino UNO, un sensor de ritmo cardiaco llamado Pulse Sensor, un módulo Bluetooth para la placa Arduino llamado HC-05 y además desarrollaremos una aplicación para el sistema operativo Android. Con todo esto lo que pretendemos es realizar un detector de mentiras. Calcularemos con Arduino las pulsaciones que tiene la persona y las enviaremos vía Bluetooth a un teléfono móvil que tenga la aplicación, y esta será la encargada de verificar si la persona está diciendo una verdad o una mentira” [2]

En este proyecto nos plantea un propuesta de cómo podemos combinar el Arduino, un módulo Bluetooth HC-05 y un sensor de ritmo cardiaco para crear un detector de mentiras y que pueda mostrar esa información en una app con un celular Android, un proyecto que trata de usar varias ideas en conjunto con un fin en específico, si bien la idea y la implementación es buena, de igual manera observamos que la opción de dotar al Arduino de capacidades Bluetooth es la utilización de un módulo por aparte, que si bien para este propósito no representa mucho problema, sería más optimizado si todo estuviera en un solo chip.

Módulo ESP8266 y sus aplicaciones en el internet de las cosas

“En la actualidad existen infinidad de dispositivos electrónicos que interactúan con nosotros en nuestra vida cotidiana y la necesidad de comunicarnos con ellos, de compartir información y realizar funciones a distancia mediante el uso de internet es cada vez mayor. Para lograr esta conexión, es necesario el uso de una interfase entre estos dispositivos y el mundo del internet. Por lo que esta investigación nos llevara a conocer más a fondo un módulo Wi-Fi ESP8266 de bajo costo, ideal para aplicaciones de internet de las cosas (IOT) y saber cómo elegir el más adecuado para nuestros futuros proyectos, enfocados en el internet de las cosas. Se obtuvo como resultado de este análisis, que existen factores importantes para la elección de estos dispositivos, como lo son la normativa de comunicación en cada país, la cantidad de memoria y sobre todo el costo beneficio, dependiendo del hardware con que se cuenta en el proyecto. Concluimos que antes de adquirir un módulo ESP8266 debemos realizar un análisis, tomando en cuenta estos factores para una correcta elección.” [3]

Este trabajo es más bien un análisis a profundidad de cómo se puede utilizar el módulo ESP8266 en el Internet de las cosas para proyectos, analizando cada una de las propuestas de diferentes fabricantes, costos, características, normativas, etc. Al ser un análisis solamente de las placas de desarrollo en que están basados en el ESP8266 se puede concluir que dependiendo de cuál sea el proyecto se puede elegir una amplia gama de opciones, pero casi en la mayoría presenta una limitante fundamental, necesita la ayuda de un microcontrolador externo para que funcione de una manera más optima o para ampliar la capacidad de ejecución de tareas más pesadas.

Internet of Things with ESP8266

"The Internet of Things (IoT) is the network of objects such as physical things embedded with electronics, software, sensors, and connectivity, enabling data exchange. ESP8266 is a low cost Wi-Fi microcontroller chip that has the ability to empower IoT and helps the exchange of information among various connected objects. ESP8266 consists of networkable microcontroller modules, and with this low cost chip, IoT is booming. This book will help deepen your knowledge of the ESP8266 Wi-Fi chip platform and get you building exciting projects.

Kick-starting with an introduction to the ESP8266 chip, we will demonstrate how to build a simple LED using the ESP8266. You will then learn how to read, send, and monitor data from the cloud. Next, you'll see how to control your devices remotely from anywhere in the world. Furthermore, you'll get to know how to use the ESP8266 to interact with web services such as Twitter and Facebook. In order to make several ESP8266s interact and exchange data without the need for human intervention, you will be introduced to the concept of machine-to-machine communication.

The latter part of the book focuses more on projects, including a door lock controlled from the cloud, building a physical Bitcoin ticker, and doing wireless gardening. You'll learn how to build a cloud-based ESP8266 home automation system and a cloud-controlled ESP8266 robot. Finally, you'll discover how to build your own cloud platform to control ESP8266 devices.

With this book, you will be able to create and program Internet of Things projects using the ESP8266 Wi-Fi chip"[4]

En español: "El Internet de las cosas (IoT) es la red de objetos, como cosas físicas integradas con electrónica, software, sensores y conectividad, que permiten el intercambio de datos. ESP8266 es un chip de microcontrolador Wi-Fi de bajo costo que tiene la capacidad de potenciar IoT y ayuda al intercambio de información entre varios objetos conectados. ESP8266 consta de módulos de microcontroladores que se pueden conectar en red, y con este chip de bajo costo, IoT está en auge. Este libro lo ayudará a profundizar su conocimiento de la plataforma del chip Wi-Fi ESP8266 y lo ayudará a construir proyectos emocionantes.

Comenzando con una introducción al chip ESP8266, demostraremos cómo construir un LED simple usando el ESP8266. Luego aprenderá a leer, enviar y monitorear datos desde la nube. A continuación, verá cómo controlar sus dispositivos de forma remota desde cualquier parte del mundo. Además, aprenderá a usar el ESP8266 para interactuar con servicios web como Twitter y Facebook. Para hacer que varios ESP8266 interactúen e intercambien datos sin necesidad de intervención humana, se le presentará el concepto de comunicación de máquina a máquina.

La última parte del libro se centra más en los proyectos, incluida la cerradura de una puerta controlada desde la nube, la construcción de un ticker físico de Bitcoin y la jardinería inalámbrica. Aprenderá a construir un sistema de automatización del hogar ESP8266 basado en la nube y un robot ESP8266 controlado por la nube. Finalmente, descubrirá cómo crear su propia plataforma en la nube para controlar dispositivos ESP8266.

Con este libro, podrá crear y programar proyectos de Internet de las cosas utilizando el chip Wi-Fi ESP8266."

Básicamente es un libro donde explica desde cero como programar el ESP8266, y utilizarlo en proyectos para el Internet de las cosas, es un buen libro ya que te explica cosas sencillas hasta más complejas paso por paso, el inconveniente que le veo es que tienes que comprar el libro para poder tener acceso al libro completo, es bueno ya que así podemos

ayudar al autor pero para un estudiante no siempre se cuenta con el recurso económico, aunque sigue siendo un buen manual de como programar el ESP8266 desde cero.

Creación de biblioteca Wi-Fi ESP para microcontroladores PIC

"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 Wi-Fi 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 reducción 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."[5]

Como se mencionó en la justificación de este trabajo, el laboratorio de Diseño de Sistemas con Microprocesadores de la FES Aragón no cuenta con un manual de prácticas ni mucho menos basado en la placa de desarrollo ESP32, pero en el caso de la facultad de Ingeniería en Ciudad Universitaria de la UNAM, se tiene una propuesta recién publicada de prácticas usando una versión del ESP8266 llamada ESP-01 pero está enfocada en dos cosas diferentes, primero, la propuesta se realizó como material didáctico para la asignatura de Circuitos Digitales y segundo, su planteamiento está en que el ESP01 sea un complemento de un PIC que cumpla con los requisitos mínimos para usar la librería, en su caso utilizaron el PIC 16LF1939 para hacer las pruebas y desarrollar el manual de prácticas.

Si bien el trabajo muestra una excelente investigación y desarrollo como lo es la creación de una librería para poder usarla con un PIC, el problema sigue siendo que el módulo

ESP-01 necesita de otro microcontrolador para poder realizar cosas más complejas y de otro modulo para su programación.

Toda esta información recopilada de sistemas anteriores refuerza la propuesta planteada al usar el ESP32 como base para desarrollar un manual de prácticas, porque al estar todo integrado en un solo chip y en una sola placa, no se necesita de más componentes externos para poder programarla, haciéndola idónea para proyectos ya sea de investigación o propuestas para dar solución a una problemática.

Marco Conceptual

Como este trabajo es una propuesta de prácticas para un laboratorio, la información más detallada se encontrara en cada una de ellas, por lo cual en el marco conceptual solo se mencionaran de manera breve unos términos que se consideran importantes, como el hardware IoT, protocolos de comunicación IoT y un pequeño vistazo a como está construida la placa de desarrollo existentes basados en el ESP32 y la que se ocupó para el desarrollo de estas prácticas, por lo cual una explicación un poco más detallada se verá en dichas prácticas o también se verá en forma de preguntas de investigación que el alumno tendrá que realizar para complementar la información o investigación que se realizó en la introducción de cada práctica.

Hardware IoT

En el mercado existen muchas plataformas que nos pueden servir para un proyecto IoT, el libre por excelencia es Arduino como microcontrolador y Raspberry pi como microprocesador, pero no solo debemos quedarnos con esas dos opciones, a continuación de muestra un listado de algunos dispositivos que son libres y podemos usar sin ningún problema para el entorno IoT:[6]

- Arduino:
- Arduino familia MKR para IoT
- ESP8266, HW low cost con Wi-Fi
- ESP32
- NodeMCU
- Wemos D1 Mini
- Moteino
- Artik de Samsung
- Waspmote
- Particle
- Geothings:
- IoT de Texas Instrument

- RIoTboard
- Goblin 2
- Gemalto, conectividad IoT:
- Renexa RX111 MCUs
- Flutter
- WavIoT
- Tessel placa IoT con node.js
- Enocean (Ultra low power y power harvesting)

Protocolos de comunicación inalámbricos más comunes en IoT

Protocolo Bluetooth Clásico: IEEE 802.15.1

Utilizado principalmente para conexiones que se requieren de una sincronización de datos continua como por ejemplo unos audífonos inalámbricos, la música tiene que transmitirse de manera ininterrumpida, la última versión del Bluetooth se llama 5.2 y opera en 79 canales con una banda de 1MHz.[7]

Protocolo Wi-Fi IEEE 802.11 a/b/g/n

Utiliza dos bandas ISM gratuitas, 2.4 GHz y 5.8 GHz, pero la mayoría de las aplicaciones de Internet de las cosas utilizan la banda 2.4 GHz dado que su rango de alcance en mucho mayor que el de 5.8 GHz.[7]

Protocolo ZigBee IEEE 802.15.4

Se desarrollo en 2004 como una alternativa al Wi-Fi y Bluetooth para aplicaciones de bajo consumo, apenas alcanza una tasa de transferencia de bits de 20-250kbit, muy baja, pero es suficiente para aplicaciones IoT.[7]

Protocolo LoRaWAN

Es un protocolo de largo alcance que se utiliza para las redes de área amplia de bajo consumo, yendo de unos cuantos metros a 10 kilómetros, tiene más de 10000 puertas de enlace y más de 110000 miembros en la comunidad.[7]

ESP32

El ESP32 es un sistema en un chip (*System On a Chip*) circuito integrado que integra todos o algunos de los componentes de una computadora u otro sistema electrónico, comúnmente se integra con un CPU, memoria, puertos de entrada-salida y un almacenamiento secundario, aunque un SoC también puede incluir módems de radio o una unidad de procesamiento gráfico. [8], [9]

El SoC del ESP32 por su parte integra las siguientes características:[10]

- Wi-Fi (2.4GHz)
- Bluetooth
- Procesador dual Xtensa® 32-bit LX6
- Coprocesador Ultra Low Power
- Múltiples periféricos.

El chip del ESP32 es diseñado por Espressif System pero quien se encarga de la fabricación es la empresa TSMC.[11]

Ilustración 1: SoC del ESP32

Existen varias familias del chip del ESP32, cada una tiene características diferentes o componentes más actualizados, en la siguiente tabla de nombran las más importantes:

Ordering code	Core	Embedded flash	Package
ESP32-D0WD-V3	Dual core	No embedded flash	QFN 5*5
ESP32-D0WDQ6-V3 (NRND)	Dual core	No embedded flash	QFN 6*6
ESP32-D0WD (NRND)	Dual core	No embedded flash	QFN 5*5
ESP32-D0WDQ6 (NRND)	Dual core	No embedded flash	QFN 6*6
ESP32-U4WDH	Dual core[1]	4 MB embedded flash (80 MHz)	QFN 5*5
ESP32-S0WD	Single core	No embedded flash	QFN 5*5
Note: All above chips support Wi-Fi b/g/n + Bluetooth/Bluetooth LE Dual Mode connection.			

Tabla 1: Diferentes modelos del SoC del ESP32

La nomenclatura de cada chip se puede saber por el siguiente esquema:

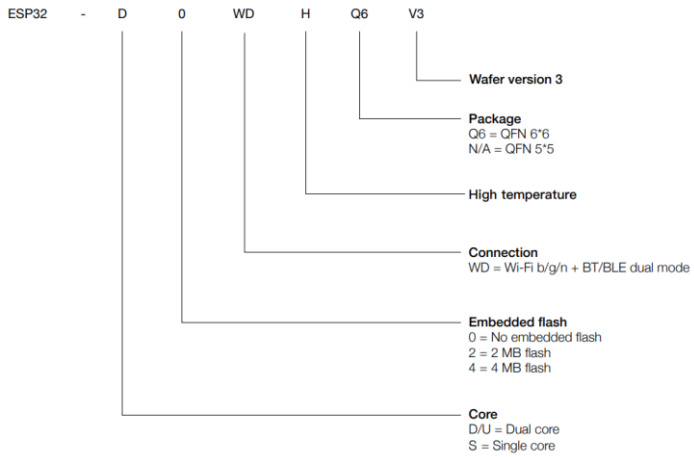


Ilustración 2: Nomenclatura de los diferentes Soc´s del ESP32

Cada chip es montado en un módulo acompañado de más componentes como un cristal de cuarzo, una memoria SPI, etc. Los diferentes módulos fabricados son demasiados, pero nombraremos los más importantes a continuación:[12]

ESP32-WROOM-32D

Ilustración 3: Modulo ESP32-WROOM-32D

Este es el módulo más popular que integra el integrado ESP32-D0WD, la razón de su popularidad es su adaptabilidad, y la variedad de aplicaciones que se pueden hacer con el chip, tiene mejoras en los sensores para un consumo menor de energía y transmisión de música.

ESP32-WROOM-32

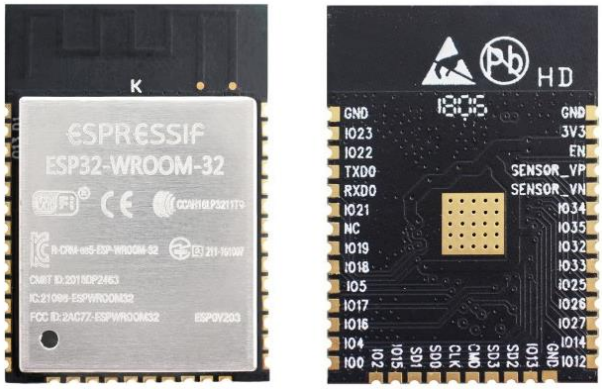


Ilustración 4: Modulo ESP32-WROOM-32

EL ESP32-WROM-32 es el módulo original del ESP32 que trajo la compañía Espressif. Es la primera versión del ESP32 por lo que contiene el chip ESP32-D0WDQ6 diseñado para ser escalable y adaptable.

Si bien fue el primer módulo y el que inicio la serie de módulos con muchas mejoras o características, ya no se recomienda para diseños nuevos y su producción ha sido lentamente sustituida por el modelo anterior mencionado.

ESP32-WROOVER

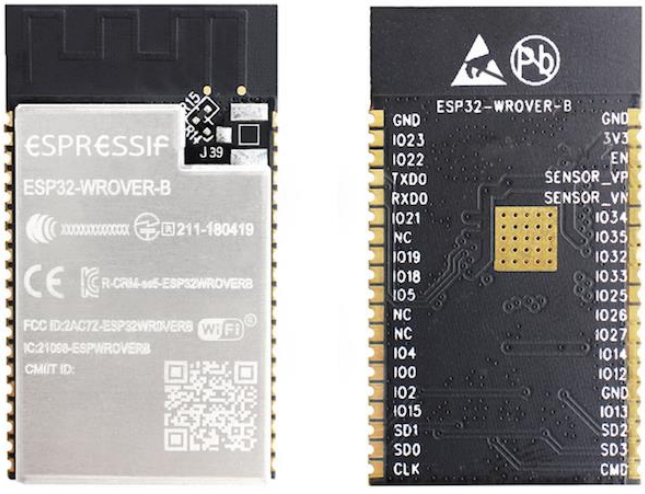


Ilustración 5: Modulo ESP32-WROOVER

Este módulo viene en dos versiones, uno con un espacio en la PCB para conectar una antena y otro con un conector IPEX para una antena externa de un adaptador Wi-Fi o de un modem.

Como el módulo anterior también contiene el chip ESP32-D0WDQ6, pero a diferencia del anterior este tiene más puertos que pueden ser usados externamente y la frecuencia del CPU se puede ajustar desde los 80MHz a los 240Mhz entre otras características.

Estos son módulos que ya integran las diferentes versiones del chip ESP32, pero como se puede observar en las imágenes no tienen pines para conectarlo a una Protoboard, reguladores de voltaje para su alimentación o un chip y conector para su programación a través de una computadora, por lo que se estos módulos son montados en placas de desarrollo que integran las características anteriormente mencionadas y más.

Estas placas de desarrollo pueden o no ser fabricadas por Espressif, ya que cada fabricante es libre de hacer el diseño que mejor le convenga.

Los más populares que se pueden encontrar en el mercado son:

ESP32 CAM

Ilustración 6: Placa de desarrollo ESP32 CAM

Es una placa de desarrollo que integra una cámara modelo OV2640 de 2 megapíxeles y con un conector para tarjetas Micro SD aunque la desventaja es que solo posee 9 pines GPIO, es comúnmente usada como cámara de seguridad gracias a su conexión Wi-Fi.

ESP32-DevKitC

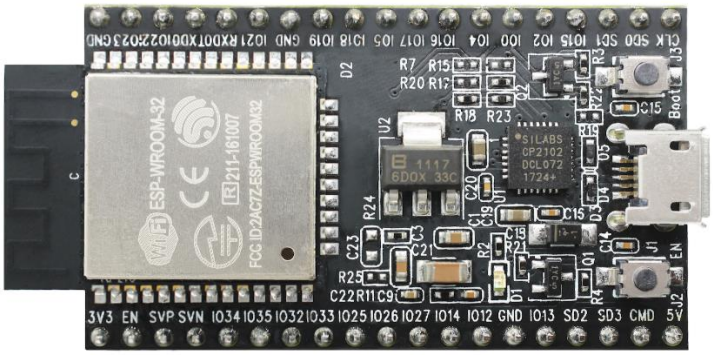


Ilustración 7: Placa de desarrollo ESP32-DevKitC

Es la placa de desarrollo fabricada por Espressif, una de las populares y más usadas en el mercado, su diseño con Headers machos hacia abajo hace posible su montaje en una Protoboard o su conexión con jumpers hembra.

Posee todo lo necesario para su fácil programación, tiene un puerto micro-USB, un chip de comunicación para el módulo y una computadora, un regulador de voltaje y botones de reseteo.[12]

HUZZAH32

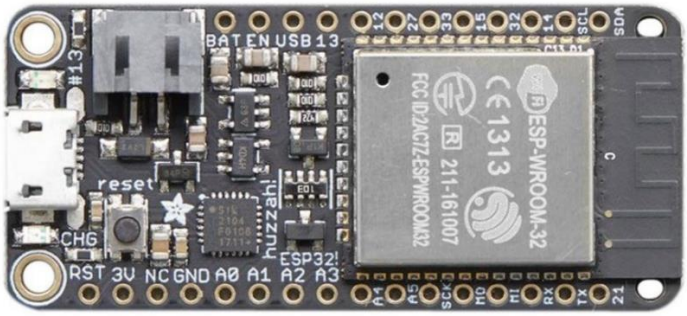


Ilustración 8: Placa de desarrollo HUZZAH32

Referida como “feather board” por Adafruit, es una placa de desarrollo favorita para los iniciantes en Iot, ya que contiene características especiales que lo hacen ideal para proyectos de IoT como lo son un conector para baterías de polímeros de litio o iones de litio, tamaño reducido para un montaje más compacto, etc.[13]

Node-MCU-32S

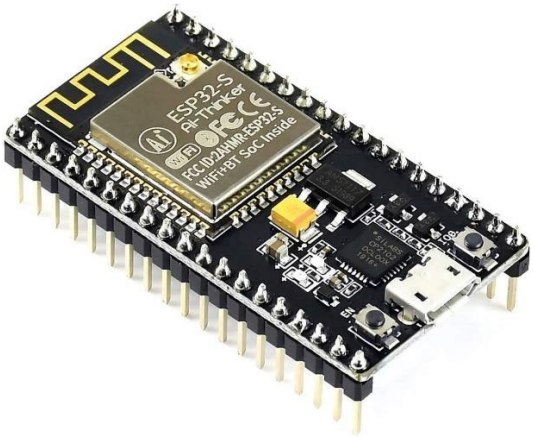


Ilustración 9: Placa de desarrollo Node-MCU-32S

Es una placa de desarrollo parecida a la fabricada por Espressif, pero es de código abierto, es decir, toda la documentación acerca de diagramas, esquemático o componentes que tiene está disponible para todo público.[14]

Para la realización de estas prácticas se usó la placa de desarrollo denominada como ESP32 devkit v1, pero cualquiera de las versiones anteriores es compatible, solo hay que tener cuidado con las características especiales de cada placa para evitar incompatibilidades en puertos usados.

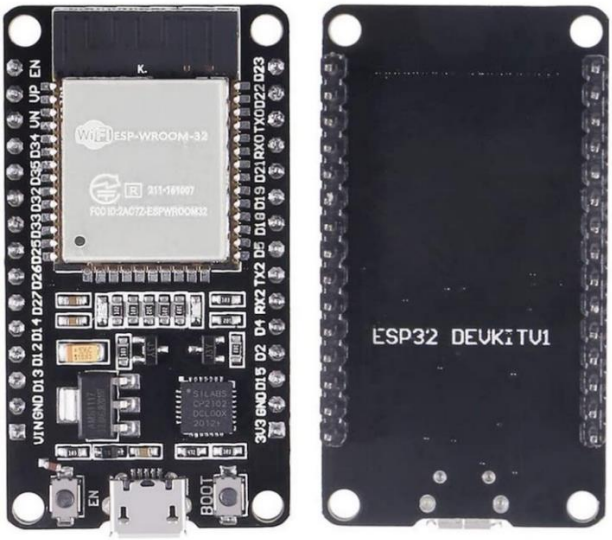


Ilustración 10: Placa de desarrollo ESP32 devkit v1

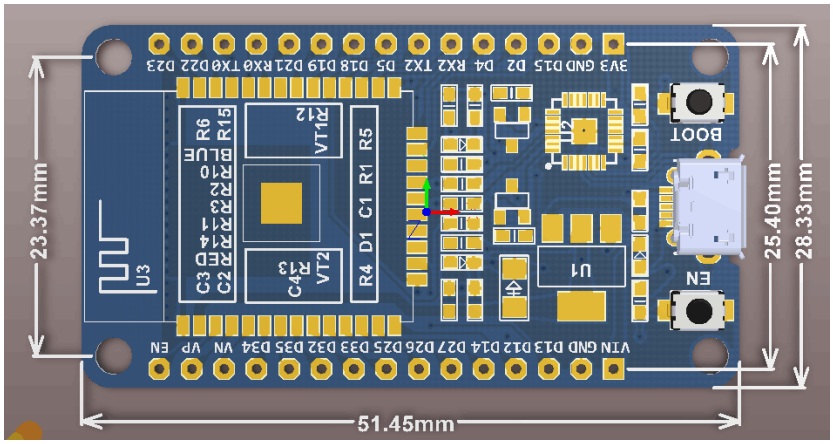


Ilustración 11: PCB de la placa de desarrollo ESP32 devkit v1

El esquemático de esta placa de desarrollo es el siguiente:

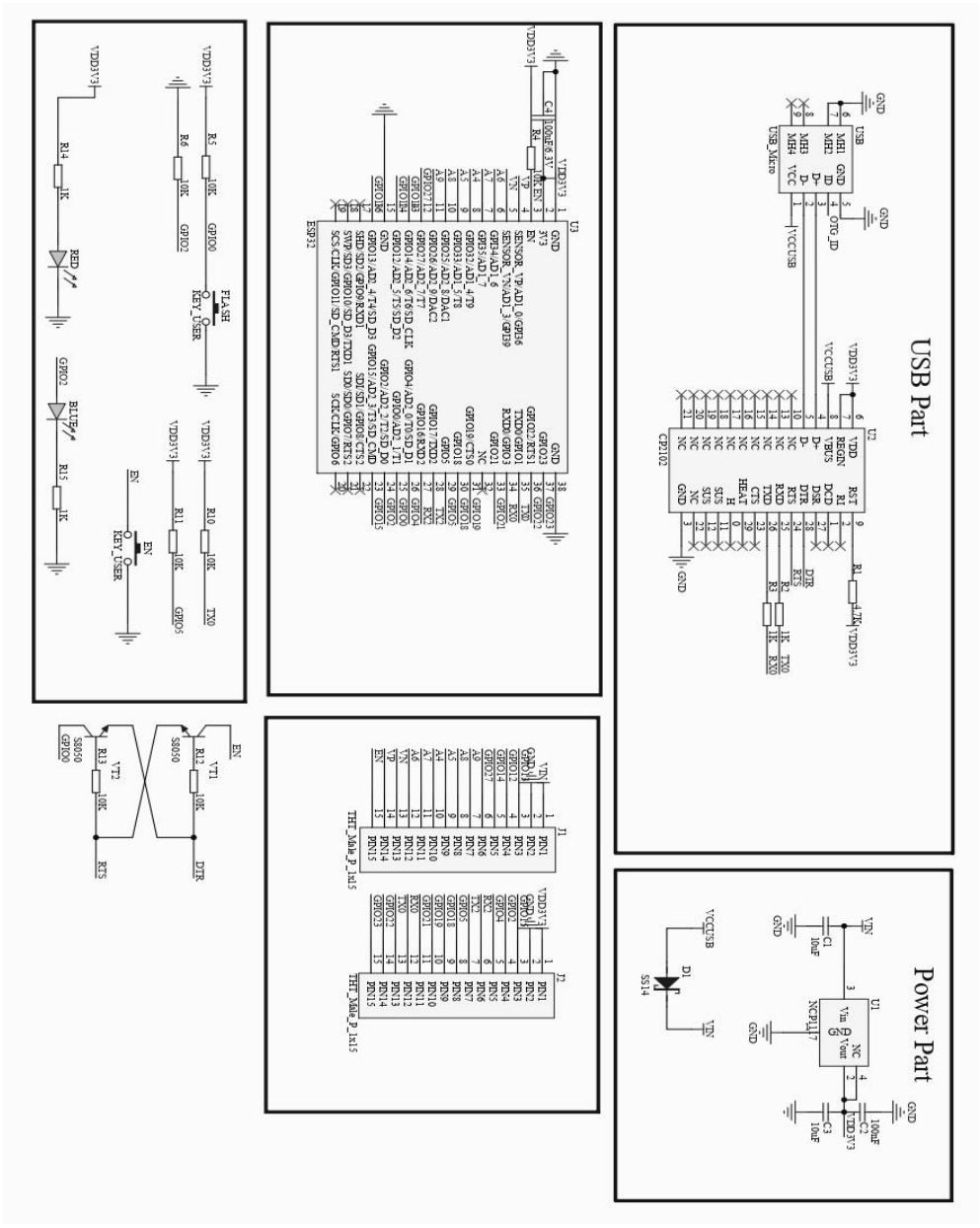


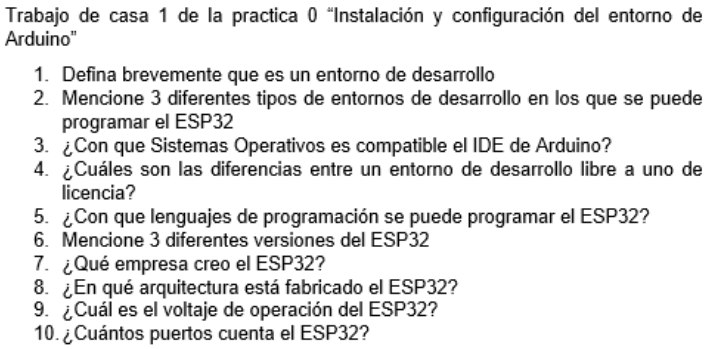
Ilustración 12: Esquemático de la placa de desarrollo ESP32 devkit v1

Capítulo 3

Estructura de las practicas:

El desarrollo de estas prácticas está conformado por lo siguiente:

Un cuestionario o trabajo de casa, esto con el fin de complementar y hacer que el alumno tenga las nociones básicas antes de la realización de cada práctica, cuentan con un mínimo de 7 preguntas

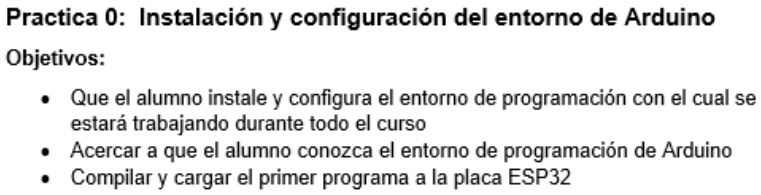


Trabajo de casa 1 de la practica 0 "Instalación y configuración del entorno de Arduino"

1. Defina brevemente que es un entorno de desarrollo
2. Mencione 3 diferentes tipos de entornos de desarrollo en los que se puede programar el ESP32
3. ¿Con que Sistemas Operativos es compatible el IDE de Arduino?
4. ¿Cuáles son las diferencias entre un entorno de desarrollo libre a uno de licencia?
5. ¿Con que lenguajes de programación se puede programar el ESP32?
6. Mencione 3 diferentes versiones del ESP32
7. ¿Qué empresa creo el ESP32?
8. ¿En qué arquitectura está fabricado el ESP32?
9. ¿Cuál es el voltaje de operación del ESP32?
10. ¿Cuántos puertos cuenta el ESP32?

Ilustración 13: Ejemplo de trabajo de casa de la practica 0

Una pequeña lista con cada objetivo que se pretende alcanzar al término de la practica



Practica 0: Instalación y configuración del entorno de Arduino

Objetivos:

- Que el alumno instale y configura el entorno de programación con el cual se estará trabajando durante todo el curso
- Acercar a que el alumno conozca el entorno de programación de Arduino
- Compilar y cargar el primer programa a la placa ESP32

Ilustración 14: Ejemplo del título y objetivos de la practica 0

Cuenta con una introducción breve de los temas más relevantes de lo que va a tratar la práctica y están explicados de una manera lo más simple posible para que el alumno pueda entenderlo fácilmente

Introducción:

Arduino IDE (Entorno de Desarrollo Integrado, por sus siglas en ingles)

Es un software oficial introducido por Arduino.cc gratis, libre y multiplataforma disponible en sistemas operativos como lo son macOS, Windows y Linux.

El entorno de desarrollo está inspirado en Processing (https://processing.org/) usado por profesores del Instituto de Diseño de Ivrea, como es el mismo lugar donde nació Arduino comparten mucha similitud, pero están escritos en diferentes lenguajes de programación, Processing está escrito principalmente en Java y Arduino IDE en C/C++.

Ilustración 15: Ejemplo de introducción de la practica 0

Una lista con el material que necesitara el alumno para la correcta realización de cada practica

Material para el desarrollo de la práctica:

- Jumpers
- Resistencia 220 ohm ½ Watt
- Leds de colores de 5mm
- Protoboard
- Placa de desarrollo ESP32
- Cable micro-USB

Ilustración 16: Ejemplo de la lista de materiales de la practica 0

Desarrollo que estarán los pasos detallados para la correcta realización de cada práctica, explicado con diagramas de conexión, códigos y explicaciones relevantes.

Desarrollo:

Primero procederemos a descargar e instalar el entorno de desarrollo de Arduino desde la página oficial, todo este procedimiento está basado para instalarse en Windows 10. *Nota: Para versiones anteriores de Windows este procedimiento puede variar.*

1. Primero descargaremos el software de Arduino del siguiente link https://www.arduino.cc/en/main/software

Ilustración 17: Ejemplo de desarrollo de la practica 0

Trabajo de casa 1 de la practica 0: “Instalación y configuración del IDE de Arduino”

1. Defina brevemente que es un entorno de desarrollo
2. Mencione 3 diferentes tipos de entornos de desarrollo en los que se puede programar el ESP32
3. ¿Con que Sistemas Operativos es compatible el IDE de Arduino?
4. ¿Cuáles son las diferencias entre un entorno de desarrollo libre a uno de licencia?
5. ¿Con que lenguajes de programación se puede programar el ESP32?
6. Mencione 3 diferentes versiones del ESP32
7. ¿Qué empresa creo el ESP32?
8. ¿En qué arquitectura está fabricado el ESP32?
9. ¿Cuál es el voltaje de operación del ESP32?
10. ¿Con cuántos puertos cuenta el ESP32?

Practica 0: Instalación y configuración del IDE de Arduino

Objetivos:

- Que el alumno instale y configura el entorno de programación con el cual se estará trabajando durante todo el curso
- Acercar a que el alumno conozca el entorno de programación de Arduino
- Compilar y cargar el primer programa a la placa ESP32

Introducción:

Arduino IDE (Entorno de Desarrollo Integrado, por sus siglas en inglés) es un software oficial introducido por Arduino.cc gratis, libre y multiplataforma disponible en sistemas operativos como lo son macOS, Windows y Linux.[15]

El entorno de desarrollo está inspirado en Processing (https://processing.org/) usado por profesores del Instituto de Diseño de Ivrea, como es el mismo lugar donde nació Arduino comparten mucha similitud, pero están escritos en diferentes lenguajes de programación, Processing está escrito principalmente en Java y Arduino IDE en C/C++.

Sus inicios se remontan al año 2005 en el instituto de Diseño Interactivo de Ivrea (Italia) surgiendo por la necesidad de crear un dispositivo que se pudiera utilizar en las aulas, de bajo coste, compatible en cualquier sistema operativo y lo principal, que tuviera una constante documentación adaptada a gente que sin ningún conocimiento pudiera aprenderlo de manera rápida y sencilla.

Desgraciadamente el Instituto cerró sus puertas el mismo año por lo que el temor de que fuera olvidado todo el esfuerzo de los profesores y alumnos, se decidió que fuera "liberado" a la comunidad para que así más personas lograran conocerlo y mantenerlo "vivo", proponiendo mejoras y sugerencias, lo cual funciono, actualmente se ha consolidado como un software y hardware libre de ámbito mundial.[16]

¿Qué se refiere cuando se habla de "Software Libre"?

Basado en la *Free Software Foundation* entidad sin ánimo de lucro encargada de promover el uso y desarrollo del software Libre a nivel mundial, indica que para que un

software sea considerado “software libre” debe de cumplir las siguientes cuatro “libertades”:[17]

- Libertad 0: Ejecutar el programa como se desee sin el importar el propósito
- Libertad 1: Estudio del funcionamiento del programa y que se pueda modificar por ende el acceso al código fuente es un requisito previo.
- Libertad 2: Libertad de redistribuir copias
- Libertad 3: Todas las versiones modificadas de terceros serán distribuidas sin restricción para que todos se beneficien de las mejoras del programa. El acceso al código fuente es un requisito previo para esto.

Para efectos legales el programa debe ser sometido a algún tipo de licencia de distribución como lo son GPL (*General Public License*), o la LGPL (*Lesser General Public License*).

Arduino es “Software Libre” porque es distribuido en una combinación de las licencias GPL y LGPL por esto, cualquier persona que quiera contribuir a su desarrollo lo puede hacer sin restricción, puede aportar nuevas características, nuevas funciones, resolver problemas o “bugs”, etc. Esto hace una comunidad de muchas personas que atraves de internet colaboren mutuamente haciéndolo evolucionar a un fin común.

¿Por qué elegir el entorno de desarrollo de Arduino?

Hay muchos entornos de desarrollo que se usan para programar el ESP32 como lo son MicroPyton o el marco de desarrollo de IoT (IDF) de Espressif, pero el software de Arduino nos ofrece ciertas ventajas:[18]

- Arduino IDE es libre por ende cualquier persona puede aportar a su desarrollo, esto hace que haya un ecosistema donde puedas encontrar mucha información al respecto.
- Arduino tiene una gran comunidad atraves de internet esto hace que exista mucha documentación acerca de proyectos, ideas, soluciones, practicas, etc. Encontrando muchos documentos bien detallados.

- El entorno de programación es multiplataforma, compatible con los principales sistemas operativos del mercado como lo son Windows, macOS X y Linux.

- El entorno y el lenguaje de programación son claros y simples, esto hace que sean fáciles de aprender y utilizar tanto para una persona inexperta en programación como para una persona con avanzados conocimientos en programación.

Material para el desarrollo de la práctica:

- Jumpers
- 1 Resistencia 220 ohm ½ Watt
- 5 leds de colores de 5mm
- Protoboard
- Placa de desarrollo ESP32
- Cable micro-USB

Desarrollo:

Primero procederemos a descargar e instalar el entorno de desarrollo de Arduino desde la página oficial, todo este procedimiento está basado para instalarse en Windows 10.

Nota 1: Para versiones anteriores de Windows este procedimiento puede variar Y asegúrese de tener conexión a internet para los siguientes pasos.

1. Primero descargaremos el software de Arduino del siguiente link https://www.arduino.cc/en/main/software

Ilustración 18: Página oficial de Arduino

2. Seleccionamos la opción “Win 7 and newer”

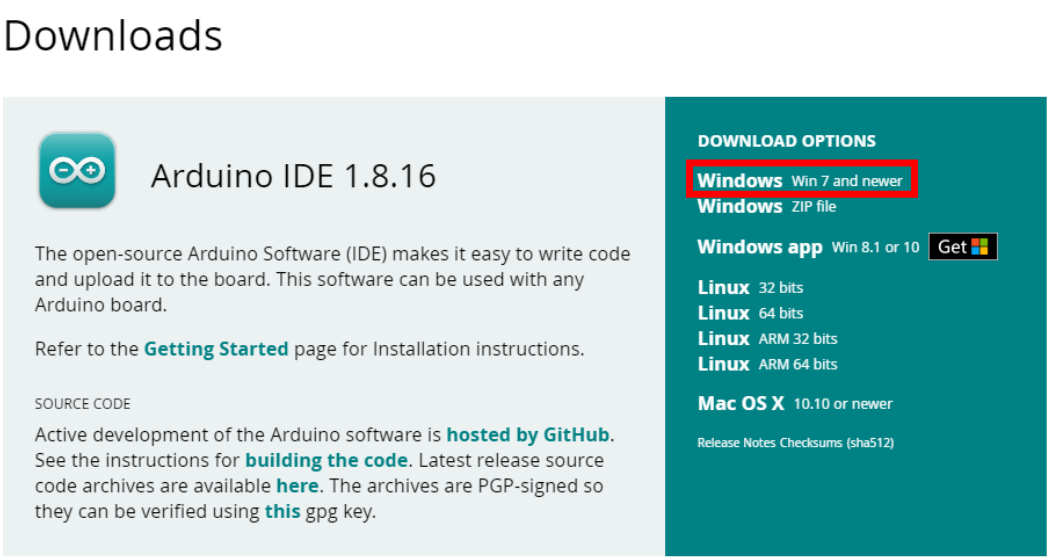


Ilustración 19: Opciones de descarga

3. Si aparece lo siguiente, solo dar clic en “JUST DOWNLOAD”

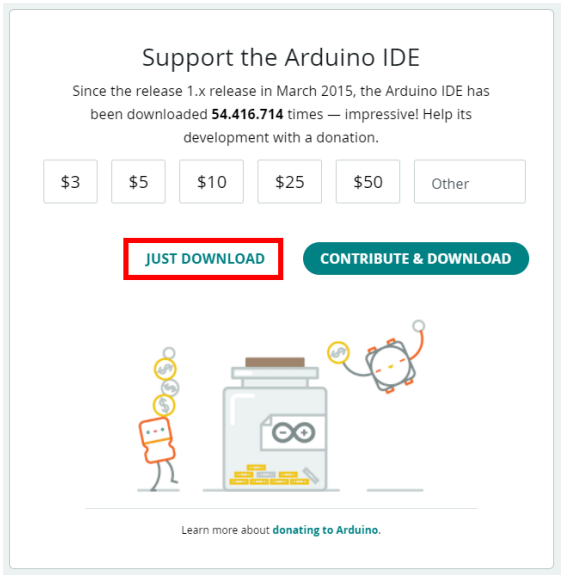


Ilustración 20: Aviso de donación

4. Ejecutar el archivo descargado haciendo doble clic sobre él.

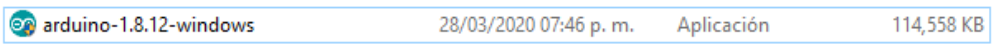


Ilustración 21: Archivo descargado

5. Saldrá una ventana de permisos, damos clic en “si”

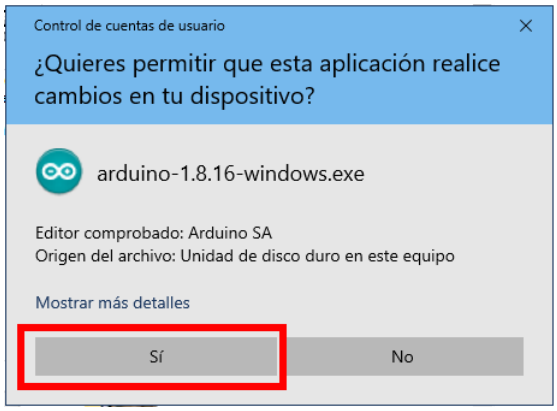


Ilustración 22: Ventana de solicitud de permisos

6. Comenzará el asistente de instalación, para continuar, damos clic en “I Agree”

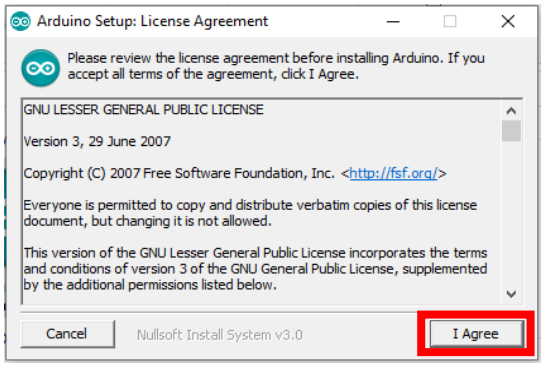


Ilustración 23: Asistente de instalación

7. La siguiente ventana mostrara que componentes queremos instalar, NO desmarcar nada y dar clic en “Next”

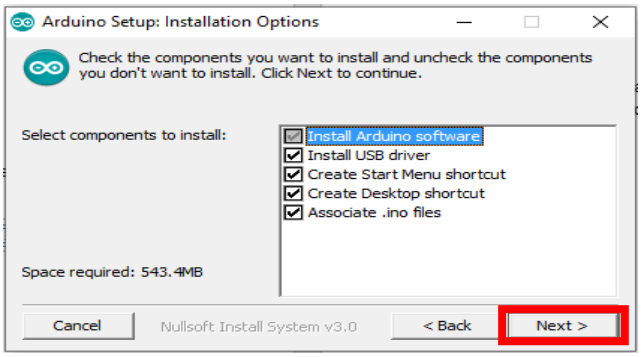


Ilustración 24: Opciones de instalación de componentes

8. Nos pedirá una ruta de instalación del programa, no modificarla y dar clic en “Install”

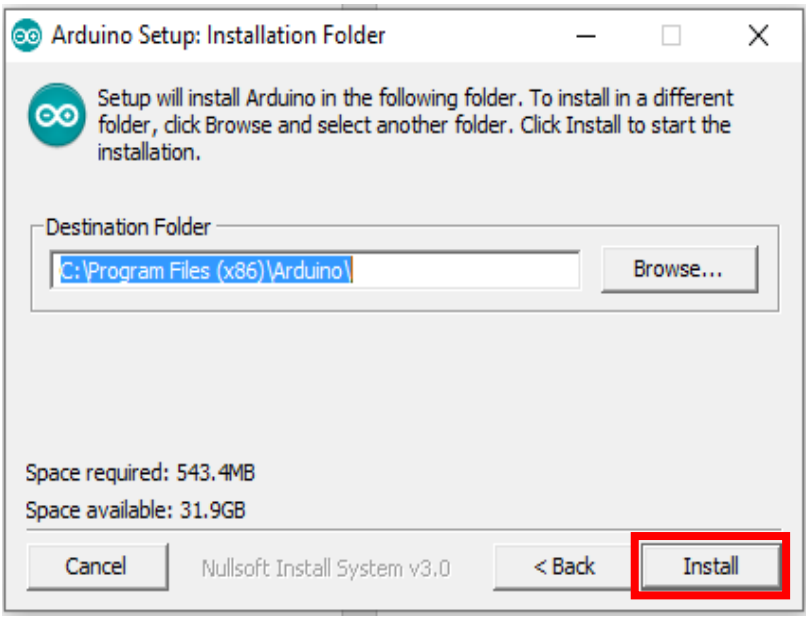


Ilustración 25: Ruta de instalación

9. Comenzará la instalación

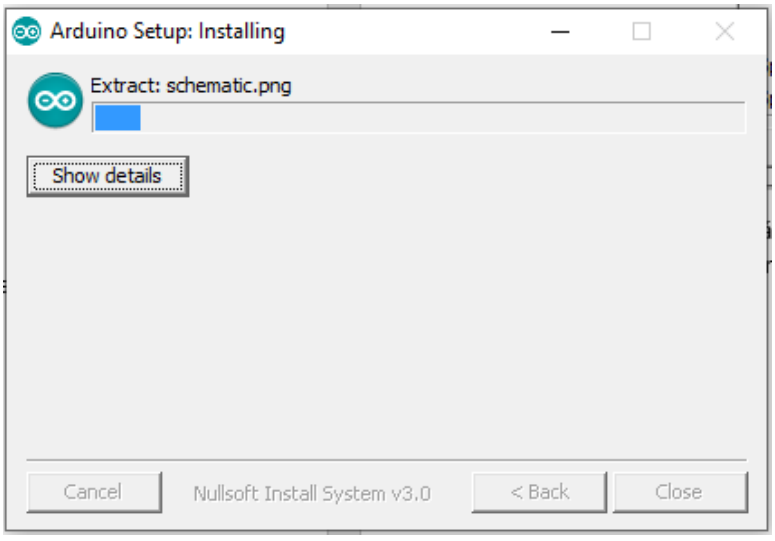


Ilustración 26: Progreso de la instalación

10. Si es una instalación desde cero, nos pedirá permiso para instalar unos drivers, solo dar clic en “instalar” a cada uno.

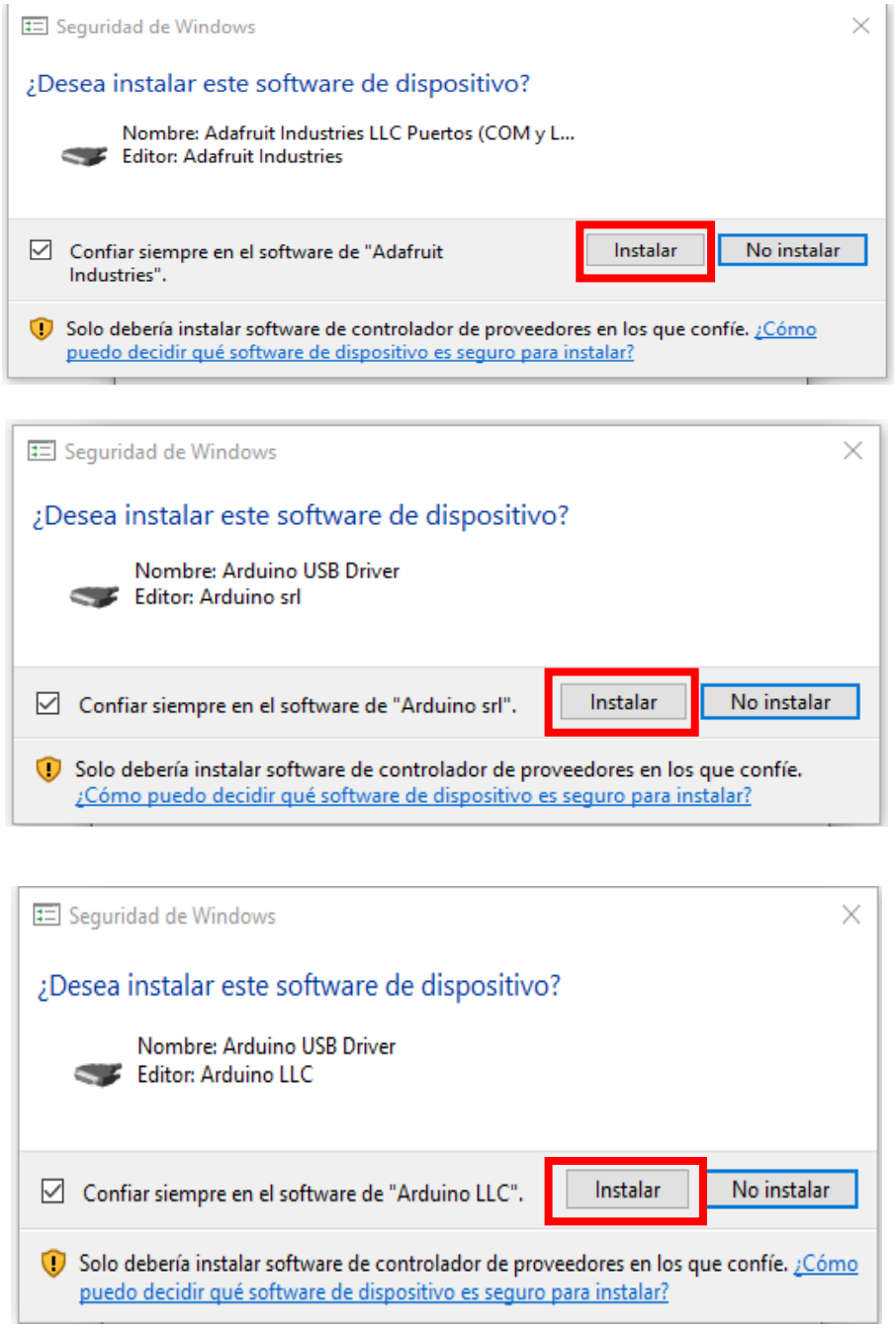


Ilustración 27: Solicitud de instalación de drivers

11. Por último, aparecerá que el programa termino de instalarse, dar clic en “Close” para terminar.

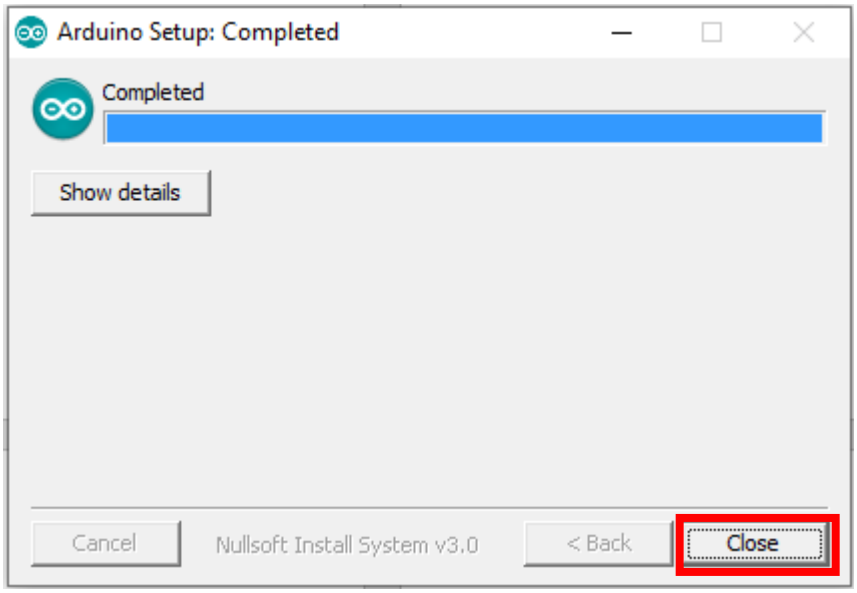


Ilustración 28: Finalización de la instalación

Instalaremos las Librerías y configuramos el IDE de Arduino para programar el ESP32.

1. Abrimos el Arduino IDE dando doble clic en el icono que se generó en el escritorio.

Ilustración 29: Icono de Arduino IDE

2. Vamos a “Archivo” y dar clic en “Preferencias”

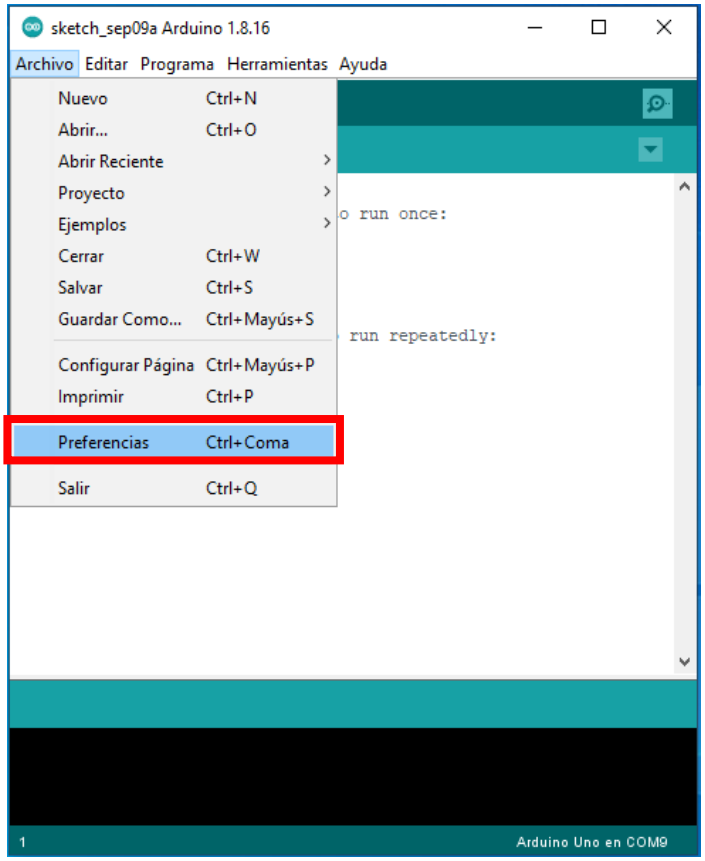


Ilustración 30: Selección de "Preferencias"

3. Se abrirá una nueva ventana

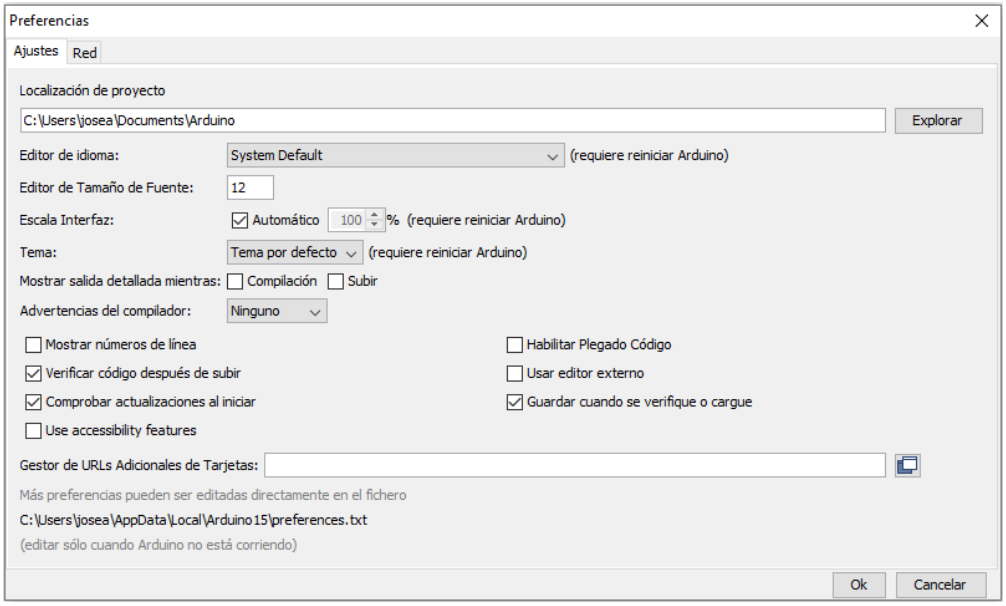


Ilustración 31: Ventana de Preferencias de Arduino IDE

4. Transcribimos el siguiente link en la parte de “Gestor de URLs Adicionales de Tarjetas” y dar clic en OK

Link: https://dl.espressif.com/dl/package_esp32_index.json

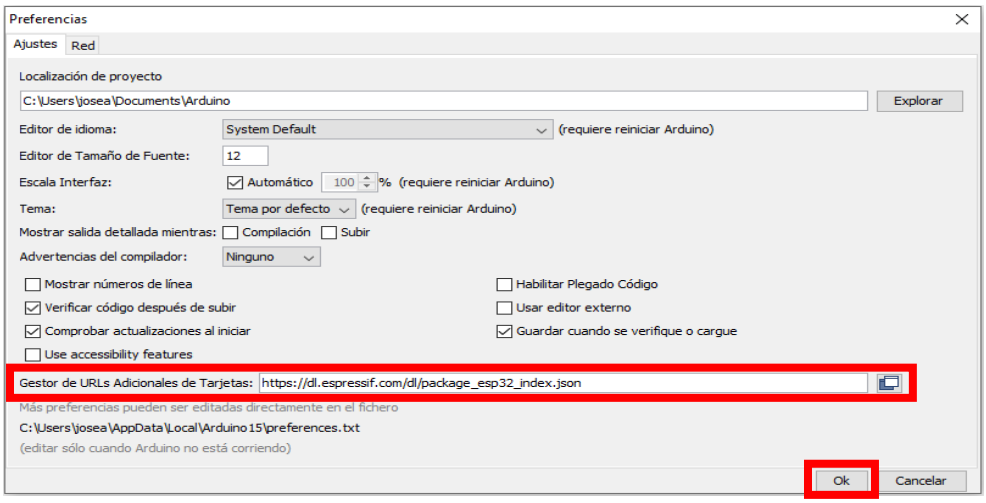


Ilustración 32: Configuración del IDE de Arduino

5. Seguiremos la siguiente ruta: “Herramientas”, “Placas”, seleccionar “Gestor de tarjetas”

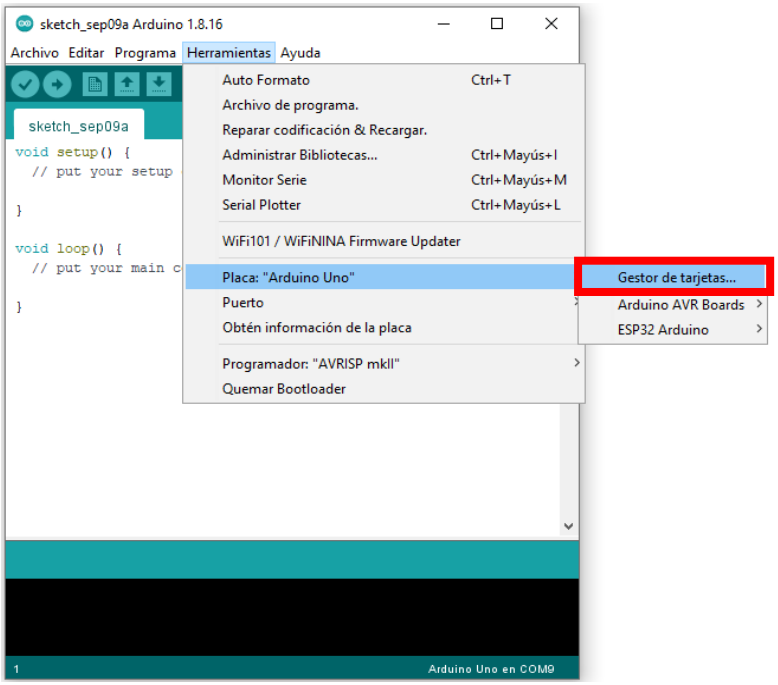


Ilustración 33: Selección de "Gestor de tarjetas"

6. Se abrirá una ventana, en el cuadro de búsqueda escribimos “ESP32” presionamos la tecla “Enter” del teclado y seleccionamos la primera opción dando clic en “Instalar”

Ilustración 34: Ventana del Gestor de tarjetas de Arduino IDE

7. Esperamos hasta que termine la instalación.

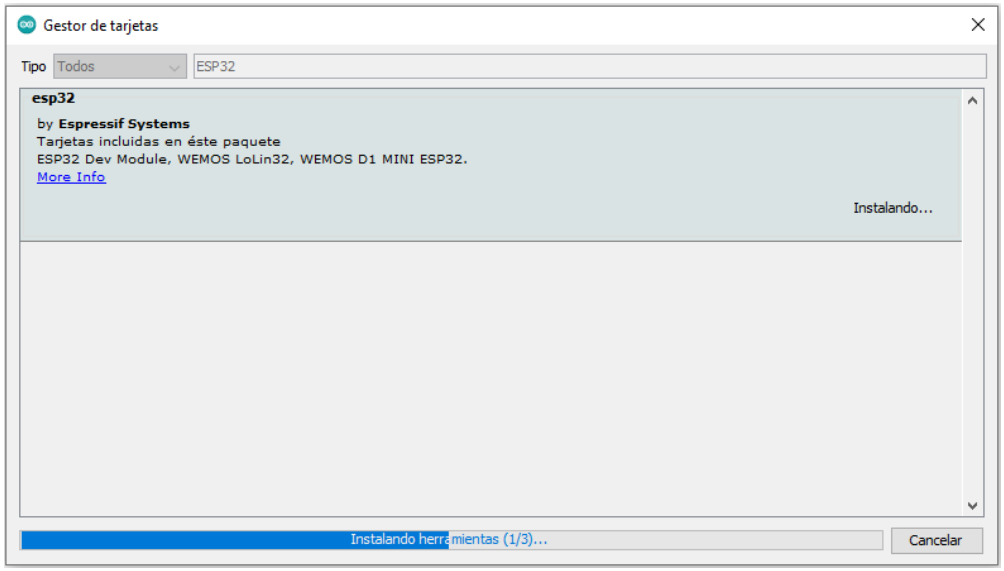


Ilustración 35: Instalación de la librería del ESP32

8. Una vez terminado dar clic en “Cerrar”

Ilustración 36: Instalación finalizada de la librería del ESP32

9. Seguiremos la siguiente ruta “Herramientas”, “placa”, “ESP32 Arduino” seleccionar “ESP32 Dev Module”.

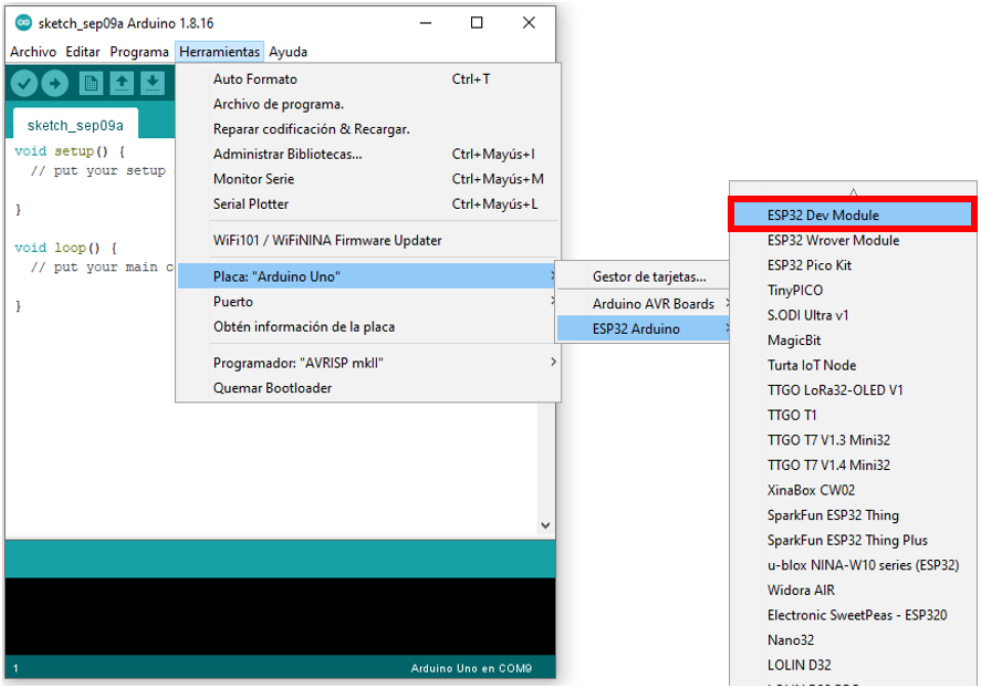


Ilustración 37: Selección de la placa "ESP32 Dev Module"

10. En la pestaña de “Herramientas” seleccionar en el apartado de “Aproad Speedy:” seleccionar el número “115200”

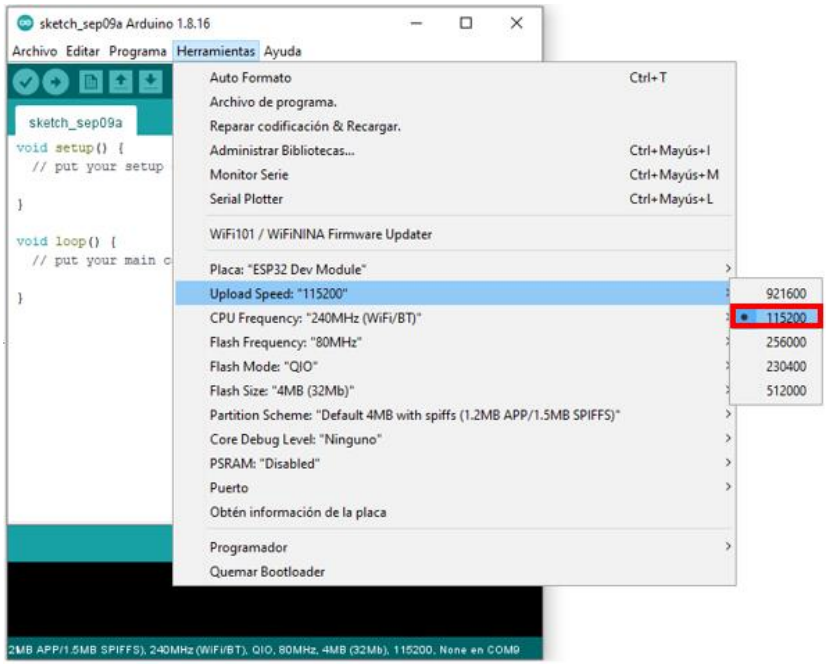


Ilustración 38: Configuración del "Aproad Speedy"

11. Asegurarse que en la pestaña “Herramientas” en el apartado de “CPU Frequency” Este seleccionada la opción “240 MHZ (Wi-Fi/BT)”

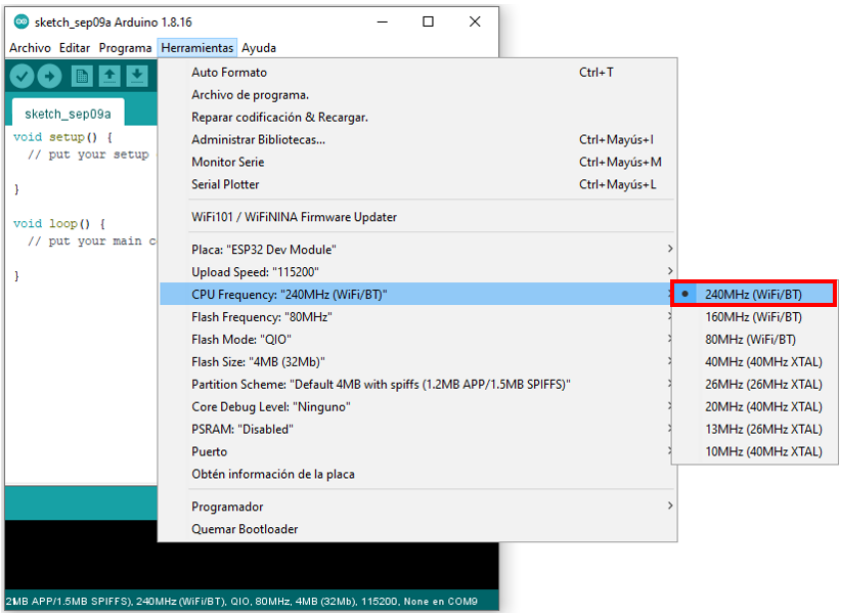


Ilustración 39: Configuración del "CPU Frequency"

12. En la pestaña “Herramientas” seleccionar en el apartado de “Flash Frequency:” seleccionar la segunda opción “40 MHZ”

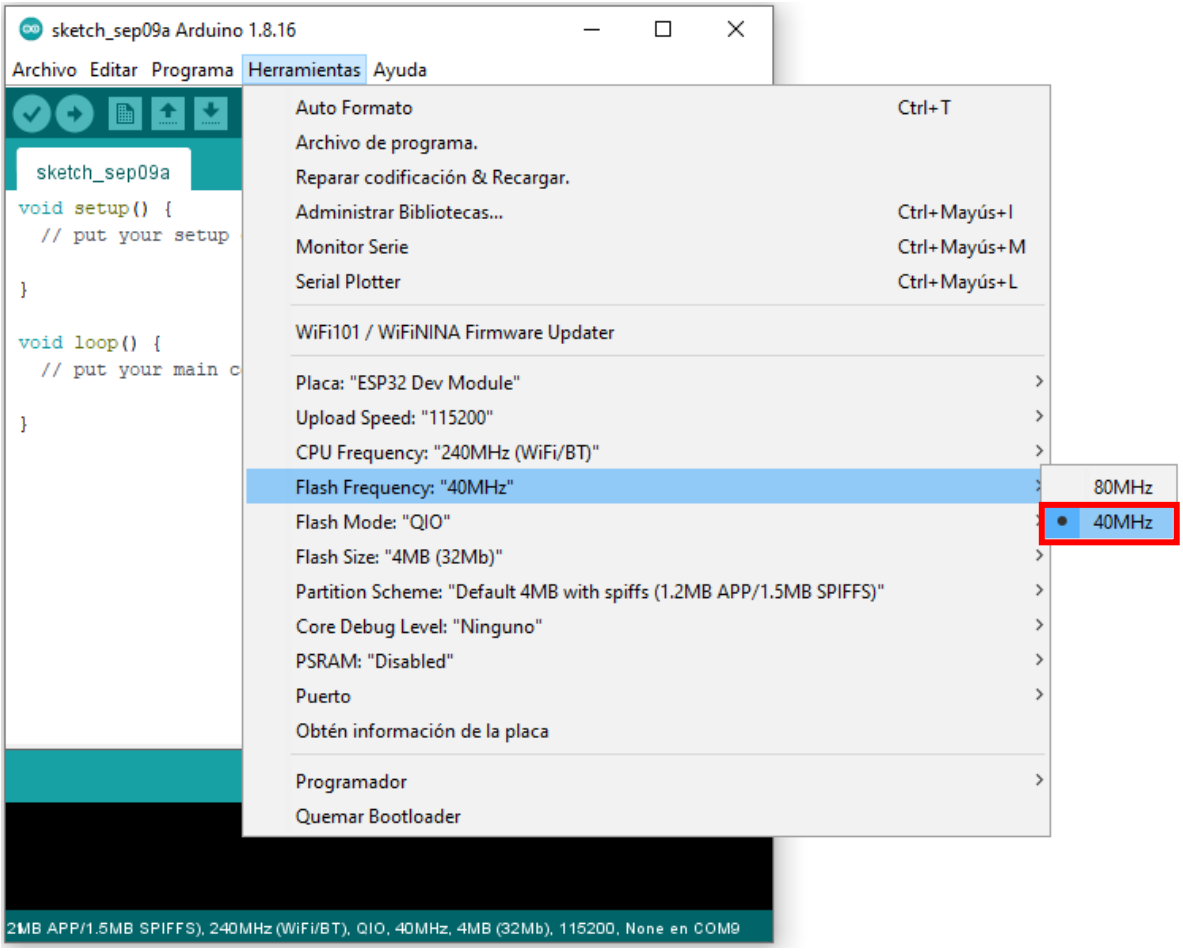


Ilustración 40: Configuración de "Flash Frequency"

Cargaremos el primer programa para comprobar el funcionamiento del ESP32 el cual será un led conectado a una de las terminales del ESP32 parpadee cada cierto tiempo, tendremos que seguir los siguientes pasos, todo bajo Windows 10.

1. Conectar el ESP32 con un cable USB a un puerto libre de la computadora luego ir al Administrador de dispositivos de Windows haciendo clic derecho en el logo de Windows.

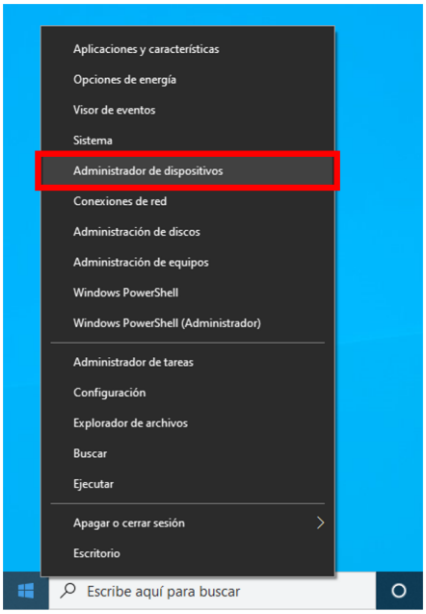


Ilustración 41: Administrador de dispositivos

2. Se abrirá una ventana, dar clic en “Puertos (COM y LPT)”, Si se instaló correctamente aparecerá algo parecido a “Silicon Las CP21x USB ti UART Bridge (COM9)”, el número que esta después de COM es el puerto al cual está conectado.

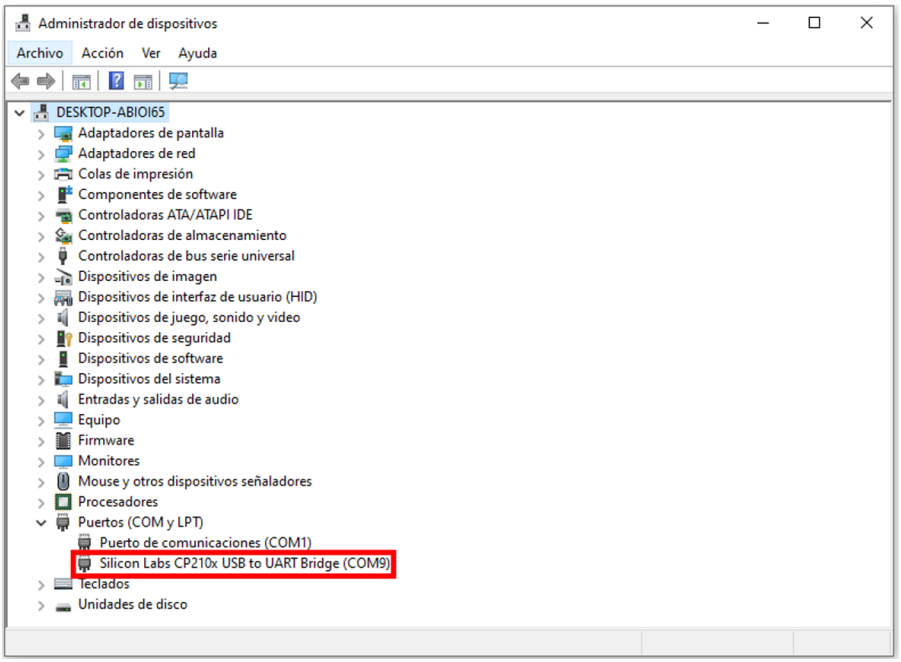


Ilustración 42: Ventana del Administrador de dispositivos

3. Regresamos al IDE de Arduino e iremos a la pestaña “Herramientas” y en el apartado de “Puerto” seleccionar el puerto que apareció en la ventana de “Administración de dispositivos”

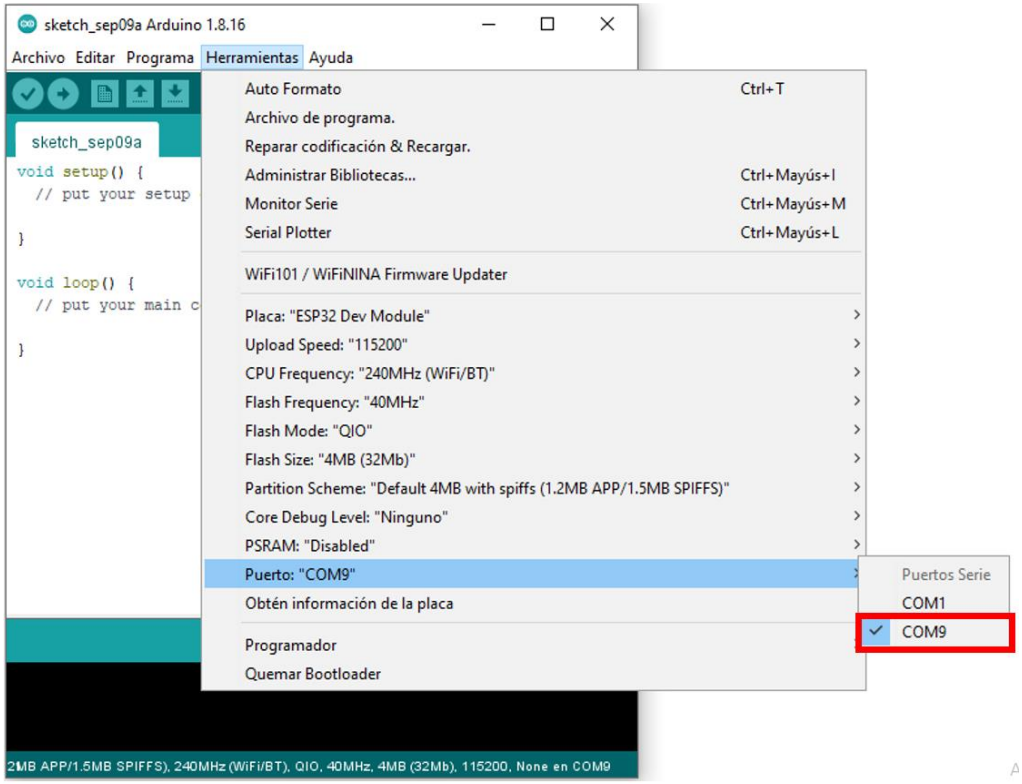


Ilustración 43: Selección del puerto COM9

4. Transcribir el código siguiente y pegarlo en el IDE de Arduino.

Código:

```
void setup() {
pinMode(23, OUTPUT); // Configuramos el pin 23 como salida
}
void loop() {
digitalWrite(23, HIGH); //Mandamos un pulso alto por el pin
configurado 23
delay(1000); //Espera de 100 ms
digitalWrite(23, LOW); //Mandamos un pulso bajo por el pin
configurado 23
delay(1000); //Espera de 100 ms
}
```

5. Después hacemos clic en el botón subir (Marcado con un cuadro rojo)

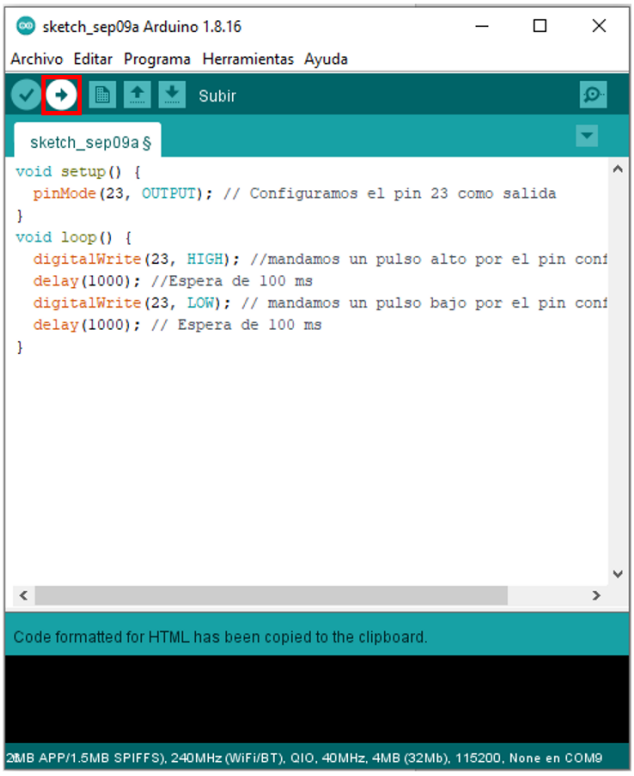


Ilustración 44: Cargar el primer programa

6. Si no se ha guardado el archivo, el entorno dirá que debe de guardar el archivo primero antes de cargarlo al ESP32, guardarlo en un lugar seguro.

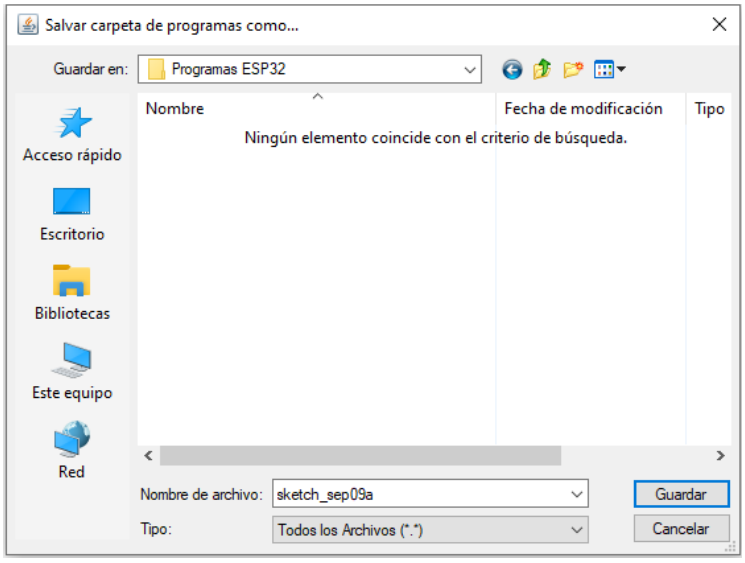


Ilustración 45: Ventana de guardado

7. Después se procederá a compilar el programa si todo sale bien se procederá a subirlo, y en la parte inferior aparecerá la leyenda “Subiendo…”, en el recuadro negro empezaran a parecer varios textos. Cuando aparezca el texto “Connecting……” Presionar el botón “BOOT” de la ESP32.

Nota: En algunos ESP32 más actuales no es necesario presionar el botón “BOOT”, revisar la documentación del modelo de ESP32 que está usando para más información

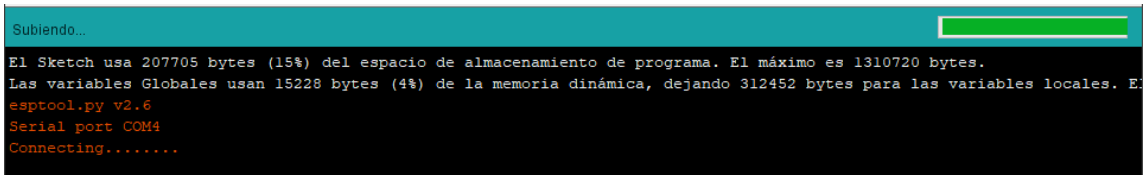


Ilustración 46: Cuadro de progreso

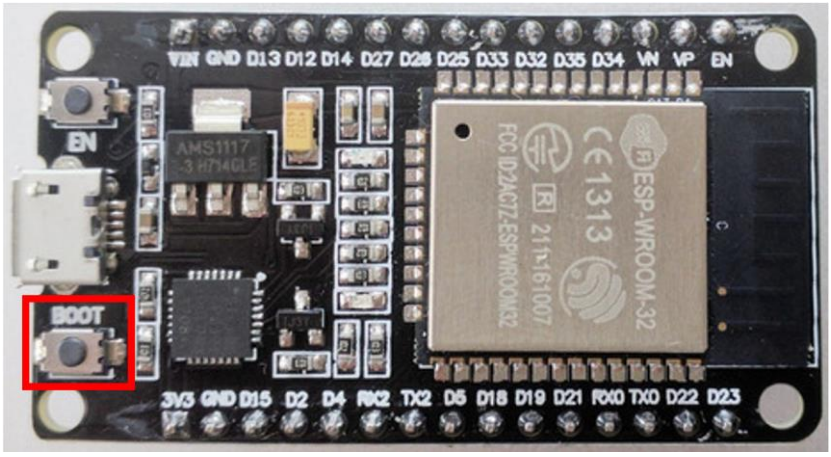


Ilustración 47: Ubicación del botón "BOOT" del ESP32

8. Si todo sale bien aparecerá el siguiente texto “Leaving…. Hard resetting via RTS pin….” indicando que el programa se compilo y se ha cargado al ESP32.

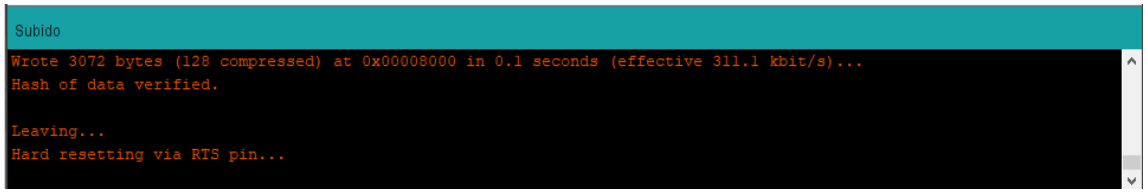


Ilustración 48: Finalización del compilado y subida del programa

9. Para comprobar el funcionamiento se armará el siguiente circuito, debe de parpadear el led cada segundo.

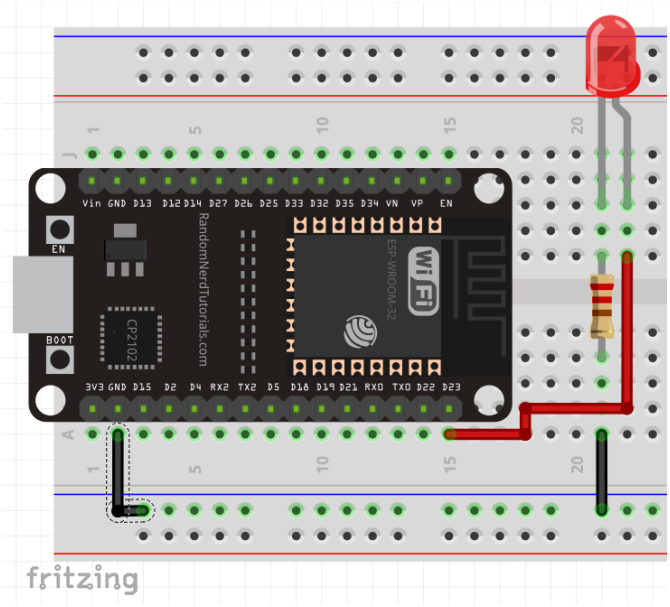


Ilustración 49: Diagrama de comprobación de funcionamiento

Trabajo de casa 2 de la practica 1: Puertos GPIO del ESP32 y UART

1. ¿En qué otros dispositivos encontramos los puertos GPIO?
2. ¿Cuáles con las características eléctricas de los puertos GPIO del ESP32?
3. ¿Describa que es una entrada de *Pull-Up* y de *Pull-Down*?
4. ¿Qué es una interrupción por hardware y de software?
5. ¿Qué es una comunicación *Half Duplex* y *Full Duplex*?
6. Con respecto a la pregunta anterior, ¿Cuáles son sus diferencias?
7. Describa el funcionamiento de una comunicación de una UART y USART
8. ¿Aparte de la comunicación UART que otros protocolos de comunicación hay?

Practica 1. Puertos GPIO del ESP32 y UART

Objetivos:

- Que el alumno comprenda la definición de puertos GPIO
- Hacer que el alumno conozca la distribución de pines y conocer su funcionamiento en el ESP32 usando ejemplos fáciles de entender
- Conocer el estándar de comunicación UART y cómo usarlo en el ESP32

Introducción:

Puertos GPIOs del ESP32

GPIO (*General Purpose Input/Output*, Entrada/Salida de Propósito General) es una interfaz que permite a un chip usar un pin como entrada o salida que se puede programar como se desee en un tiempo de ejecución, es decir, el usuario puede programar este pin en la rutina de ejecución de un programa como él quiera.[19], [20]

Cuando el pin es configurado como una salida puede transferir datos o información a dispositivos externos de manera eficiente y cuando el pin es configurado como entrada puede "leer" datos de periféricos externos y manipularlos internamente en tiempo real, esto proporciona un mecanismo eficiente para enviar o recibir datos a dispositivos externos según lo requiera el usuario.[21]

Aunque en general solo se pueden configurar como entrada o salida de datos en algunos chips tienen la capacidad de configurar estos pines como interrupciones de CPU o en los más modernos controlar y usar el acceso directo a memoria (DMA) esto para transferir grandes paquetes de datos de una manera más rápida y eficiente.

En el caso del ESP32 tiene 34 pines GPIO que pueden ser programados para realizar diferentes funciones, pero algunos de ellos tienen restricciones o capacidades diferentes, por ejemplo: solo pueden ser entradas o salidas digitales, pueden leer señales analógicas o con capacidad táctil capacitiva.

La mayoría de estos pines pueden ser configurados internamente como entradas *pull-up*, *pull-down* o como alta impedancia, cuando son configurados como entradas los datos se

pueden leer directamente desde el registro del chip, pueden ser configurados como interrupciones de CPU o se pueden usar para leer datos bidireccionalmente.

El ESP32 posee muchos pines, pero no todos pueden usarse por el usuario y algunos solo pueden usarse de una forma especial.

Nota: todos los puertos del ESP32 tienen un voltaje máximo de 3.3v, no conectar un voltaje superior a este, en caso de hacerlo dañaremos internamente el ESP32 volviéndolo inservible.

La siguiente tabla muestra los pines que pueden usarse como entradas o salidas, así como pequeñas notas para tener en cuenta si tienen consideraciones especiales.

GPIO	Input	Output	Notas
0	pulled up	OK	Salida PWM al inicio del programa
1	TX pin	OK	Salida de depuración
2	OK	OK	Conectado a un LED en la PCB
3	OK	RX pin	Estado alto en el inicio del programa
4	OK	OK	
5	OK	OK	Salida PWM al inicio del programa
6	x	x	Conectado a la ISP integrada
7	x	x	Conectado a la ISP integrada
8	x	x	Conectado a la ISP integrada
9	x	x	Conectado a la ISP integrada
10	x	x	Conectado a la ISP integrada
11	x	x	Conectado a la ISP integrada
12	OK	OK	Al inicio del programa se pondrá en alto
13	OK	OK	
14	OK	OK	Salida PWM al inicio del programa
15	OK	OK	Salida PWM al inicio del programa
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	

32	OK	OK	
33	OK	OK	
34	OK		Solamente entrada
35	OK		Solamente entrada
36	OK		Solamente entrada
39	OK		Solamente entrada

Tabla 2: Características de los pines del ESP32

Basado en la placa ESP32 DEV KIT V1 los pines están acomodados de la siguiente manera.

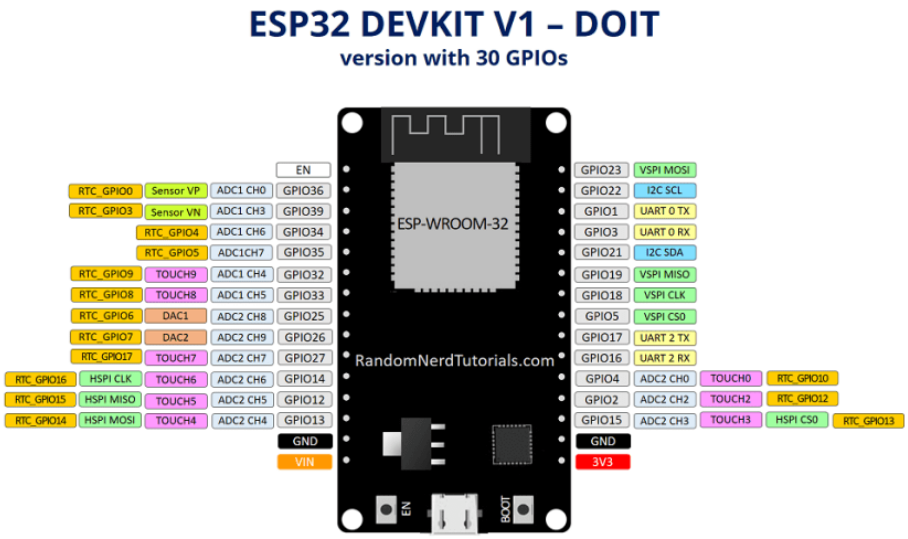


Ilustración 50: Distribución de pines del ESP32

La UART (*Universal Asynchronous receiver-transmitter*, en español: Transmisor-Receptor Asíncrono Universal) es un dispositivo de hardware que se utiliza para hacer una comunicación serie asíncrona donde las características de los datos y su velocidad de transmisión pueden ser configurables como requiera el usuario, generalmente se encuentra como un circuito integrado individual o forma parte de un conjunto de más circuitos conectados entre sí, utilizado para la comunicación en serie entre una computadora o puerto a un dispositivo externo.[22]

Toma la información en bytes de datos y los transmite uno por uno de forma secuencial iniciando desde el menos significativo al más significativo utilizando “marcadores” para saber cuál es el bit de inicio y el ultimo, de esta manera el dispositivo que reciba la señal de datos se encargue de la sincronización.

Generalmente la UART no genera directamente los voltajes para la transmisión y recepción de datos, tienen que ayudarse de otros módulos externos que ya tienen estándares para la señalización de voltaje como los son: El estándar RS-232, RS-422 y el RS-485 de la EIA (*Electronic Industries Alliance*, en español: Alianza de Industrias Electrónicas).[22]

Su utilización se remonta desde los inicios de la transmisión de datos entre computadoras dando así también un precedente para el hardware utilizado en Internet.

El diseño del primer UART se lo debemos a Gordon Bell reconocido ingeniero informático, gerente y antiguo empleado de *Digital Equipment Corporation* (DEC), creado principalmente para equiparlos a las computadoras PDP-Series siendo el primer computador de *Digital Equipment* denominado como PDP-1, la UART se le llamo: unidad de línea, pero era de gran tamaño por lo que rápidamente lo convirtieron en un single-chip, posteriormente diseñadores de Western Digital crearon el primer chip UART llamado el WD1402A en 1971.[23]–[25]

Desde su creación fue muy implementado para la comulación entre dispositivos, por ejemplo, el *National Semiconductor* 8250 fue implementado en la original IBM PC´s como una tarjeta adaptadora en la década de los 80´s, después en la década de los 90´s nuevos UART fueron mejorados con búferes en un solo chip, permitiendo mayor velocidad de transmisión y sin pérdida de datos.

A partir de la década de los 2000 gran parte de las computadoras compatibles con IBM PC eliminaron los puertos COM RS-232 que se usaban para usar la UART, cambiándolos por los famosos USB.

Actualmente la UART ya no se ocupa para transmitir datos entre computadoras, pero aún sigue siendo muy usado internamente por los procesadores o microprocesadores del mercado para brindar a los diseñadores de hardware la posibilidad de interactuar ellos.

El ESP32 basado en el datasheet del fabricante posee tres interfaces UART denominadas como UART0, UART1 y UART2 las cuales son compatibles con el estándar RS-232 y RS-485, con una comunicación de hasta 5Mbps de velocidad y estas interfaces pueden usarse directamente por el controlador DMA (*Direct Memory Access*) o por la CPU.

Nota: Aunque el ESP32 posee tres interfaces UART solo se pueden usar dos, el UART0 y la UART2, la UART1 está conectada internamente al controlador de interfaz USB para la programación del ESP32.

Material para el desarrollo de la práctica:

- Protoboard
- 6 leds de colores de 5mm
- 6 resistencias de 220 ohm ½ watt
- Placa ESP32
- Jumpers
- 6 Push Button de dos o cuatro terminales
- Cable micro USB

Desarrollo:

Ejercicio 1 Entradas y salidas:

Para familiarizarnos con el entorno y los puertos del ESP32 primero empezaremos a configurar los puertos del ESP32 para utilizarlos como entras y salidas.

Para eso se deberá transcribir el siguiente código al IDE de Arduino.

```
void setup() {
pinMode(16,INPUT); //Se configura el pin 16 como entrada
pinMode(21,OUTPUT); //Se configura el pin 21 como salida
}
void loop() {
if (digitalRead (16) == HIGH)  //Se hace la condición, si la
lectura digital del pin 16 es alto
{
  digitalWrite(21,HIGH);  //Se mandara un pulso alto al pin
21
}
else
{
  digitalWrite(21,LOW); //Si no se cumple la condición,
mandara un pulso bajo
}
}
```

Una vez transcrito el código se procederá a compilarlo y cargarlo al ESP32 usando los pasos de la práctica anterior

Se armará el siguiente circuito para probar su funcionamiento.

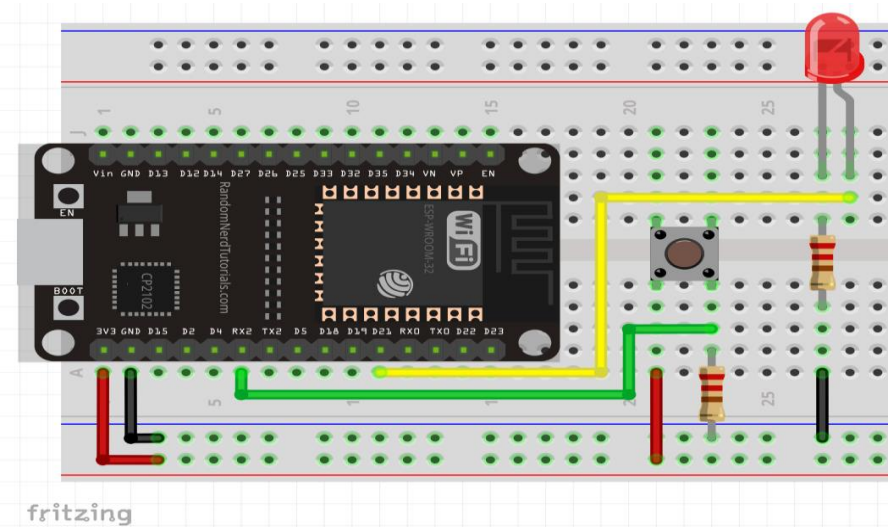


Ilustración 51: Circuito del Ejercicio 1

Lo que se quiere observar es que cada vez que el *push button* es presionado, el LED encenderá y cuando no esté siendo presionado el led se quedará apagado.

Ejercicio 2 Contador binario:

Ahora implementaremos un pequeño contador binario para usar más puertos del ESP32

Para eso usaremos el siguiente código:

```
int outPin[] = {19, 21, 22, 23}; //Configuramos un ARRAY de
los pines a usar

void setup() {
  //Utilizamos un arreglo con el ciclo FOR para definir todos
los pines de ARRAY como salidas
  int i = 0;
  for ( i = 0; i < 4; i++)
    pinMode(outPin[i], OUTPUT);
}
void loop() {
  //Para hacer la cuenta ascendente utilizaremos la ayuda de
dos ciclos FOR
```

```
  int i = 0, j = 0; //Primero definimos las variables para
usar los contadores

  for ( i = 0; i < 16; i++) //Este ciclo FOR nos servirá para
poner un límite al contador, al ser un contador de 4 bits el
límite seria 16
  {
    for ( j = 0; j < 4; j++)//Este ciclo FOR nos servirá para
indicar que pin del ARRAY estara en estado ALTO o BAJO
    {
      if ( ( (i >> j) & 1 )  == 1 )//Esta condición es la que
se encargara de saber cual led cambiara o no su estado
        digitalWrite(outPin[j], HIGH);
      else
        digitalWrite(outPin[j], LOW);
    }
    delay(1000); //Un pequeño retraso para que la cuenta
binaria cambie cada 1000ms
  }
}
```

Implementaremos el siguiente diagrama para comprobar su funcionamiento

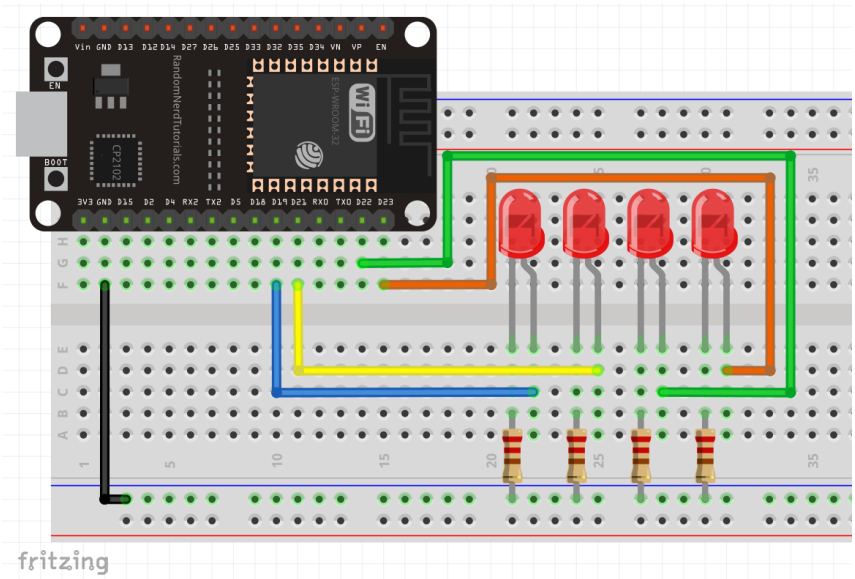


Ilustración 52: Circuito del Ejercicio 2

Se debe observar una cuenta binaria de 0 a 15 con los leds conectados con un retraso de 1000ms o 1 segundo.

Ejercicio 3 Interrupciones:

Ahora implementaremos algo llamado “Interrupciones”, como tal, cualquier pin del ESP32 lo podemos usar como interrupciones excepto a los pines que solo los podemos usar como entradas (Véase Tabla 3).

Para generar las interrupciones haremos una mezcla de los dos ejemplos anteriores y utilizaremos la comunicación serial UART.

Copiaremos en siguiente código al IDE de Arduino:

```
int outPin[] = {19, 21, 22, 23}; //Configuramos un ARRAY de
los pines a usar

void setup() {
  //Utilizamos un arreglo con el ciclo FOR para definir todos
los pines de ARRAY como salidas
  int i = 0;
  for ( i = 0; i < 4; i++)
  pinMode(outPin[i], OUTPUT);
  Serial.begin(115200); //Inicializamos el monitor Serial con
una velocidad de 115200 baudios
  pinMode(18, INPUT_PULLDOWN); //Configuramos el pin 18 como
entrada con resistencia de PULLDOWN
  attachInterrupt(18, isr, HIGH); //Esta instrucción es la
que hace que el pin 18 se comporte como una interrupción.
}
//Este pequeño código es lo que hará si el programa recibe
una interrupción
void isr() {
  Serial.println("¡Interrupción!");
}
void loop() {
  //Para hacer la cuenta ascendente utilizaremos la ayuda de
dos ciclos FOR
  int i = 0, j = 0; //Primero definimos las variables para
usar los contadores

  for ( i = 0; i < 16; i++) //este ciclo FOR nos servirá para
poner un límite al contador, al ser un contador de 4 bits el
límite seria 16
  {
    for ( j = 0; j < 4; j++)//Este ciclo FOR nos servirá para
indicar que pin del ARRAY estará en estado ALTO o BAJO
   {
```

```
if ( ( (i >> j) & 1 )  == 1 )//Esta condición es la que se
encargara de saber cuál led cambiara o no su estado
        digitalWrite(outPin[j], HIGH);
      else
        digitalWrite(outPin[j], LOW);
    }
    delay(1000); //Un pequeño retraso para que la cuenta
binaria cambie cada 1000ms
  }
```

Antes de Armar el circuito haremos una breve explicación de cómo se generan las interrupciones

Para hacer que un pin del ESP32 se comporte como un pin de interrupción primero tenemos que definir que dicho pin lo ocuparemos como entrada, por eso no ayudamos de la siguiente instrucción: "pinMode(18, INPUT_PULLDOWN);", en vez de poner INPUT como en el primer ejercicio ponemos INPUT_PULLDOWN para usar la resistencia de *pull-down* que tiene el ESP32 internamente, también se puede poner cono resistencia de *pull-up*, para eso pondríamos INPUT_PULLUP.

Después usamos la instrucción "attachInterrupt(18, isr, HIGH);" para hacerle saber al ESP32 que el pin 18 lo usaremos como interrupción, los tres parámetros que están entre paréntesis son:

El primero es el pin que anteriormente definimos como entrada, el segundo es otro "programa o código" que se ejecutara cuando la interrupción se genere y el tercero es el modo con en el que se generara la interrupción, es decir, si queremos que se genere con pulsos altos o bajos, tenemos varias opciones para eso, las cuales son:

- *LOW*: Para activar la interrupción cuando el pin este en estado bajo
- *HIGH*: Para activar la interrupción cuando el pin este en estado alto
- *FALLING*: Para activar la interrupción cuando el pin pasa de un estado alto a uno bajo
- *RISING*: Para activar la interrupción cuando el pin pase de un estado bajo a uno alto
- *CHANGE*: Para activar la interrupción cada vez que el pin cambie de estado

Una vez entendido lo anterior armaremos el siguiente circuito para comprobar su funcionamiento

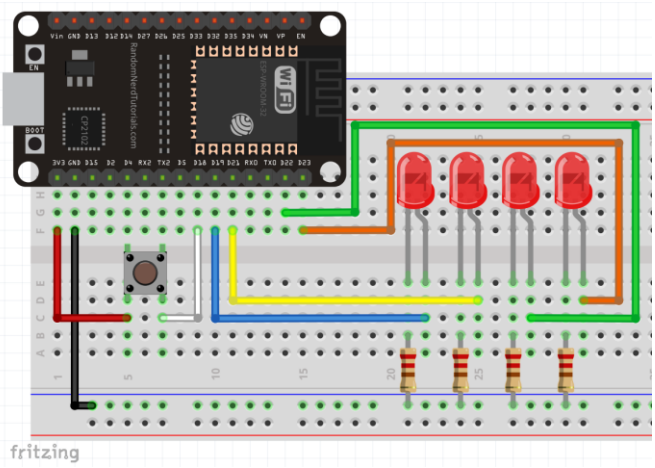


Ilustración 53: Circuito del Ejercicio 3

Como usaremos el monitor Serial del Arduino IDE primero tendremos que configurarlo, para ello vamos a la interfaz de Arduino IDE y en la parte superior derecha hay un botón con un icono de una lupa, damos clic sobre ese botón.

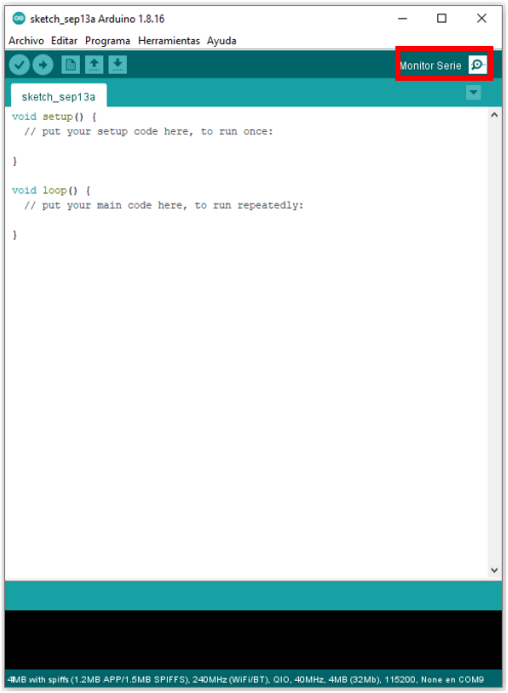


Ilustración 54: Interfaz del Arduino IDE

Se desplegará la siguiente ventana

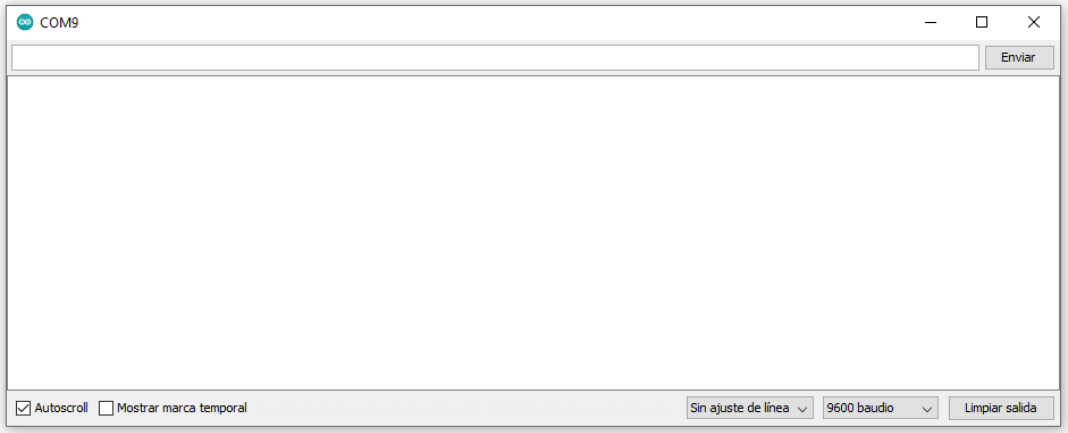


Ilustración 55: Ventana del Monitor Serial de Arduino IDE

Asegurarse que las opciones de la parte inferior derecha estes configuradas como se indica en la imagen

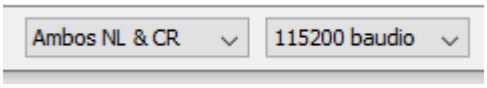


Ilustración 56: Configuración de la ventana del Monitor Serial

En este caso cada vez que el *push button* es presionado mandara el mensaje de "¡Interrupción!" que se mostrara en el monitor serial.

Ilustración 57: Ventana del Monitor Serial mostrando la acción de la interrupción

Trabajo de casa 3 de la practica 2: Periféricos internos del ESP32 (ADC, DAC, PWM, Touch sensors)

1. Mencione los diferentes tipos de convertidores analógico-digital (ADC)
2. De la pregunta anterior, describa el funcionamiento de los diferentes tipos de convertidores analógico-digital
3. ¿En qué tipo de aplicaciones podemos encontrar un convertidor analógico-digital?
4. Describa el funcionamiento de un convertidor digital-analógico (DAC)
5. ¿Cuáles son los aparatos en que podemos encontrar un convertidor digital analógico?
6. ¿Cuál es el funcionamiento básico de una señal PWM?
7. ¿En qué aplicaciones se puede usar una señal PWM?
8. ¿Qué tipos de sensores táctiles o touch existen?
9. Describa el funcionamiento de los tipos de sensores de la pregunta anterior

Practica 2: Periféricos internos del ESP32 (ADC, DAC, PWM, Touch sensors)

Objetivos:

- Hacer que el alumno conozca los periféricos internos del ESP32
- Conocer como configurar los periféricos ADC Y DAC para señales analógicas
- Comprender como generar una señal PWM en el ESP32
- Saber cómo es el funcionamiento del Touch sensor del ESP32

Introducción:

ADC (Convertidor Analógico Digital)

Un conversor analógico digital es un dispositivo electrónico que convierte una señal eléctrica de tensión o corriente del dominio analógico al dominio digital por el medio de un cuantificador y codificándose en un código binario el cual puede ser procesado, manipulado, computarizado, transmitido o almacenado.[26]

El ESP32 posee 2 convertidores Analógico-Digital (ADC) con 18 canales, es decir, 18 pines del ESP32 están conectados al convertidor, pero en el PCB no todos están implementados, algunos tienen conexiones internas con otros componentes por lo que no los podremos usar, los pines que podemos usar con el convertidor están marcados con la nomenclatura ADC1 CH# y ADC2 CH#, lo que nos deja con un total de 15 puertos, (marcados con recuadros rojos).

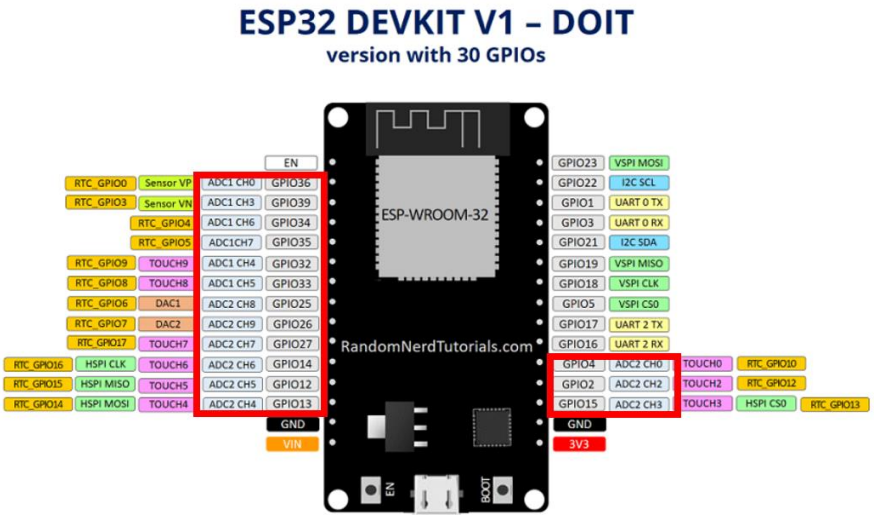


Ilustración 58: Ubicación de pines del ADC del ESP32

El convertidor Analógico-Digital tiene una resolución de 12bits lo que puede tomar un valor de 0 a 4,095 con un voltaje de 0v hasta 3.3v lo cual ese es el voltaje máximo permitido, si es más voltaje corremos el riesgo de dañar el ESP32.

Nota: El convertidor ADC2 no puede ser usado si esta activado el Wi-Fi.

El convertidor Analógico-Digital es de tipo SAR (*Successive-approximation-register*) Conversor Analógico-Digital de Aproximaciones sucesivas, esto quiere decir que el convertidor estará comparando sucesivamente hasta que el resultado sea igual o parecido al voltaje de entrada, por ejemplo:

Si tenemos un convertidor analógico digital de 4 bits (D0-D1-D2-D3) y el voltaje de entrada es de 10 V, el "SAR" iniciara una cuenta binaria empezando con todos los registros Dn en 0, después el registro con más peso (D3) cambiara su valor en 1, dando un valor de 1000 en binario que es igual a 8 en decimal, como el valor esta debajo del Vin, el "SAR" cambia ahora el siguiente bit (D2) en 1, dando como resultado 1100 en binario que es igual a 12 en decimal, como ahora está por encima del Vin, vuelve a cambiar el bit (D3) a 0 y el bit (D2) lo deja igual, dando como resultado 1010 en binario que es igual a 10 en decimal, como ahora el Vin es igual al valor del "SAR" termina de hacer la aproximación y termina con el resultado de 1010.

DAC (Convertidor Digital-Analógico)

Es un dispositivo electrónico que convierte información digital (bits) a una señal analógica que puede ser tensión, corriente o carga eléctrica.

El ESP32 posee 2 convertidores digital analógico de 8 bits de resolución lo que nos da la posibilidad de transmitir señales analógicas por dos pines del ESP32 con un voltaje de 0 a 3.3v, los pines están marcados como DAC1 y DAC2 (Recuadro rojo).

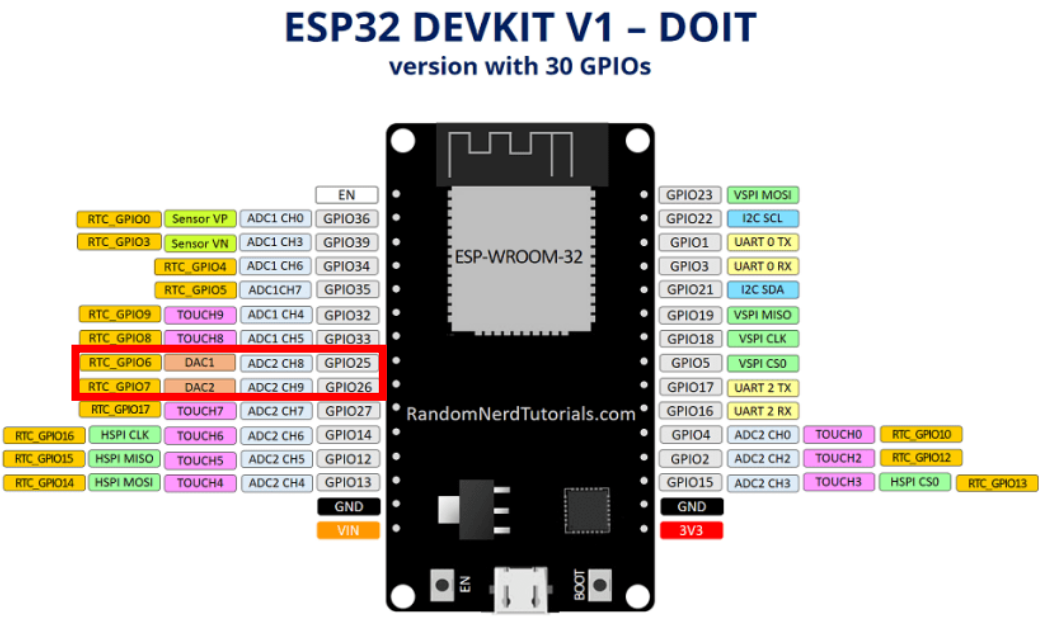


Ilustración 59: Ubicación de pines del DAC del ESP32

PWM

Es una técnica en la que se modifica el ciclo de trabajo de una señal periódica para transmitir señales analógicas cuya señal portadora será digital o para controlar la cantidad de energía que se envía a una carga.[27]

El ESP32 posee dos módulos PWM uno para controlar servomotores y leds inteligentes, y otro exclusivamente usado para LED´s, el controlador PWM consta de temporizadores, operadores y un submódulo de captura dedicado, capaz de proporcionar sincronización síncrona o independiente.

EL módulo PWM para LED puede generar hasta 16 canales independientes con un reloj de 80Mz y cada canal puede seleccionar un temporizador de 20 bits con conteo configurable.

Cualquier pin GPIO puede generar una señal PWM, pero se debe de tener cuidado con ciertos pines que pueden generar una señal inestable (Véase tabla 2 de la práctica anterior).

Sensores táctiles capacitivos GPIOs

Un sensor táctil es un dispositivo que mide la información del contacto físico entre objetos u otros dispositivos, esto le permite saber si hay un contacto o proximidad cercana generalmente de un humano a un dispositivo.[28]

La detección capacitiva se basa en una capacitancia ideal entre dos placas, cuando un objeto que sea conductivo o tenga una diferencia dieléctrica se acerca o toca las dos placas, su capacitancia cambiara en función del objeto, ese cambio de capacitancia se puede medir y nos hace saber si hubo una interacción o contacto entre el dispositivo y un objeto.[29]

El ESP32 tiene 10 sensores táctiles capacitivos, es decir, 10 pines especiales que pueden detectar cuando son tocados por un dedo humano u otros objetos siempre y cuando tengan una carga eléctrica, se pueden usar ya sea de forma individual o en malla, los pines que se pueden usar para esta función están marcados como TOUCH#.

Nota: El ESP32 versión con 30 GPIOs solo posee 9 pines táctiles, la versión con 36 GPIOs si posee los 10 pines táctiles.

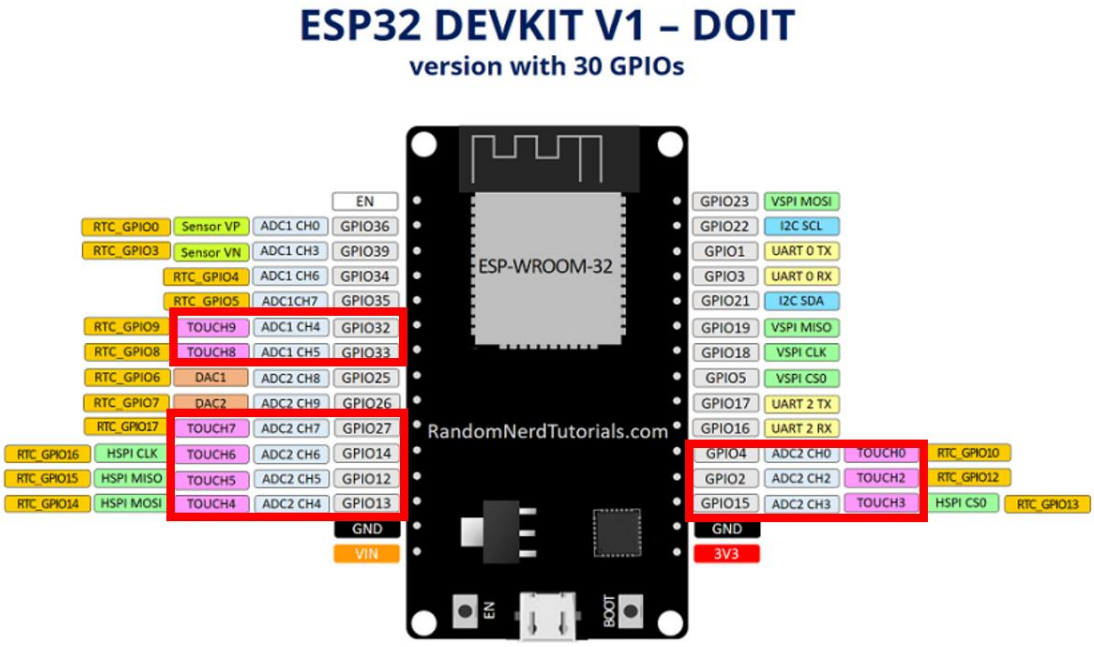


Ilustración 60: Ubicación del Sensor Touch del ESP32

Material para el desarrollo de la práctica:

- Protoboard
- 5 leds de colores de 5mm

- 5 resistencias de 220ohm ½ watt
- Jumpers
- Placa ESP32
- Potenciómetro de 10k ohm
- Cable micro USB

Desarrollo:

Ejercicio 1 ADC:

Para usar el conversor analógico digital y ver su funcionamiento transcribiremos el siguiente código al IDE de Arduino:

```
// Configuramos el pin 34 que será donde conectaremos el
Potenciómetro
const int potPin = 34;

// Variable donde guardaremos el valor del Potenciómetro
int potValor = 0;

void setup() {
  Serial.begin(115200); //inicializamos el monitor serial
  delay(1000);
}
void loop() {
  // Leemos el valor del potenciómetro y lo guardamos en
potVAlor
  potValor = analogRead(potPin);
  Serial.println(potValor); //Imprimimos el valor leído en el
monitor serial.
  delay(500);
}
```

Compilaremos y subiremos el programa al ESP32, después armaremos el siguiente circuito:

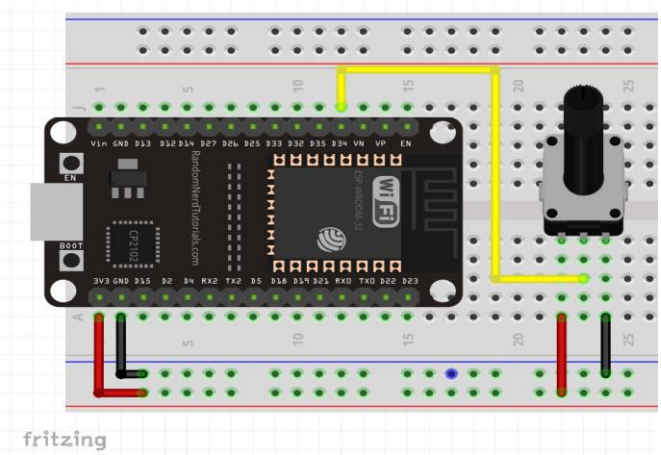


Ilustración 61: Diagrama de conexión del Ejercicio 1

Desconectamos y conectamos el ESP32 de la computadora o presionamos el botón "EN" para reiniciar el ESP32. Después abrimos el monitor serial del IDE de Arduino y se debe de observar que el valor registrado en el monitor serial varía dependiendo de la posición del potenciómetro cambiando de un valor de 0 a 4095.

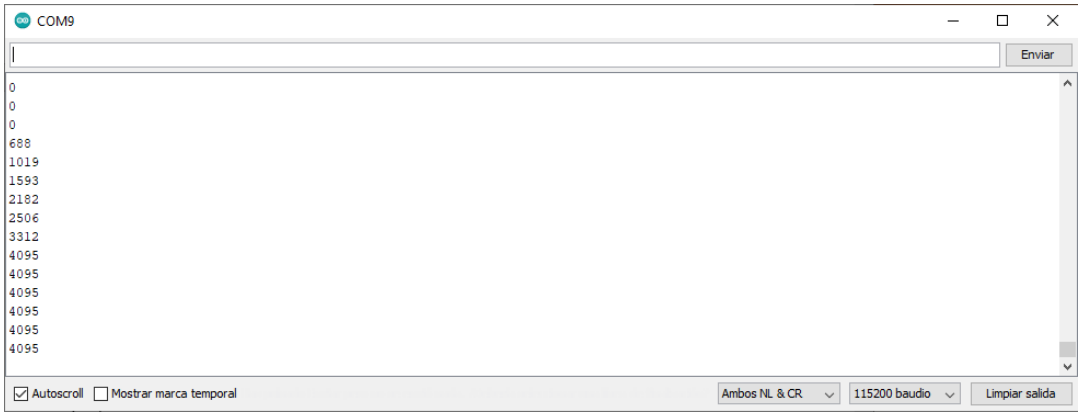


Ilustración 62: Comprobación del ADC

Ejercicio 2 PWM:

En este ejercicio nos enfocaremos en el uso de PWM para leds, por lo cual usaremos el siguiente código:

```
// Definimos el pin 16 con el nombre ledPin
const int ledPin = 16;  // 16 corresponde al pin GPIO16
// Definimos las propiedades de la señal PWM
const int frecuencia= 5000;
const int ledChannel = 0;
const int resolution = 8;
```

```
void setup(){
// Configuramos que tendrá que hacer la señal PWM
ledcSetup(ledChannel, frecuencia, resolution);

// Configuramos el Puerto GPIO para que pueda sacar la señal
PWM
ledcAttachPin(ledPin, ledChannel);
}
void loop(){
// Incrementa el brillo del LED
for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
// Le asignamos al LED la señal PWM
ledcWrite(ledChannel, dutyCycle);
delay(15);
}
// Decremento del billo del LED
for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
// Le asignamos al LED la señal PWM
ledcWrite(ledChannel, dutyCycle);
delay(15);
}
}
```

Una vez transcrito el código lo compilamos y cargamos al ESP32, después aremos el siguiente circuito para comprobar su funcionamiento.

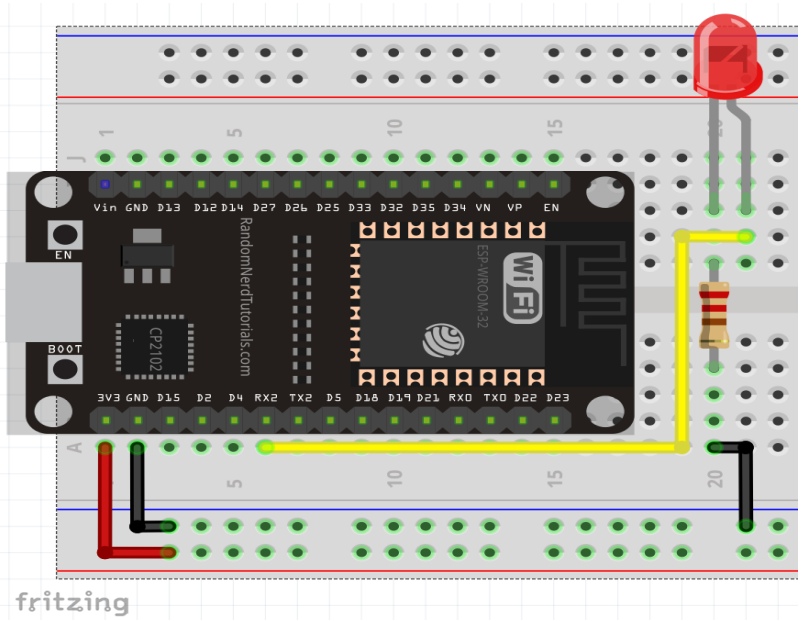


Ilustración 63: Diagrama de conexión para el Ejercicio 2

El led tendrá que incrementar su brillo y reducirlo lentamente.

Ejercicio 3 DAC:

Antes de usar el DAC (Convertidor digital-analógico) debemos tener un osciloscopio ya que nos ayudaremos de este para visualizar la forma de onda que es capaz de proporcionarnos el ESP32

Para usar el DAC usaremos el siguiente código:

```
void setup() { //No es necesario poner nada en el setup
}
void loop() {
  for (int deg = 0; deg < 360; deg = deg + 1){ // haremos un
ciclo for para que podamos "dibujar" la forma de onda
    dacWrite(25, int(128 + 127 * (sin(deg*PI/180))));
    //usamos la función "dacWrite" para usar el DAC
    //El primer número será el puerto donde sacaremos la
señal analógica
    //El segundo número es una expresión matemática para que
sepa el programa que tipo de onda queremos
    dacWrite(26, int(128 + 127 * (cos(deg*PI*4/180))));
  }
}
```

Para comprobar su funcionamiento solo es necesario que el ESP32 esté conectado al USB y que los pines 25 y 26 estén conectados a los canales A y B de un osciloscopio.

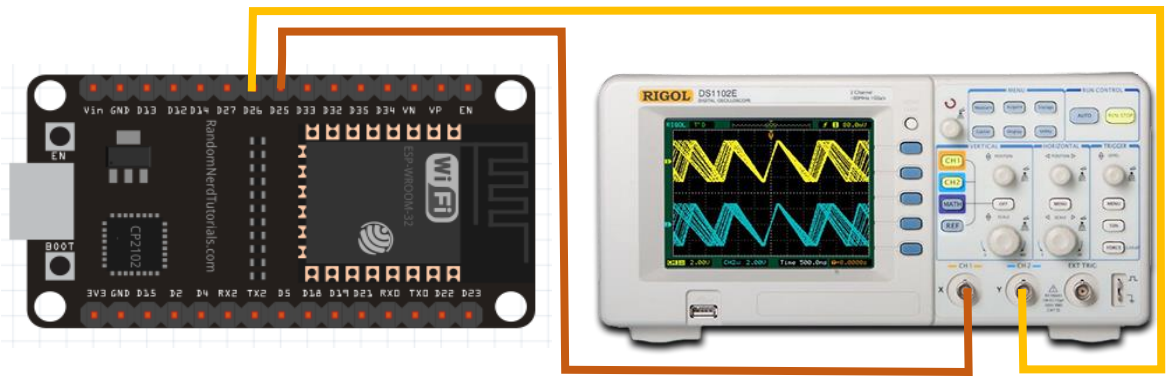


Ilustración 64: Diagrama de conexión del ESP32 a un osciloscopio

Se deberá mostrar una forma de onda como la siguiente:

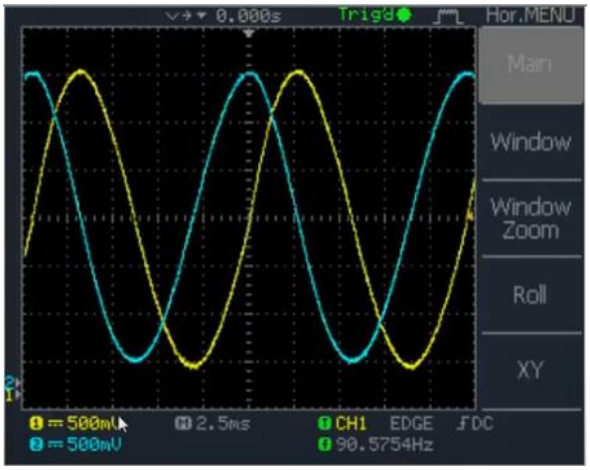


Ilustración 65: Tipo de señales que se deben observar en el osciloscopio

Ejercicio 4 Sensor Touch:

Para usar el sensor touch del ESP32 primero debemos saber cuál es la función a usar la cual es la siguiente:

```
touchRead(GPIO);
```

Con esta simple instrucción el pin GPIO se podrá usar como un sensor touch, el valor GPIO dependerá del pin que ocuparemos y que tenga la posibilidad de usarse como tal.

En este ejercicio usaremos el sensor touch para prender o apagar un LED como si fuera un interruptor[30].

Transcribiremos el siguiente código al IDE de Arduino:

```
// configuramos los pines a usar
const int touchPin = 4;
const int ledPin = 16;

// valor base del sensor
const int valorBase = 25;
// variable donde se guardará el valor del sensor touch
int variableTouch ;

void setup(){
  Serial.begin(115200);// inicializamos el monitor serial
  delay(1000);
  // ponemos el pin del LED como salida
  pinMode (ledPin, OUTPUT);
}
```

```
void loop(){
  // leemos el valor del sensor touch y lo guaramos en la
variable variableTouch
  variableTouch = touchRead(touchPin);
  Serial.print(variableTouch);
  // Imprimimos el valor leído del sensor en el monitor
serial
  // hacemos la siguiente condición si el valor base es mayor
que el valor leído encenderá el led, en caso contrario el led
estará apagado
  if(variableTouch < valorBase){
    // Encendemos el LED
    digitalWrite(ledPin, HIGH);
    Serial.println(" - LED on"); //Imprimimos en el monitor
serial que el LED este encendido
  }
  else{
    // Apagamos el LED
    digitalWrite(ledPin, LOW);
    Serial.println(" - LED off");// Imprimimos en el monitor
serial que el LED esta apagado.
  }
  delay(100);
}
```

Una vez transcrito armaremos el siguiente diagrama:

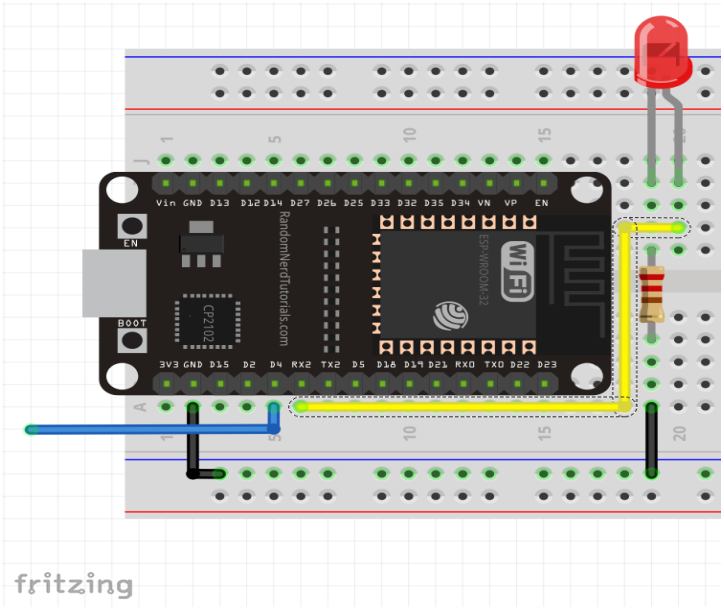


Ilustración 66: Diagrama de conexión para usar el Sensor Touch del ESP32

El cable azul será un jumper conectado de un extremo al pin 4 del ESP32 y del otro extremo quedara sin conectar para que cuando se toque con un dedo el LED encenderá y cuando no este tocando con el dedo, el LED quedara apagado.

Trabajo de casa 4 de la practica 3: Bluetooth y Wi-Fi configuración básica en el ESP32

1. Mencione brevemente cómo funciona el Bluetooth
2. ¿Cuál es la frecuencia que usa el Bluetooth?
3. Mencione cuales son las clases y versiones del Bluetooth
4. Describa los diferentes modos de trabajo del Bluetooth maestro y esclavo
5. ¿En qué aparatos o dispositivos podemos encontrar el Bluetooth?
6. Describa cómo funciona una red Wi-Fi
7. ¿Cuáles son los diferentes estándares de una red Wi-Fi?
8. ¿Cuál es la frecuencia que usa la red en Wi-Fi?
9. ¿Cuáles son las desventajas del Wi-Fi?
10. ¿En qué aparatos o dispositivos se usa el Wi-Fi?

Practica 3: Bluetooth y Wi-Fi configuración básica en el ESP32

Objetivos:

- Que el alumno conozca las nociones básicas de la comunicación Bluetooth, así como surgió.
- Configurar el módulo interno Bluetooth del ESP32
- Hacer que el alumno conozca la comunicación Wi-Fi así como un poco de su historia
- Configurar el módulo Wi-Fi interno del ESP32

Introducción:

Bluetooth

Es un estándar global de hardware y software de comunicación inalámbrica de corto alcance, diseñado en un inicio para la comunicación de audio sin usar cables, ahora es ampliamente usado para compartir información en forma de datos simples o no tan complejos entre dos dispositivos como, por ejemplo, imágenes, texto, música, etc.

Los inicios del Bluetooth se remontan desde el año de 1994 con la compañía de telecomunicación ERICSSON, la cual quería desarrollar una interface de comunicación de radio de bajo costo entre teléfonos móviles y sus accesorios. Su principal motivo fue eliminar los cables físicos que se conectaban para transferir información entre un teléfono móvil y una computadora o unos audífonos. Su investigación los llevo a un sistema de comunicación por radio de corto alcance el cual llamaron MC link.[31]

No fue hasta febrero de 1998 que las compañías Nokia, IBM, Toshiba, Intel y Ericsson fundaron el *Bluetooth Group of Special Interest* (SIG) creando un conjunto de áreas de negocio: dos lideres del mercado en fabricación de Computadoras (IBM y Toshiba), dos lideres del mercado en las telecomunicaciones de la época (Nokia y Ericsson) y un líder en la fabricación de microchips (Intel), el propósito principal fue establecer que el "Bluetooth" sea un estándar y un software de comunicación entre dispositivos móviles, computadoras y sus periféricos, asegurando una gran compatibilidad entre los dispositivos de otros fabricantes. Mas tarde en el año de 1999 otras empresas se unieron al grupo las cuales por mencionar algunas son, Microsoft, 3com, Compaq, Dell, Motorola, Xircom y otros.[32]

El primer teléfono móvil que fue anunciado en incorporar el Bluetooth fue el Ericsson T39 en el año 2000 pero por ciertas razones tuvo que ser cancelado el lanzamiento y no fue sino hasta el siguiente año que el Ericsson T68m fue lanzado comercialmente y se le atribuye ser el primer dispositivo comercial que incluía el “Bluetooth”. En paralelo, IBM introducía al mercado el IBM ThinkPad A30 en octubre del 2001 el cual fue la primera Laptop que integraba el Bluetooth.[33], [34]

El nombre “Bluetooth” es un homenaje al rey danés y noruego Harald Blåtand que en ingles se escribe como Harold Bluetooth, conocido por ser un buen comunicador y unificar tribus noruegas, suecas y danesas aproximadamente en el año de 970, también el logo del Bluetooth son las runas hagall y berkana "HB" las cuales son las iniciales del rey Haral.[35], [36]

Ilustración 67: Nomenclatura del logo Bluetooth

El ESP32 integra un controlador de enlace Bluetooth y una banda base Bluetooth, los cuales se encargan de los protocolos de banda base, rutinas de enlace, modulación/demodulación, procesamiento de paquetes, flujo de bits, saltos de frecuencia, etc.

El Radio Bluetooth y la banda base soportan las siguientes características[37]:

- Potencias de salida de transmisión de clase 1, clase 2 y clase 3, y un rango de control dinámico de hasta 21 dB
- Modulación π / 4 DQPSK y 8 DPSK
- Alto rendimiento en la sensibilidad del receptor NZIF con una sensibilidad mínima de -94 dBm
- Funcionamiento de clase 1 sin PA externo
- Proporciona interfaz UART HCI, hasta 4 Mbps
- Proporciona interfaz SDIO / SPI HCI

- Proporciona interfaz de audio PCM / I²S

Básicamente si se tiene experiencia usando el módulo HC-06 o el HC-05 comúnmente usado con la placa de Arduino, usar el Bluetooth del ESP32 se notarán muchas similitudes ya que esencialmente son lo mismo.

Wi-Fi

Es un tipo de red inalámbrica la cual tiene como objetivo compartir información de manera inalámbrica entre dos o más dispositivos, comúnmente usada para la comunicación de internet entre dispositivos móviles como Smartphones o laptops.[38]

Los inicios del Wi-Fi se remontan desde el año 1985 cuando el departamento *Federal Communications Commission* de los Estados unidos liberaron las bandas del espectro de radio en 900 MHZ, 2.4 GHz y 5.8 GHz para el uso sin licencia lo cual quiere decir que cualquier persona podía usar dichas bandas.[39], [40]

Pero al ser bandas libres había un caos en el uso de dichas frecuencias, porque cada fabricante tenía su estándar y no había compatibilidad entre dispositivos de diferentes compañías. No fue hasta el año de 1997 que se creó un comité llamado 802.11 aprobado por el *Institute of Electrical and Electronics Engineers* (IEEE) el cual era un estándar de comunicación inalámbrica que unificaría las diferentes tecnologías de los diferentes fabricantes haciendo que fuera común para todos los dispositivos del mercado. Mas tarde formaron la *Wireless Ethernet Compatibility Alliance* (WECA) ahora llamada Wi-Fi Alliance, el cual tenía como propósito fomentar el uso del nuevo entandar de comunicación Wi-Fi.[41], [42]

Ilustración 68: Logo de la Wi-Fi Alliance

El ESP32 implementa un TCP/IP compatible con el estándar 802.11 b/g/n con un protocolo Wi-Fi MAC, soportando el *Basic Service Set* (BSS) STA y SoftAP. Teniendo una

administración de energía el cual se encarga de minimizar lo más posible el uso de la misma.[43]

El Wi-Fi del ESP32 posee las siguientes características técnicas:

- 802.11b/g/n
- 802.11n MCS0-7 justo a la banda de 20 MHz y 40 MHz
- 802.11n MCS32 (RX)
- 802.11n 0.4 µs Intervalo de guardia
- Transferencia de datos hasta 150 Mbps
- Hasta un poder de transmisión de 20.5 dBm
- Potencia de transmisión ajustable.
- Diversidad de antenas (Quiere decir que uno o más de los pines GPIOs pueden ser seleccionados como antena para minimizar los efectos del desvanecimiento del canal)
- 4 interfaces Wi-Fi virtuales
- Protección RTS, protección CTS, ACK de bloqueo inmediato
- Desfragmentación
- TX / RX A-MPDU, RX A-MSDU

Material para el desarrollo de la práctica:

- Protoboard
- Placa ESP32
- Smartphone con Android 6.0 o superior (IPhone no es compatible)
- Cable micro USB

Desarrollo:

Ejercicio 1 Bluetooth:

Antes de la realización de este ejercicio se recomienda descargar la aplicación “Serial Bluetooth Terminal” disponible en la Play Store de Android.

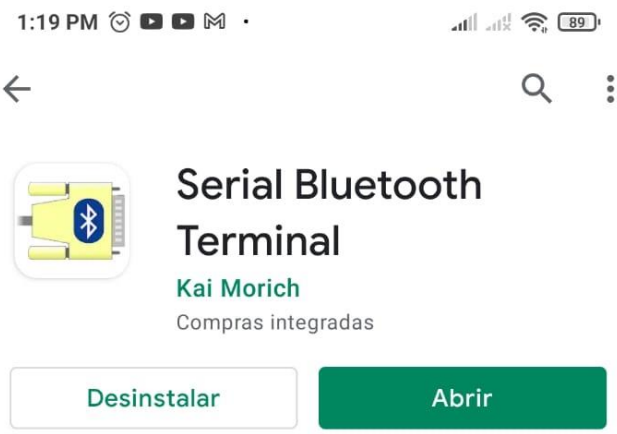


Ilustración 69: Aplicación "Serial Bluetooth Terminal"

Comenzaremos con una comunicación serial Bluetooth entre Smartphone y computadora, para eso usaremos el siguiente código[44]:

```
#include "BluetoothSerial.h" //Incluimos la librería del
Bluetooth Serial

//Checamos si el Bluetooth se habilito correctamente
#if !defined(CONFIG_BT_ENABLED) ||
!defined(CONFIG_BLUEDROID_ENABLED)
#error ¡Bluetooth no está habilitado! Ejecute `make
menuconfig` y habilítelo
#endif

BluetoothSerial SerialBT; //llamamos al Bluetooth Serial como
"Serial BT"

void setup() {
  Serial.begin(115200); // Inicializamos el monitor serial
  SerialBT.begin("ESP32test"); //Nombre del Bluetooth
  Serial.println("Dispositivo iniciado y listo para ser
emparejado con Bluetooth");
}

void loop() {
  //Esta condición es para que, si hay datos del monitor
serial, estos sean enviados al Smartphone
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  //Esta condición es para que, si hay datos del Smartphone,
estos sean enviados al monitor serial
```

```
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20); //pequeño delay para que no haya problemas de
comunicación.
}
```

¿Como funciona el código?

El código anterior entabla una comunicación serial Bluetooth entre dos dispositivos

Inicia con la inclusión de la librería “BluetoothSerial”

```
#include "BluetoothSerial.h"
```

La siguiente línea es una comprobación si el Bluetooth se inició correctamente, se puede omitir, pero se recomienda usarla siempre

```
#if !defined(CONFIG_BT_ENABLED) ||
!defined(CONFIG_BLUEDROID_ENABLED)
#error ¡Bluetooth no está habilitado! Ejecute `make menuconfig` y
habilítelo
#endif
```

En el Setup del programa configuramos los siguientes parámetros:

Cambiamos el nombre de BluetoothSerial a SerialBT para hacerlo lo más sencillo posible y no escribir todo el nombre, igual se puede cambiar por cualquier otro nombre o simplemente usar el que está por defecto

```
BluetoothSerial SerialBT;
```

Inicializamos la comunicación serial con una tasa de baudios de 115200

```
Serial.begin(115200);
```

Inicializamos el Bluetooth serial y le asignamos un nombre en este caso es “ESP32test” pero se puede cambiar el nombre por el que el usuario desee

```
SerialBT.begin("ESP32test");
```

Mostramos un pequeño mensaje de que todo está listo

```
Serial.println("Dispositivo iniciado y listo para ser
emparejado con Bluetooth");
```

En el loop se encargará de enviar y recibir la información vía Bluetooth serial, el cual consta de dos partes

La primera se encarga de checar si hay datos en el monitor serial para enviar, si hay manda la información atraves del Bluetooth al dispositivo conectado

```
if (Serial.available()) {
     SerialBT.write(Serial.read());
  }
```

Usamos los siguientes comandos para hacer la lectura y escritura de datos:

```
SerialBT.write(); //Envía los datos usando el Bluetooth serial
Serial.read(); //Lee la información recibida del monitor serial
```

La segunda parte se encarga de lo contrario a la anterior, checa si hay datos del Bluetooth serial, si hay, mandara esos datos al monitor serial

```
if (SerialBT.available()) {
     Serial.write(SerialBT.read());
  }
```

Una vez transcrito el código comenzaremos a testearlo para ver su funcionamiento para eso primero iniciaremos el monitor serial y presionamos el botón “EN” del ESP32 para reiniciarlo y nos muestre el siguiente mensaje:

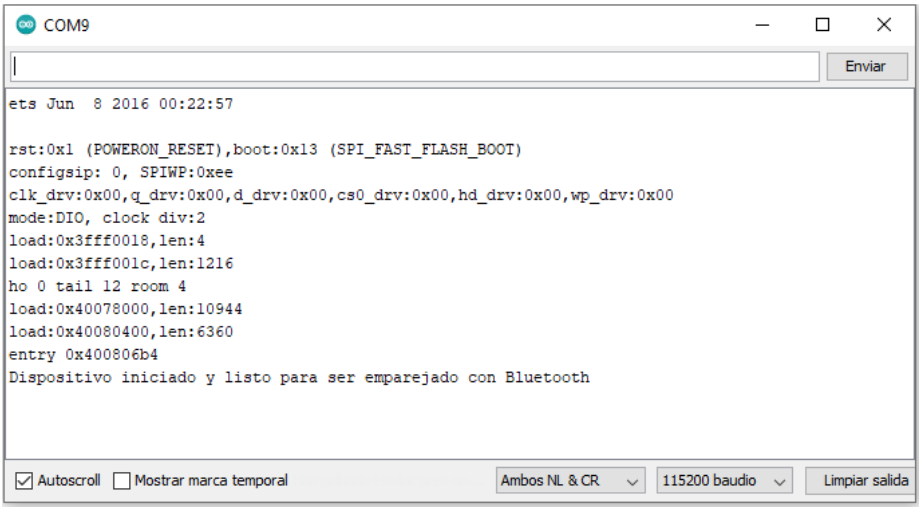


Ilustración 70: Comprobación del código del Ejercicio 1

Si todo está en orden y el mensaje se muestra bien iremos ahora a nuestro Smartphone y nos dirigimos a las configuraciones del Bluetooth para vincular un nuevo dispositivo:

Nota: Los siguientes pasos están basados en un Smartphone con Android MIUI 12.0.10, en otros dispositivos los pasos pueden cambiar ligeramente.

Vamos a configuraciones del cajón de aplicaciones.

Ilustración 71: Icono de configuración en Android

Nos dirigimos al apartado Bluetooth.

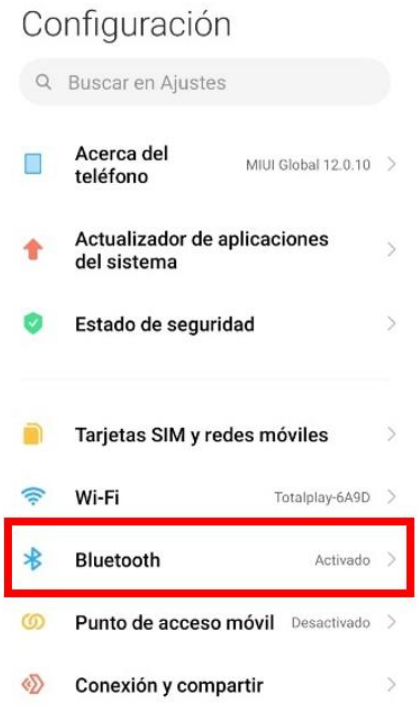


Ilustración 72: Ubicación del apartado Bluetooth

Esperamos hasta que aparezca el nombre de nuestro ESP32

Ilustración 73: Nombre del Bluetooth del ESP32 encontrado por el Smartphone

Nos preguntara si deseamos vincular el dispositivo, tocamos la opción de vincular

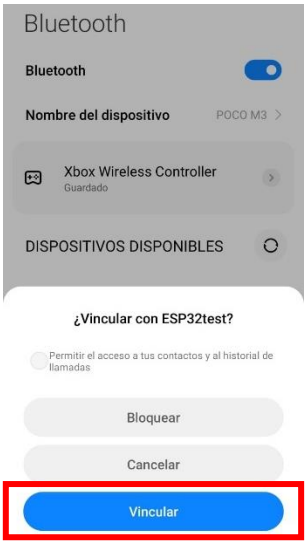


Ilustración 74: Ubicación del botón vincular

Si todo está bien, aparecerá en nuestra lista de dispositivos vinculados

Ilustración 75: Comprobación de que la vinculación fue exitosa

Ahora nos dirigimos a la aplicación “Serial Bluetooth terminal” que descargamos previamente

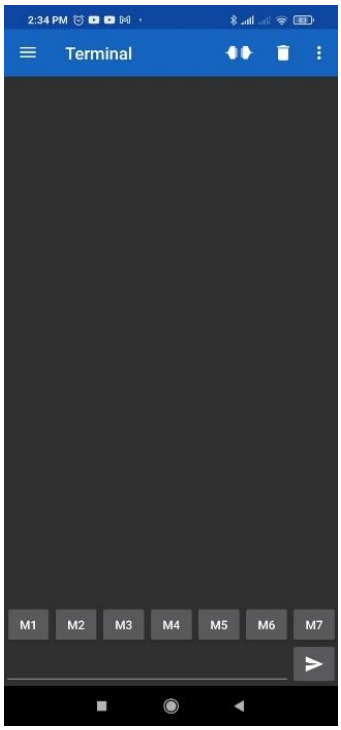


Ilustración 76: Interface de la aplicación "Serial Bluetooth Terminal"

En la parte superior izquierda hay un botón con tres líneas, lo seleccionamos, nos mostrara las siguientes opciones y Seleccionamos “Devices”

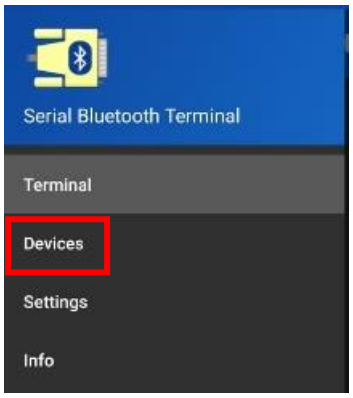


Ilustración 77: Selección del apartado "Devices"

Nos mostrara una lista de nuestros dispositivos vinculados, seleccionamos el de nuestro ESP32

Ilustración 78: Selección del Bluetooth del ESP32

Si la conexión es exitosa se mostrará el siguiente mensaje

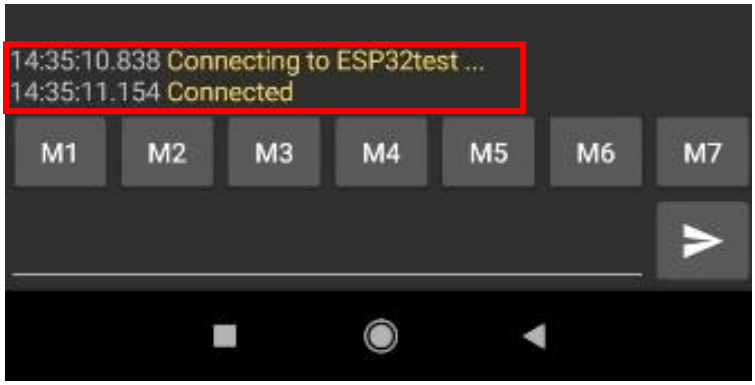


Ilustración 79: Mensaje de conexión exitosa entre el Smartphone y el ESP32

Todo listo, ahora podemos enviar datos en forma de texto entre ambos dispositivos, empezamos escribiendo la palabra “hola mundo” y presionamos el botón de mandar (Cuadro rojo).

Ilustración 80: Envió de datos del Smartphone al ESP32

En el monitor serial se debe de mostrar el mismo mensaje

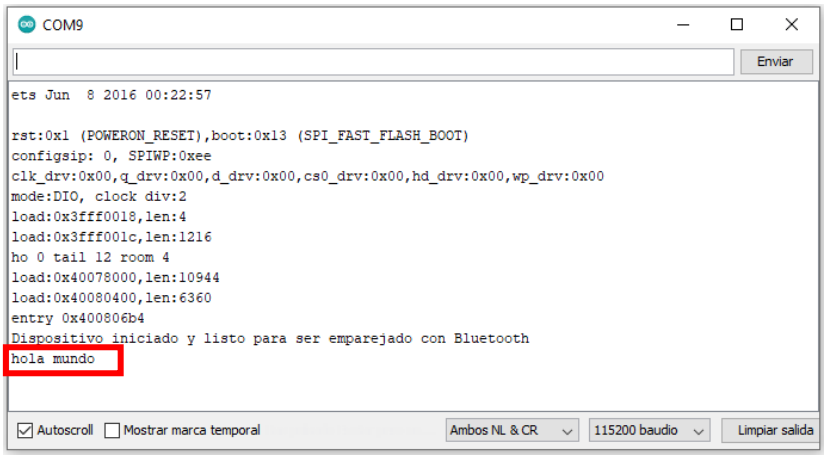


Ilustración 81: Comprobación de que el mensaje se envió y recibió correctamente

Ahora en el monitor serial escribimos “¿Hola, como estas?” y presionamos la tecla “Enter” o el botón “enviar”

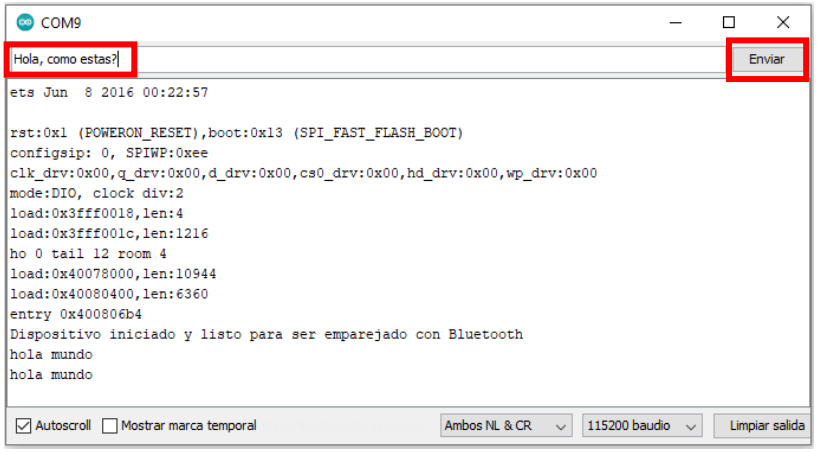


Ilustración 82: Envió de datos del ESP32 al Smartphone

En la aplicación deberá de mostrarse el mismo mensaje, pero con un color distinto y listo, ahora podemos enviar cualquier tipo de mensajes entre ambos dispositivos.

Ilustración 83: Comprobación de que el mensaje se envió y recibió correctamente

Para desconectarlo solo pulsamos el botón marcado en rojo y nos mostrara un mensaje de “Disconnected”

Ilustración 84: Desconexión del ESP32 y del Smartphone

Ejercicio 2 Wi-Fi:

Para la conexión a una red Wi-Fi con el ESP32 primero tenemos que escanear las redes disponibles, para eso nos ayudaremos del siguiente código[45]:

Nota: Para este ejercicio se recomienda el uso de un modem o un Reuter con Wi-Fi propio, no es necesario que tenga conexión a internet, pero si se usan redes públicas sin contraseña puede ser inestable la conexión.

```
#include "Wi-Fi.h" //Iniciamos la librería Wi-Fi

void setup()
{
    Serial.begin(115200); //Inicializamos el monitor Serial

    // Configure Wi-Fi en modo de estación y desconéctese de
un AP si estaba conectado previamente
    Wi-Fi.mode(WI-FI_STA);
    Wi-Fi.disconnect();
    delay(100);
    Serial.println("Configuraciones realizadas con éxito");//
Mostramos un mensaje si todo se configuro correctamente
}

void loop()
{
    Serial.println("Iniciando escaneo"); //Mensaje indicando
que iniciamos el escaneo

    // Wi-Fi.scanNetworks devolverá el número de redes
encontradas
    int n = Wi-Fi.scanNetworks(); //Escaneo y conteo de las
redes Wi-Fi
    Serial.println("Escaneo terminado"); //Cuando termine el
escaneo mostramos un mensaje indicando que termino
    if (n == 0) { //Condición en caso de no encontrar ninguna
red Wi-Fi
        Serial.println("Ninguna red encontrada");
    } else { //Si se encontró al menos una red Wi-Fi
mostramos un mensaje de cuantas redes fueron encontradas
        Serial.print(n);
        Serial.println(" Redes encontradas");
        for (int i = 0; i < n; ++i) {
            // Imprimimos SSID y RSSI para cada red
encontrada y mostrando si cuenta con contraseña con un "*"
```

```
            Serial.print(i + 1);
            Serial.print(": ");
            Serial.print(Wi-Fi.SSID(i));
            Serial.print(" (");
            Serial.print(Wi-Fi.RSSI(i));
            Serial.print(")");
            Serial.println((Wi-Fi.encryptionType(i) == WI-
FI_AUTH_OPEN)?" ":"*");
            delay(10);
        }
    }
    Serial.println("");
    //Esperamos un poco para el siguiente escaneo.
    delay(5000);
}
```

¿Como funciona el código?

El comando Wi-Fi.mode(); nos ayudara para configurar el modo en que el Wi-Fi del ESP32 trabajara, hay varias opciones para ello, las cuales son las siguientes[45]:

- `Wi-Fi.mode(WI-FI_STA);` Station mode: El ESP32 conectara a un Access Point
- `Wi-Fi.mode(WI-FI_AP);` Modo Access Point: El ESP32 se podrá conectar a otro ESP32
- `Wi-Fi.mode(WI-FI_STA_AP);` Modo Access Point y Station, Posibilidad de conectarse a múltiples Access Point.

El comando Wi-Fi.disconnect(); desconectara el ESP32 de una red Wi-Fi en caso de estarlo, se recomienda usarlo para que las conexiones automáticas no interfieran.

Wi-Fi.scanNetworks(); este comando iniciará el escaneo de redes y nos devolverá el valor de las redes encontradas.

Serial.print(Wi-Fi.SSID(i)); este comando imprimirá el nombre de cada red Wi-Fi encontrada

Serial.print(Wi-Fi.RSSI(i)); este comando imprimirá el RSSI el cual es un indicador de la estimación del nivel de poder de la señal Wi-Fi

Una vez que el código se haya cargado al ESP32 abrimos el monitor serial y presionamos el botón “EN” del ESP32, esperamos un poco y nos desplegara la lista de redes encontradas

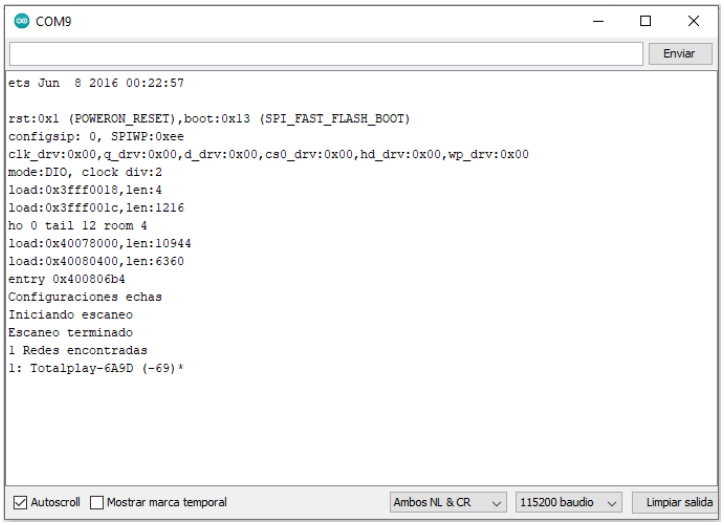


Ilustración 85: Escaneo correcto de las redes Wi-Fi

En este caso solo se encontró una red, aunque el escaneo se repetirá cada 5 segundos y se podrán encontrar más redes Wi-Fi

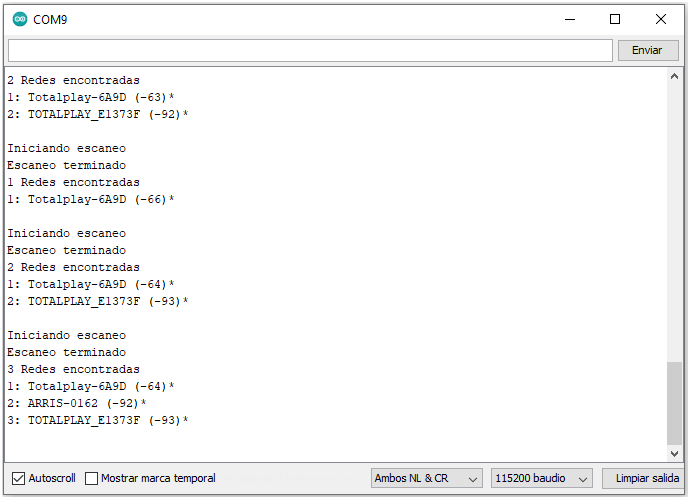


Ilustración 86: Escaneo continuo de las redes Wi-Fi

Ahora nos conectaremos a una red Wi-Fi, en este caso será la red con nombre "Totalplay-6A9D" para ello se usará el siguiente código

```
#include <Wi-Fi.h> //Usamos la librería Wi-Fi

// Reemplaza los valores entre comillas con los valores de la
red Wi-Fi que desee conectarse
const char* ssid = "Totalplay-6A9D"; //Nombre de la red Wi-Fi
const char* password = "Pucky4patas"; //Contraseña de la red
Wi-Fi

//Se necesitará un pequeño "programa" para la conexión a la
red Wi-Fi
void initWi-Fi() {
  Wi-Fi.mode(WI-FI_STA); //Ponemos el ESP32 en modo Station
Mode
  Wi-Fi.begin(ssid, password); //Conectaremos a la red usando
el nombre y la contraseña anteriormente escrita
  Serial.print("Conectando a la red Wi-Fi ..");
  //Esperemos hasta que la conexión se establezca
correctamente y tengamos una dirección IP
  while (Wi-Fi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  //Imprimimos la dirección IP que el modem o Router le
asigno al ESP32
  Serial.println("");
  Serial.print("Direccion IP: ");
  Serial.println(Wi-Fi.localIP());
}

void setup() {
  Serial.begin(115200); //Inicializamos el Monitor Serial
  initWi-Fi(); //Ejecutamos el pequeño "programa" para la
conexión a la red Wi-Fi
  //Imprimimos el RSSI para saber a qué potencia de
transmisión se encuentra la red Wi-Fi
  Serial.print("RRSI: ");
  Serial.println(Wi-Fi.RSSI());
}

void loop() {
}
```

Para ver si el código funciono correctamente iniciamos el monitor serial y presionamos el botón “EN” del ESP32 y nos debería mostrar algo como lo siguiente:

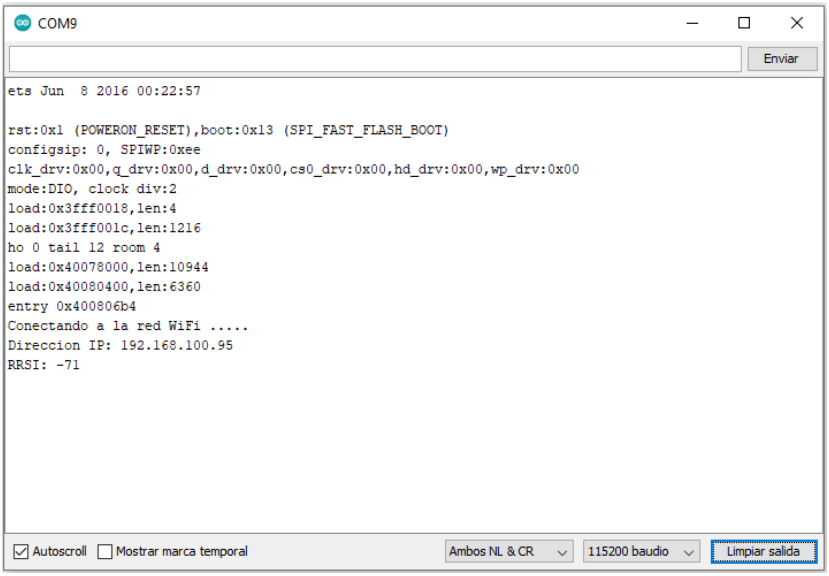


Ilustración 87: Conexión exitosa entre el ESP32 a la red Wi-Fi

En caso de que el programa se quede en “conectando...” indica que el nombre de la red o contraseña están mal escritas o existe otro tipo de error en la conexión.

Trabajo de casa 5 de la practica 4: CPU de doble núcleo del ESP32

1. ¿Cuál fue el primer CPU fabricado con transistores?
2. Describa los componentes internos de un CPU
3. ¿Cuál es la diferencia entre un microprocesador y un microcontrolador?
4. ¿Aparte de una computadora, en que otros dispositivos podemos encontrar un microprocesador?
5. ¿A qué se refiere con que un procesador es de 4, 8, 16 o 32 bits?
6. Haga un diagrama a bloques de la comunicación interna en un microprocesador de doble núcleo
7. Mencione las diferencias entre un núcleo físico y uno lógico
8. ¿Qué es FreeRTOS?
9. ¿Qué es un “task” en programación?

Practica 4: CPU de doble núcleo del ESP32

Objetivos:

- Que el alumno comprenda el concepto de "CPU de doble núcleo" en microprocesadores
- Hacer que el alumno configure y use el doble núcleo del CPU del ESP32
- Comprender a manera básica que es FreeRTOS
- Crear y gestionar "tareas" usando FreeRTOS en el ESP32

Introducción

CPU (*Central Processing Unit*) o unidad central de procesamiento en español, es un hardware que se encarga de interpretar las instrucciones de un programa informático usando aritmética básica o lógica computacional, usado ampliamente en computadores, smartphones y otros dispositivos que requieran de un procesamiento de datos.[46], [47]

El CPU es la parte más básica dentro de un microprocesador, solo se encarga de las operaciones básicas y se debe de ayudar de otros dispositivos para recopilar datos, almacenarlos o transmitirlos a otros componentes.

Microprocesador:

En términos generales es muy similar a un CPU, hace lo mismo, interpretar instrucciones de un programa informático, pero el microprocesador contiene más dispositivos que le ayudan al control de dichas instrucciones, por ejemplo, el microprocesador se puede conectar directamente a otros dispositivos como lo son monitores o impresoras.[48]

Su velocidad de procesamiento de datos se mide por su frecuencia de reloj medida en Hertzios.

El primer microprocesador comercialmente fue introducido por Intel, llamado Intel 4004 en el año de 1970, solo tenía un CPU de 4bits y una frecuencia de reloj de 470kHz, 4 años después Intel introduciría al mercado un nuevo microprocesador, llamado el Intel 8080

el cual desplazo lentamente el uso de simples CPU´s en los computadores de esa época.[49], [50]

El impacto que tuvo el Intel 8080 en la industria de la computación fue muy grande y sirvió de base para la creación de las primeras computadoras como el Altair 8800 o el IMSAI 8080, también dio el primer paso para su uso en el sector de los videojuegos con el Gun Figth, el primer Arcade en utilizar un microprocesador.[51]

Si bien el microprocesador fue muy usado y sentó las bases de la computación, se tuvo la necesidad de crear nuevas maneras en las cuales los programas informáticos fueran ejecutados, al contener un solo núcleo, el procesador tenía una desventaja al ejecutar varias tareas, solo podía hacerlas una a la vez.

Con la llegada de los computadores personales con Windows o con Mackintosh por el año de 1990 esta limitante de volvió cada vez más notoria, los principales fabricantes de ese entonces solo daban mejoras en la arquitectura y en la velocidad de procesamiento de datos por cual años más tarde nacieron los procesadores de Doble Núcleo.[52]

Un procesador de doble núcleo (*Dual core*) es un microprocesador el cual tiene dos procesadores físicos independientes en el mismo encapsulado, pueden compartir memoria L1 o memoria principal del sistema.[52]

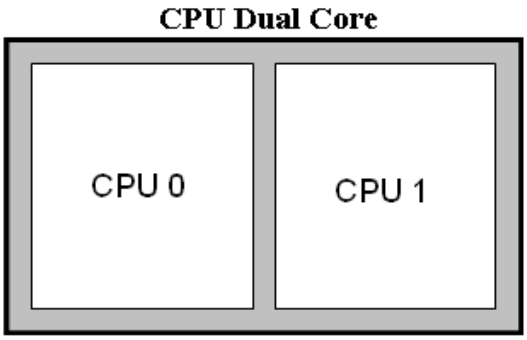


Ilustración 88: Diagrama a bloques de un CPU de Doble Nucleo

Los primeros procesadores comerciales doble núcleo de uso común fueron el Athlon X2 fabricados por AMD y el Intel Pentium Xtreme Edition 840 fabricado por Intel, ambos lanzados en el año 2005.[53]

La principal ventaja de un microprocesador con doble núcleo es la multitarea, es decir, ejecutar diversas tareas de un programa informático de manera simultánea, también en términos energéticos son más eficientes, pueden repartir las tareas entre un núcleo y otro haciéndolos gastar menos energía.[54], [55]

En este caso el ESP32 posee un microprocesador Tensilica Xtensa 32-bit LX6 de doble núcleo 0 y 1.[56]

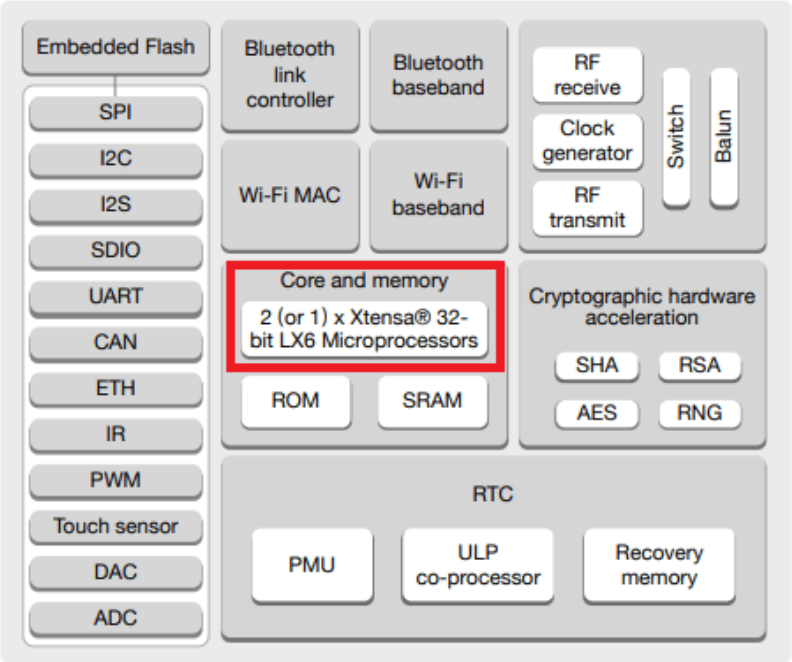


Ilustración 89: Diagrama a bloques del ESP32

El cual tiene las siguientes características técnicas:

- Canalización de 7 etapas para admitir la frecuencia de reloj de hasta 240 MHz o 160 MHz
- Conjunto de instrucciones de 16/24 bits
- Soporte para unidad de coma flotante
- Soporte para instrucciones DSP, como un multiplicador de 32 bits, un divisor de 32 bits y un MAC de 40 bits
- Soporte para 32 vectores de interrupción de aproximadamente 70 fuentes de interrupción

- Interfaz Xtensa RAM / ROM para instrucciones y datos
- Interfaz de memoria local Xtensa para un rápido acceso al registro de periféricos
- Fuentes de interrupción externas e internas
- JTAG para depuración

FreeRTOS en el ESP32

De manera nativa el SoC ESP32 es compatible con FreeRTOS, esto nos simula un sistema operativo muy básico el cual nos permite crear “tareas” o “tasks”, son como mini programas que se ejecutan “simultáneamente” dentro del ESP32, el número máximo de tareas que podemos ejecutar en el ESP32 es difícil de determinar ya que depende de la memoria RAM y de la complejidad de la tarea, un aproximado se puede establecer en 30 pero eran tareas muy básicas.[57]

El funcionamiento de FreeRTOS para la creación de “tareas” se usa el concepto de “Scheduler” o “Calendario” que es el que se encarga de gestionar y ejecutar las tareas que se han creado o programado, para que esto suceda cada tarea contiene un estado, En ejecución, suspendida o bloqueada siguiendo el siguiente diagrama de flujo:[58], [59]

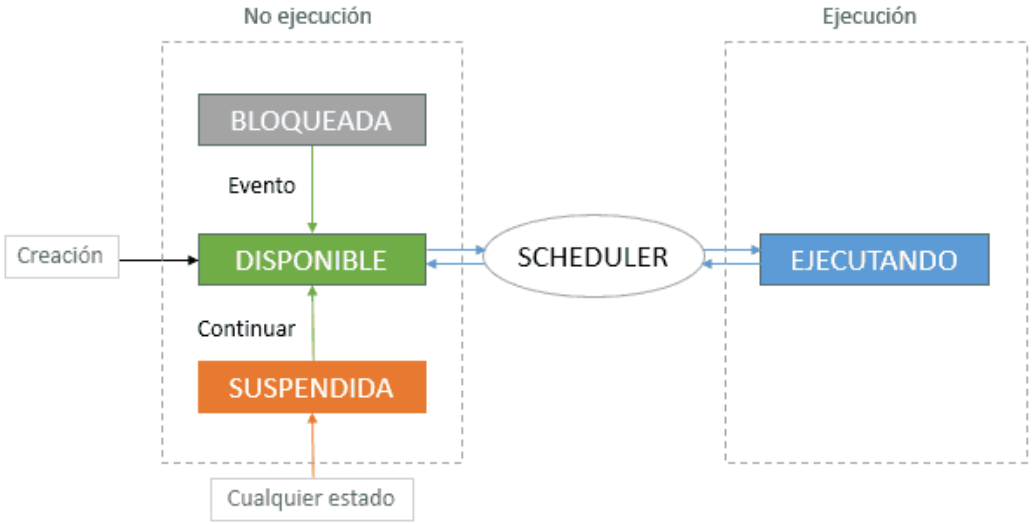


Ilustración 90: Diagrama de flujo del comportamiento de una "tarea" en FreeRTOS

Los estados serían los siguientes:

- Ejecutando: La tarea se está ejecutando en el procesador

- Disponible: La tarea esta lista para pasar a ejecutarse, pero está esperando a que la tarea anterior termine su ejecución en el procesador
- Bloqueada: La tarea esta "esperando" a que un evento, por ejemplo, un evento externo o un timeout
- Suspendida: La tarea está en "stop" totalmente detenida, para reactivarla hay que usar la instrucción XtaskResume() para reactivarla

Aunque se mencione que FreeRTOs es usado para ejecutar varias tareas "en paralelo" esto no es del todo cierto, un procesador solo puede ejecutar una tarea simultánea en cada núcleo o hilo disponible, para conseguir que las tareas se ejecuten, lo que hace FreeRTOS es repartir el tiempo que dedica el procesador a cada una de las tareas

Por ejemplo, si tenemos que ejecutar dos tareas sin FreeRTOS, cada una de las tareas se ejecutaría secuencialmente

Ilustración 91: Ejemplo de ejecución de dos tareas sin FreeRTOS

Sin embargo, al utilizar FreeRTOS las tareas funcionan "en paralelo", por ejemplo, si tenemos dos tareas de la misma prioridad 1 llamadas A y B la ejecución seria de la siguiente manera:

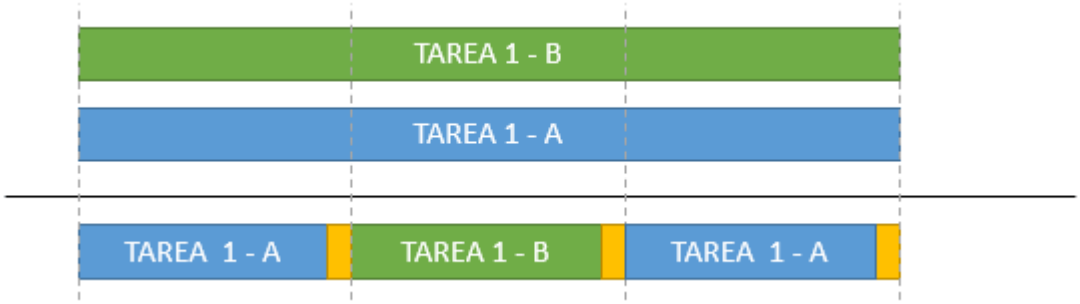


Ilustración 92: Ejemplo de ejecución de dos "tareas" con la misma prioridad usando FreeRTOS

Como se puede observar las tareas tienen la misma prioridad por lo que FreeRTOS alterna entre ambas para que puedan ejecutarse, los pequeños fragmentos de color amarillo es el tiempo que necesita el "Scheduler" para realizar tu función

En caso de que las tareas tengan diferente prioridad 1 y 2, se ejecutarían de la siguiente manera:

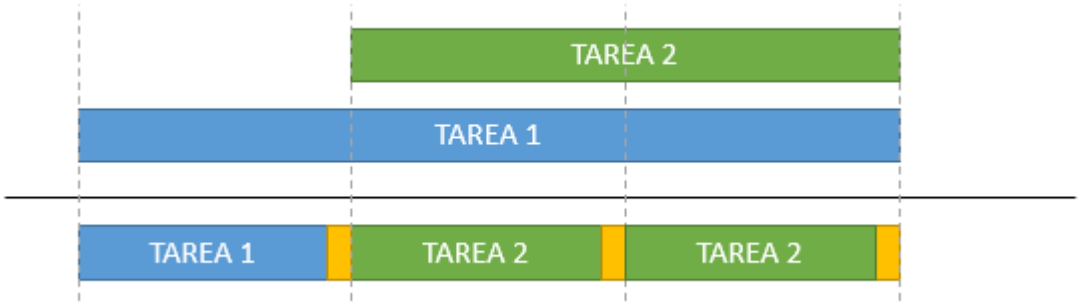


Ilustración 93: Ejemplo de ejecución de dos "tareas" con diferente prioridad en FreeRTOS

Se puede observar en el momento que la tarea de prioridad 2 está disponible pasa a su ejecución frente a la de menor prioridad, por lo tanto, tendrá que esperar hasta que las tareas de mayor prioridad pasar a estar bloqueadas o suspendidas para ejecutarse.

Material para el desarrollo de la práctica:

- Protoboard
- 5 leds de colores de 5mm
- 5 resistencias de 220ohm a ½ watt
- 1 capacitor de 100nf cerámico
- Jumpers
- Potenciómetro de 100K ohm
- Placa ESP32
- Cable micro USB

Desarrollo:

Ejercicio 1 Doble núcleo parte 1:

En las prácticas anteriores cuando cargábamos el código al ESP32 usando el Arduino IDE, por defecto el programa se ejecutaba en el núcleo 1 del microprocesador, dejando al núcleo 0 sin usar, ahora aprenderemos a cómo hacer que el ESP32 ejecute dos “programas”

a la vez, uno en cada núcleo, pero para eso primero tenemos que aprender a como diferenciarlos, para eso usaremos el siguiente código:

```
void setup() {
  Serial.begin(115200); //inicializamos el monitor serial
  Serial.print("setup() está corriendo en el núcleo  ");
  Serial.println(xPortGetCoreID()); //Imprimimos en cual
núcleo se ejecutó el "setup()"
}

void loop() {
  Serial.print("loop() está corriendo en el núcleo ");
  Serial.println(xPortGetCoreID()); //Imprimimos en cual
núcleo se ejecutó el "loop()"
  delay(5000);
}
```

La instrucción `xPortGetCoreID()` nos permite identificar en cual núcleo se está ejecutando el código si en el núcleo 0 o en el 1.

Una vez transcrito el código abrimos el monitor serial y presionamos el botón "EN" del ESP32, nos mostrara la siguiente información:

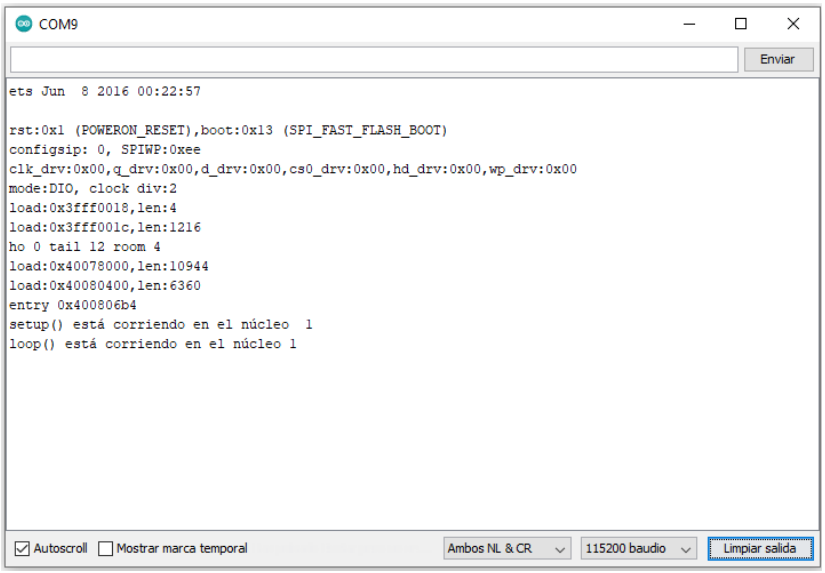


Ilustración 94: Comprobación del uso del núcleo 1 del ESP32

Como se puede observar por defecto el IDE de Arduino ejecuta todo el programa en el núcleo 1 del ESP32, para indicarle que use el núcleo 0 necesitaremos la ayuda de

FreeRTOS el cual nos permitirá crear “tareas” que es una manera sencilla de ejecutar dos “programas” en dos núcleos distintos

Ejercicio 2 Task:

Con la ayuda del siguiente código entenderemos un poco mejor a que se refiere con “tareas”, usaremos un ejemplo básico, el encendido y apagado de dos leds usando un núcleo para cada uno.

```
TaskHandle_t Task1; //creamos la tarea "Task1"
TaskHandle_t Task2; //Creamos la tarea "Task1"

// Pines a usar
const int led1 = 2; //Asignamos el puerto GPIO2 como led1
const int led2 = 4; //Asignamos al puerto GPIO4 como led2

void setup() {
  Serial.begin(115200); //Inicializamos el monitor serial
  pinMode(led1, OUTPUT); //Configuramos el pin led1 como
salida
  pinMode(led2, OUTPUT); //Configuramos el pin led2 como
salida

  //Creamos una tarea que se ejecutara en la función
Task1code() con prioridad 1 y se ejecutara en el núcleo 0
  xTaskCreatePinnedToCore(
                    Task1code,   /* Función de la tarea */
                    "Task1",     /* Nombre de la tarea */
                    10000,       /* Tamaño de pila de la
tarea */
                    NULL,        /* Parámetro de la tarea */
                    1,           /* Prioridad de la tarea */
                    &Task1,      /* Identificador de tareas
para realizar seguimiento */
                    0);          /* Asignar la tarea al
núcleo 0 */
  delay(50);

  //Creamos una tarea que se ejecutara en la función
Task2code() con prioridad 1 y se ejecutara en el nucleo 1
  xTaskCreatePinnedToCore(
                    Task2code,   /* Función de la tarea */
                    "Task2",     /* Nombre de la tarea */
                    10000,       /* Tamaño de pila de la
tarea */
```

```
                    NULL,          /* Parámetro de la tarea */
                    1,             /* Prioridad de la tarea */
                    &Task2,        /* Identificador de tareas
para realizar seguimiento */
                    1);            /* Asignar la tarea al
núcleo 1 */
    delay(50);
}

//Task1code: Parpadeo de un led cada 1000ms
void Task1code( void * pvParameters ){
  Serial.print("Task1 ejecutándose en el núcleo  ");
  Serial.println(xPortGetCoreID()); // Imprimimos en que
núcleo se está ejecutando la tarea

  for(;;){ //Este ciclo for es como el lopp(), se repetirá el
código indefinidamente
    digitalWrite(led1, HIGH); //Encendemos el LED
    delay(1000);
    digitalWrite(led1, LOW); //Apagamos el LED
    delay(1000);
  }
}

//Task2code: Parpadeo de un led cada 700ms
void Task2code( void * pvParameters ){
  Serial.print("Task2 ejecutandose en el nucleo  ");
  Serial.println(xPortGetCoreID());  //Imprimimos en que
núcleo se está ejecutando la tarea

  for(;;){ //Este ciclo for es como el lopp(), se repetirá el
código indefinidamente
    digitalWrite(led2, HIGH); //Encendemos el LED
    delay(700);
    digitalWrite(led2, LOW); //Apagamos el LED
    delay(700);
  }
}

void loop() { //Colocamos el loop para que el IDE de Arduino
no marque error.

}
```

¿Como funciona el código?

Iniciamos el código con la creación de identificadores que llamaremos Task1 y Task2

```
TaskHandle_t Task1;
TaskHandle_t Task2;
```

Asignamos los pines GPIO 2 y 4 a los leds

```
const int led1 = 2;
const int led2 = 4;
```

En el setup() inicializamos el monitor serial

```
Serial.begin(115200);
```

Asignamos los leds como salida de datos

```
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
```

Creamos la primer tarea Task 1 usando la función; xTaskCreatePinnedToCore()

```
xTaskCreatePinnedToCore(
                    Task1code,   /* Función de la tarea */
                    "Task1",     /* Nombre de la tarea */
                    10000,       /* Tamaño de pila de la
tarea */
                    NULL,        /* Parámetro de la tarea */
                    1,           /* Prioridad de la tarea */
                    &Task1,      /* Identificador de tareas
para realizar seguimiento */
                    0);          /* Asignar la tarea al
núcleo 0 */
```

La tarea estará implementada a la función Task1code(), por lo cual debemos crear la función después en el código para asignarle que es lo que ejecutara.

Usamos la misma sintaxis para crear la segunda tarea Task2

```
 xTaskCreatePinnedToCore(
                    Task2code,   /* Función de la tarea */
                    "Task2",     /* Nombre de la tarea */
                    10000,       /* Tamaño de pila de la
tarea */
                    NULL,        /* Parámetro de la tarea */
                    1,           /* Prioridad de la tarea */
```

```
                    &Task2,      /* Identificador de tareas
para realizar seguimiento */
                    1);          /* Asignar la tarea al
núcleo 1 */
```

Después creamos la función que ejecutaran las tareas, empezamos con la Task1 la cual nos mostrara en el monitor serial el mensaje que el código se ejecutó en el núcleo 0 y prendera y apagara un Led cada 1000ms

```
void Task1code( void * pvParameters ){
  Serial.print("Task1 ejecutándose en el núcleo  ");
  Serial.println(xPortGetCoreID()); // Imprimimos en que
núcleo se está ejecutando la tarea

  for(;;){ //Este ciclo for es como el lopp(), se repetirá el
código indefinidamente
    digitalWrite(led1, HIGH); //Encendemos el LED
    delay(1000);
    digitalWrite(led1, LOW); //Apagamos el LED
    delay(1000);
  }
}
```

En este caso llamamos a la función Task1code() pero puedes ponerle el nombre que quieras siempre y cuando respetes la sintaxis anteriormente configurada.

Hacemos lo mismo con la Task2 pero ahora el led prendera y apagara cada 700ms

```
void Task2code( void * pvParameters ){
  Serial.print("Task2 ejecutandose en el nucleo  ");
  Serial.println(xPortGetCoreID());  //Imprimimos en que
núcleo se está ejecutando la tarea

  for(;;){ //Este ciclo for es como el lopp(), se repetirá el
código indefinidamente
    digitalWrite(led2, HIGH); //Encendemos el LED
    delay(700);
    digitalWrite(led2, LOW); //Apagamos el LED
    delay(700);
  }
}
```

Por ultimo incluimos la función loop() vacía, esto es porque si no se coloca nos mandara un error de compilación.

```
void loop() { //Colocamos el loop para que el IDE de Arduino
no marque error.

}
```

Nota: Como comprobamos anteriormente por defecto el IDE de Arduino ejecuta la función loop() en el núcleo 1, para futuros ejercicios se puede omitir la creación de la tarea Task2 y escribir las instrucciones directamente en el loop() esto nos ayuda a simplificar mucho nuestro código.

Antes de cargar el programa al ESP32 armaremos el siguiente diagrama:

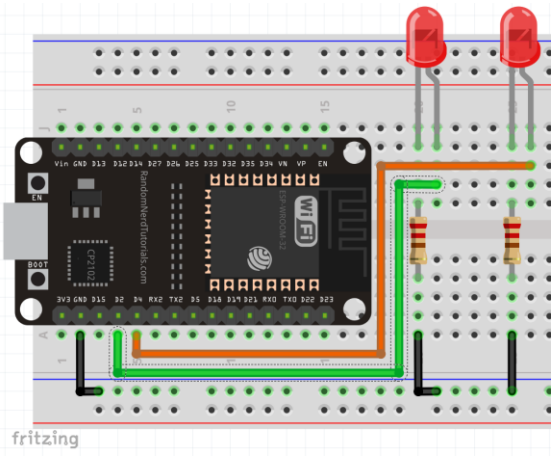


Ilustración 95: Diagrama del Ejercicio 2

Una vez armado el circuito procedemos a cargarlo al ESP32, después abrimos el monitor serial y presionamos el botón “EN” del ESP32, nos deberá mostrar la siguiente información:

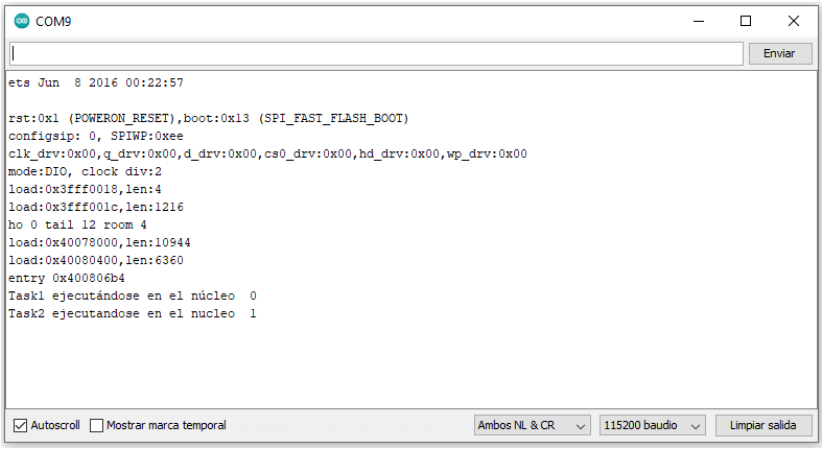


Ilustración 96: Comprobación del uso del núcleo 0 y 1 del ESP32

Como se puede observar las tareas se están ejecutando en diferentes núcleos y los leds empezaron a parpadear cada uno a diferente tiempo, este tipo de programación nos ayuda a que podamos "hacer dos cosas a la vez" es decir, ejecutar dos códigos diferentes al mismo tiempo

Ejercicio 3 Doble núcleo parte 2:

Usando este tipo de programación haremos una pequeña mezcla de dos practicas anteriores, haremos que el ESP32 muestre una cuenta binaria con leds y al mismo tiempo este usando el convertidor analógico digital.

Para ello usaremos el siguiente código:

```
TaskHandle_t Task1; //creamos la tarea "Task1"

int outPin[] = {19, 21, 22, 23}; //Configuramos un ARRAY de
los pines a usar
const int potPin = 34; // Configuramos el pin 34 que será
donde conectaremos el Potenciómetro
int potValor = 0; // Variable donde guardaremos el valor del
Potenciómetro
void setup() {
  Serial.begin(115200); //Inicializamos el monitor serial
  //Utilizamos un arreglo con el ciclo FOR para definir todos
los pines de ARRAY como salidas
  int i = 0;
  for ( i = 0; i < 4; i++)
    pinMode(outPin[i], OUTPUT);
```

```
  xTaskCreatePinnedToCore(
    Task1code,   /* Función de la tarea */
    "Task1",     /* Nombre de la tarea */
    10000,       /* Tamaño de pila de la tarea */
    NULL,        /* Parámetro de la tarea */
    1,           /* Prioridad de la tarea */
    &Task1,      /* Identificador de tareas para realizar
seguimiento */
    0);          /* Asignar la tarea al núcleo 0 */
  delay(50);
}

void Task1code( void * pvParameters ) {
  for (;;) { //Este ciclo for es como el lopp(), se repetirá
el código indefinidamente
    //Para hacer la cuenta ascendente utilizaremos la ayuda
de dos ciclos FOR
    int i = 0, j = 0; //Primero definimos las variables para
usar los contadores
    for ( i = 0; i < 16; i++) //Este ciclo FOR nos servirá
para poner un límite al contador, al ser un contador de 4
bits el límite seria 16
    {
      for ( j = 0; j < 4; j++)//Este ciclo FOR nos servirá
para indicar que pin del ARRAY estará en estado ALTO o BAJO
      {
        if ( ( (i >> j) & 1 )  == 1 )//Esta condición es la
que se encargara de saber cuál led cambiara o no su estado
          digitalWrite(outPin[j], HIGH);
        else
          digitalWrite(outPin[j], LOW);
      }
      delay(1000); //Un pequeño retraso para que la cuenta
binaria cambie cada 1000ms
    }
  }
}
void loop() {

  potValor = analogRead(potPin); // Leemos el valor del
potenciómetro y lo guardamos en potValor
  Serial.println(potValor); //Imprimimos el valor leído en el
monitor serial.
  delay(500);

}
```

Antes de cargar el programa al ESP32 armaremos el siguiente circuito:

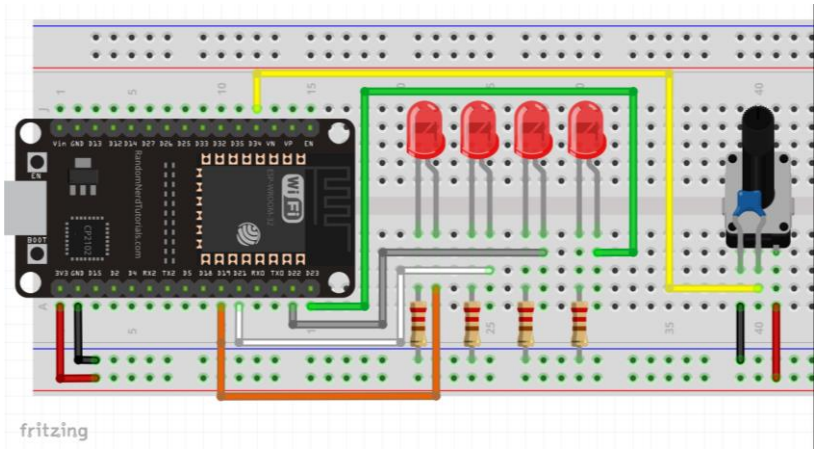


Ilustración 97: Diagrama del Ejercicio 3

Una vez echo, procederemos a transcribir el código al IDE de Arduino, lo cargaremos al ESP32 y abrimos el monitor serial, nos deberá mostrar la siguiente información del ADC

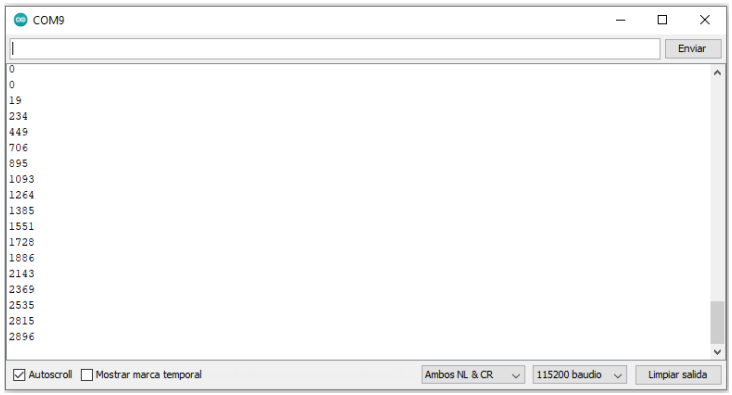


Ilustración 98: Comprobación del uso de ADC por el núcleo 1 del ESP32

Como se puede observar el ESP32 está usando el ADC y al mismo tiempo se encuentra haciendo una cuenta binaria con los leds, con esto comprobamos la utilidad de usar el doble núcleo del ESP32.

Trabajo de casa 6 de la practica 5: Sensores humedad, temperatura, ultrasonido y barométrico con el ESP32

1. Mencione los tipos de sensores que hay en el mercado
2. ¿Qué es un sensor de medida relativa?
3. ¿Un sensor es lo mismo que un transductor? ¿Por qué?
4. Describa las diferencias entre sensores con salida lineal y digital
5. Mencione las características de un sensor
6. ¿Cuál es la diferencia entre medida relativa y absoluta?
7. Aparte del sensor DHT22, mencione otros sensores de temperatura
8. ¿Como se puede medir la velocidad del sonido?
9. ¿Cuáles son las unidades de medición de la presión?
10. ¿En qué aplicaciones encontramos el sensor BMP280?

Practica 5: Sensores humedad, temperatura, ultrasónico y barométrico con el ESP32

Objetivos:

- Que el alumno conecte diferentes sensores con el ESP32
- Conectar sensores del rango de 3.3v al ESP32
- Como conectar sensores de 5v al ESP32 sin dañarlo
- Usar y configurar sensores básicos en el ESP32
- Conocer diferentes sensores que hay en el mercado

Introducción:

Sensor de humedad y temperatura DHT22

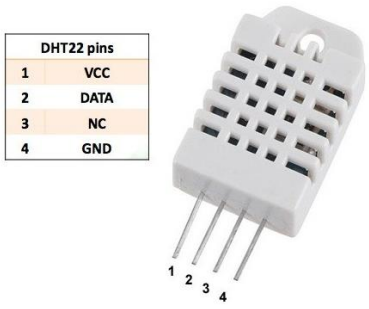


Ilustración 99: Pin-out sensor DHT22

El DHT22 también llamado sensor AM2302 es un sensor digital de humedad y temperatura relativa de bajo costo, integra en su interior un sensor capacitivo de humedad y un termistor para medir el aire circundante, los datos los envía atraves de un pin de datos, su puede usar en aplicaciones de control de temperatura o de monitoreo ambiental.[60]

Es muy usado en plataformas como Arduino y Raspberry PI, es muy barato y pequeño lo que lo hace fácil de manipular, solo posee 3 pines VCC, GND y DATA. Los coeficientes de calibración vienen grabados en una memoria OPT la cual ya vienen puestos de fábrica, su única desventaja es que el tiempo de actualización es de cada 2 segundos, es decir, cada 2 segundos mandara la información de temperatura y humedad.[61]

Tiene las siguientes características técnicas:[62]

- Voltaje de alimentación: 3.3 a 5.5v DC
- interface digital: Single bus (bidireccional)
- Rango de temperatura: -40 a 80° Celsius aproximadamente
- Rango de humedad: 0 a 100%RH aproximadamente
- Precisión de temperatura: +- 0.5° Celsius
- Precisión de humedad: +-2%RH
- Sensibilidad de temperatura: 0.1° Celsius
- Sensibilidad de Humedad 0.1%RH
- Tiempo de censado: 2 Segundos
- Consumo de corriente: 1 a 1.5 mA

Sensor ultrasónico HC-SR04

Ilustración 100: Sensor ultrasónico HC-SR04

Este sensor ultrasónico es capaz de medir la distancia a la que se encuentra un objeto con un rango de hasta 450 cm, con una precisión de +/- 0.5 cm, presentando un consumo máximo de 8mA.

Su funcionamiento es el siguiente; se emite un tren de pulsos de ondas mecánicas (TRIG), luego se captura el tren de pulsos producido por el rebote o eco de las ondas (ECHO), el cual calculara la distancia a partir de las diferencias de tiempo entre el TRIG y el ECHO en base a la velocidad de estas ondas en el aire.

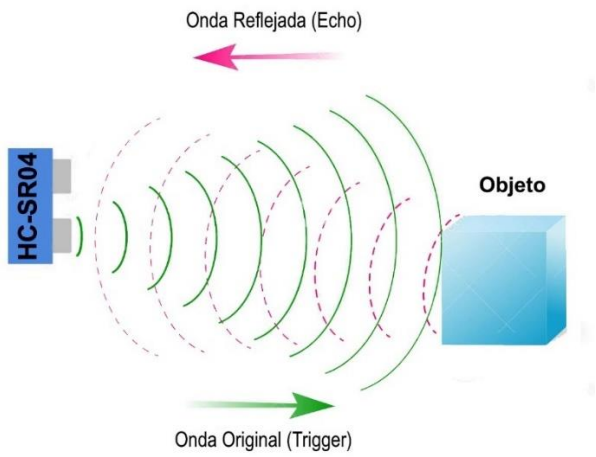


Ilustración 101: Funcionamiento del sensor ultrasónico HC-SR04

Tiene solo 4 terminales de conexión.

- 5V el cual es el voltaje positivo
- Trigger: pin que se encarga de mandar el tren de pulsos
- Echo: pin que se encarga de mandar el rebote de los pulsos antes enviado
- GND: pin de tierra

Especificaciones técnicas:

- Voltaje de alimentación: 5V DC
- Corriente de consumo: 15mA
- Distancia máxima: 4m
- Distancia mínima: 2cm
- Margen de error: 3mm o 0.3cm
- Tiempo de espera entre mediciones: 20ms

Sensor de presión barométrica BMP280

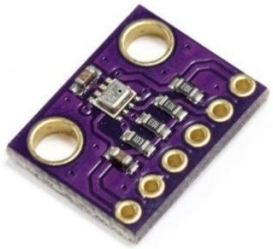


Ilustración 102: Sensor BMP280

El sensor de presión barométrica BMP280 es la evolución del BMP180 mide la altura con respecto al nivel del mar, su funcionamiento está basado en la relación que hay entre la presión y la altitud, conociendo estos valores, el sensor puede leerlo y dar una interpretación de la presión atmosférica medida en hPa, utiliza la tecnología BOSCH piezo-resistiva con gran robustez EMC alta precisión y linealidad, se puede comunicar con los protocolos 12C o SPI.[63]

La distribución de pines en la siguiente:

Ilustración 103: Pinout del sensor BMP280

- VCC: 3.3v DC
- GND: Conexión a tierra
- SCL: Conexión para I2C
- SDA: Conexión para I2C
- CSB: Conexión para ISP
- SDO: Conexión para ISP

Características técnicas:

- Voltaje de alimentación: 1.8 a 3.3v DC
- Interfaz de comunicación: I2C o SPI a 3.3v
- Rango de presión: 300 a 1100hPa
- Resolución: 0.16 Pa
- Precisión absoluta: 1hPa
- Resolución de temperatura: 0.01° Celsius
- Precisión de temperatura: 1° Celsius
- Calibrado desde fabrica
- Posee un sensor de temperatura

Material necesario para el desarrollo de la práctica:

- Placa ESP32
- Sensor de temperatura y humedad DHT22
- Sensor ultrasónico HC-SR04
- Sensor barométrico BMP280
- Protoboard
- Jumpers
- Cable micro USB
- 1 Resistencia de 10k Ohm a ½ watt
- 2 resistencias de 1 K Ohm a ½ watt

Desarrollo:

Ejercicio 1 Sensor DHT22:

Primero tenemos que descargar las librerías del sensor junto con una librería de sensores para eso nos dirigimos a la pestaña de "programa" después en "incluir librería" y seleccionamos "Administrar Bibliotecas".[64]

Nota: Para estos pasos es necesario tener una conexión a internet estable.

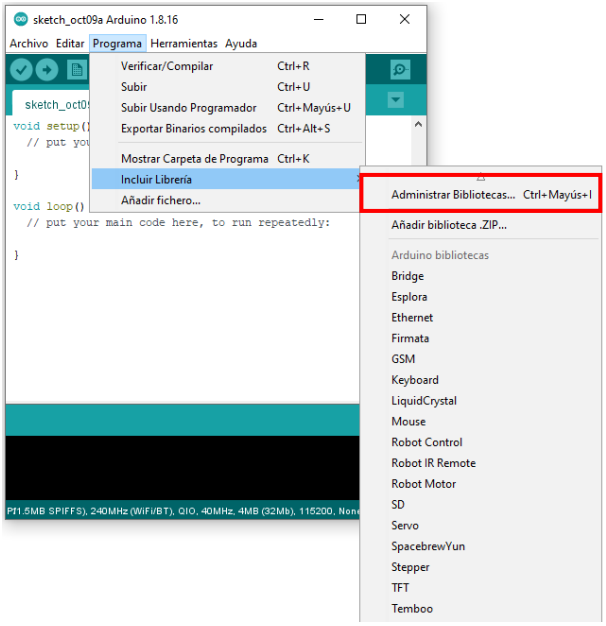


Ilustración 104: Menú para instalar librerías en el IDE de Arduino

Se abrirá una nueva ventana como esta:

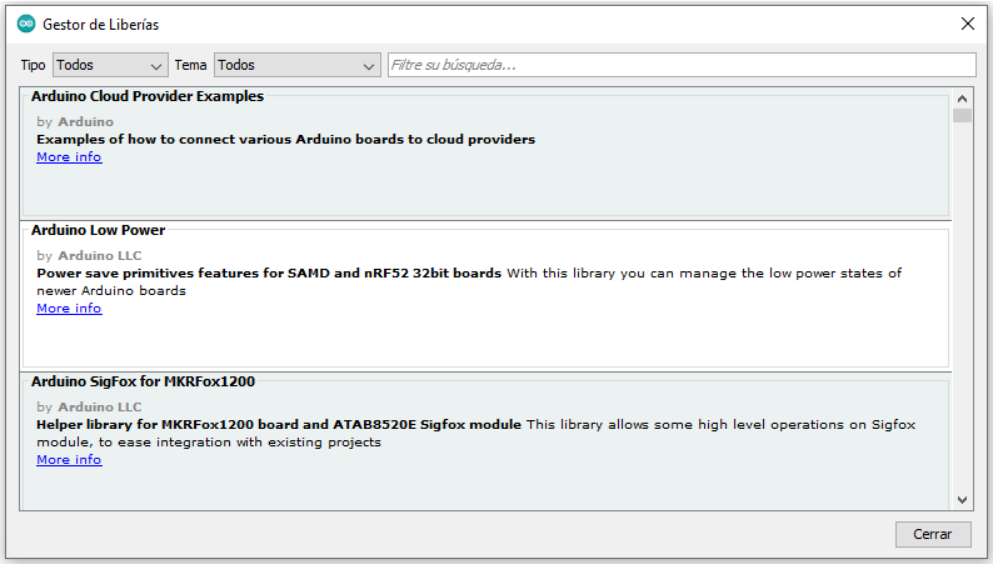


Ilustración 105: Ventana de gestor de librerías

En el cuadro de búsqueda ponemos “DHT”, presionamos Enter, y seleccionamos “instalar” en la opción marcada en el cuadro rojo:

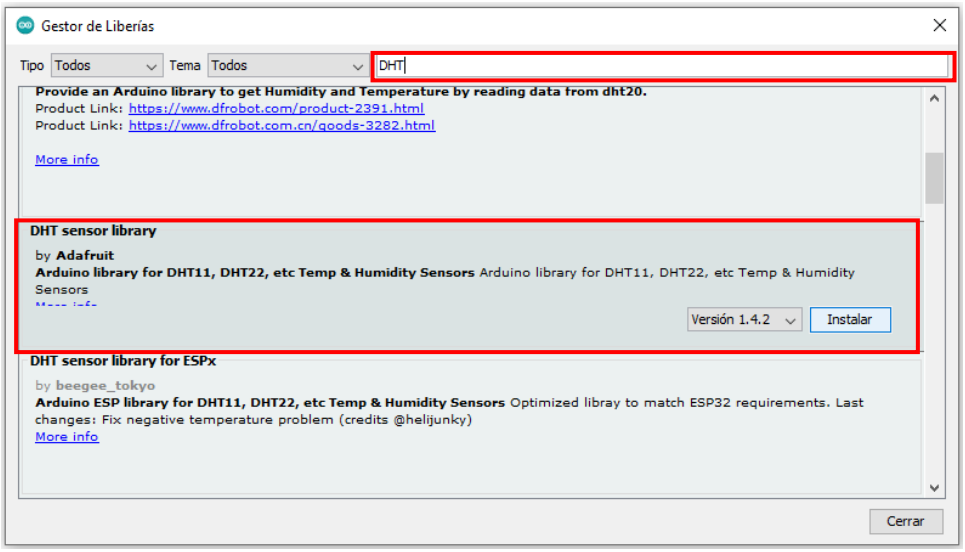


Ilustración 106: Instalación de la librería del sensor DHT22

Nos Pedirá que instalemos otra librería, le damos en “install all” para instalarla también.

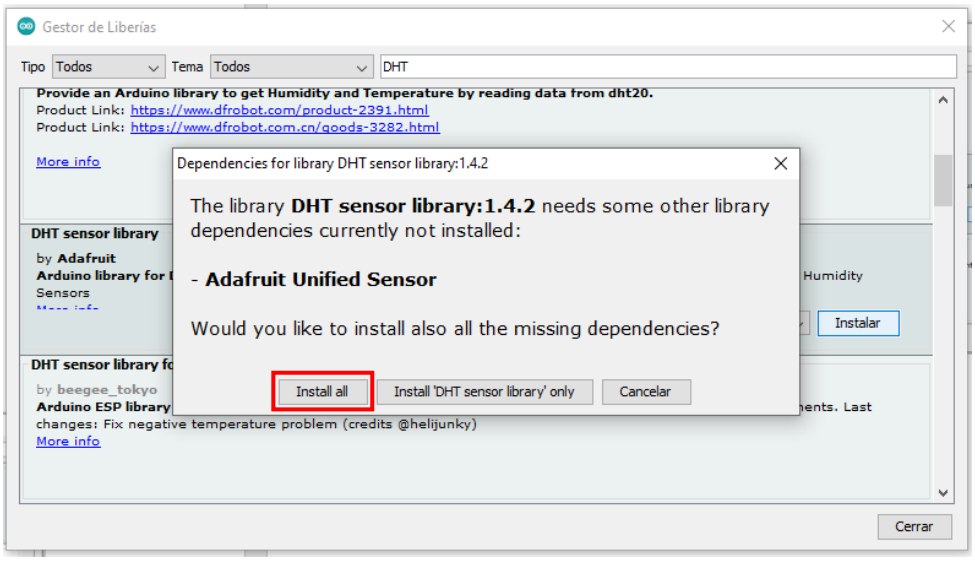


Ilustración 107: Instalación de la librería complementaria Adafruit Unified Sensor

Esperamos un poco y cuando ya se hayan instalado reiniciamos el IDE de Arduino

Una vez reiniciado transcribimos el siguiente código:

```
#include "DHT.h" //Incluimos la librería del sensor DHT
#define DHTPIN 4     // Pin para conectarlo al sensor DHT
#define DHTTYPE DHT22   // Modelo del sensor

DHT dht(DHTPIN, DHTTYPE); //Inicializamos el DHT sensor

void setup() {
  Serial.begin(115200); //Inicializamos el monitor serial
  Serial.println(F("DHTxx testeado!"));
  dht.begin(); //Inicializamos el sensor DHT
}

void loop() {
  // Esperamos unos segundos entre cada lectura
  delay(2000);
  // Lectura de la humedad y la guardamos en h
  float h = dht.readHumidity();
  // Lectura de temperatura y lo guardamos en t
  float t = dht.readTemperature();
  // Pequeño chequeo de si las lecturas anteriores se
hicieron correctamente, si no, regresa a hacerlas de nuevo
  if (isnan(h) || isnan(t)) {
    Serial.println(F("Error! Lectura del sensor
incorrecta"));
    return;
```

```
  }

  //Imprimimos las lecturas del sensor
  Serial.print(F("Humedad: "));
  Serial.print(h);
  Serial.print(F("%  Temperatura: "));
  Serial.print(t);
  Serial.println(F("°C "));

}
```

¿Como funciona el código?

Primero incluimos la librería del sensor DHT

```
#include "DHT.h"
```

Después definimos a que pin lo conectaremos, en este caso al pin GPIO4 del ESP32, pero puedes asignarle en que tú quieras

```
#define DHTPIN 4
```

Ahora seleccionamos el modelo del sensor que vamos a usar, en este caso el DHT22

```
#define DHTTYPE DHT22
```

Creamos al objeto llamado dht para asignarle las configuraciones anteriores

```
DHT dht(DHTPIN, DHTTYPE);
```

En el setup() inicializamos el monitor serial e imprimimos un mensaje de testeo

```
Serial.begin(115200);
Serial.println(F("DHTxx testeado!"));
```

Inicializamos el sensor DHT

```
dht.begin();
```

En el loop() iniciamos con un delay de 2000ms esto porque al sensor DHT le toma ese tiempo en tener lista la lectura

```
delay(2000);
```

Usamos la instrucción readHumidity() para leer la humedad y la guardamos en la variable h como flotante, es decir con punto decimal

```
float h = dht.readHumidity();
```

Usamos la instrucción readTemperature() para leer la temperatura y la guardamos en t igual como coma flotante

```
float t = dht.readTemperature();
```

Hacemos un pequeño chequeo para ver si las temperaturas fueron leídas correctamente, si no fue así, el programa mostrará un mensaje de error, regresará al inicio y las volverá a leer otra vez

```
if (isnan(h) || isnan(t)) {
    Serial.println(F("Error! Lectura del sensor
incorrecta"));
    return;
}
```

Finalmente, imprimimos todas las variables en el monitor serial

```
Serial.print(F("Humedad: "));
Serial.print(h);
Serial.print(F("%  Temperatura: "));
Serial.print(t);
Serial.println(F("°C "));
```

Antes de cargar el programa al ESP32 primero armaremos el siguiente circuito:

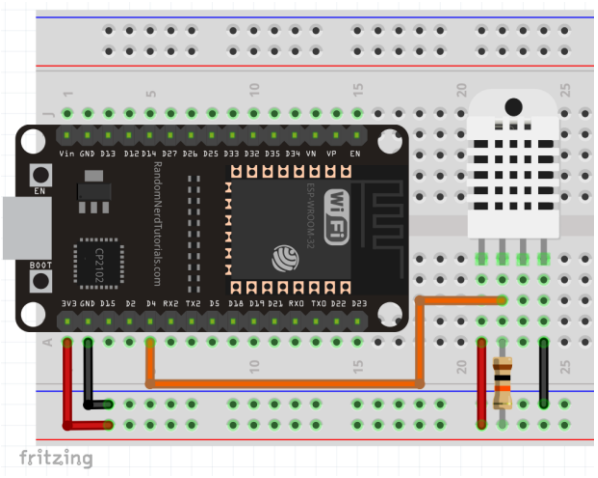


Ilustración 108: Diagrama Ejercicio 1

Una vez cargado el programa, presionamos el botón “EN” del ESP32, abrimos el monitor serial y nos deberá aparecer el valor de la humedad y la temperatura cada dos segundos.

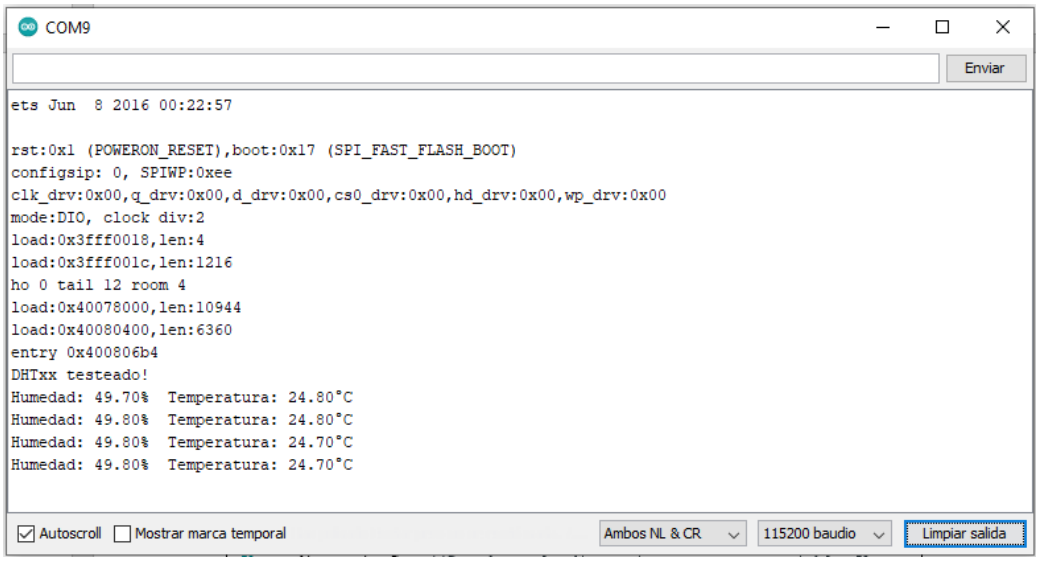


Ilustración 109: Comprobación de funcionamiento del sensor DHT22

Nota: Si presenta el mensaje de error, revise que las conexiones estén bien echas y también revisar si las librerías se instalaron correctamente

Ejercicio 2 Sensor ultrasónico HC-SR04:

Para el uso del sensor ultrasónico usaremos el siguiente código:[65]

```
//Definimos los pines a usar para el sensor
const int trigPin = 5; //Pin para el trig
const int echoPin = 18; //Pin para el echo

//Definimos la velocidad del sonido en cm/uS
#define Velocidad_Sonido 0.034

long duracion; //Guardamos la duración del rebote del sonido
float distancia_Cm; //Guardamos la distancia en cm como
variable float

void setup() {
  Serial.begin(115200); // Inicializamos el monitor serial
  pinMode(trigPin, OUTPUT); // configuramos trigPin como
salida
```

```
  pinMode(echoPin, INPUT); // configuramos echoPin como
entrada
}

void loop() {
  // mandamos pulso bajo al trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Mandamos estado alto al pin trigPin por 10 microsegundos
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Leemos el echoPin
  duracion = pulseIn(echoPin, HIGH);
  // Calculamos la distancia
  distancia_Cm = duracion * Velocidad_Sonido/2;
  // Imprimimos la distancia en centímetros
  Serial.print("Distancia (cm): ");
  Serial.println(distancia_Cm);
  delay(1000);
}
```

¿Como funciona el código?

Primero configuramos los pines que usaremos para conectar el sensor al ESP32

```
const int trigPin = 5; //Pin para el trig
const int echoPin = 18; //Pin para el echo
```

Necesitaremos la constante de velocidad del sonido en el aire para hacer el cálculo de la distancia, el valor aproximado es de 0.034 cm/uS

```
#define Velocidad_Sonido 0.034
```

Después creamos variables donde se guardarán las lecturas medidas por el sensor

```
long duracion; //Guardamos la duración del rebote del sonido
float distancia_Cm; //Guardamos la distancia en cm como
variable float
```

En el setup() primero inicializamos el monitor serial

```
Serial.begin(115200); // Inicializamos el monitor serial
```

Después configuramos los pines trigPin y echoPin como salida y entrada respectivamente

```
pinMode(trigPin, OUTPUT); // configuramos trigPin como salida
pinMode(echoPin, INPUT); // configuramos echoPin como entrada
```

En el setup() primero mandamos un pulso bajo al trigPin esto para limpiar la señal, luego mandamos un pulso alto pero solo por 10 microsegundos y luego lo ponemos en bajo

```
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Mandamos estado alto al pin trigPin por 10 microsegundos
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
```

Después usamos la instrucción pulseIn() para tener el intervalo de tiempo de la onda y la guardamos en la variable en “duración”

```
duracion = pulseIn(echoPin, HIGH);
```

Con esa información ahora calculamos la duración multiplicando ese valor por la velocidad del sonido y dividirlo entro 2

```
distancia_Cm = duracion * Velocidad_Sonido/2;
```

Finalmente imprimimos el valor calculado el cual será la distancia en cm

```
Serial.print("Distancia (cm): ");
Serial.println(distancia_Cm);
```

Antes de cargar el programa al ESP32 armaremos el siguiente circuito:

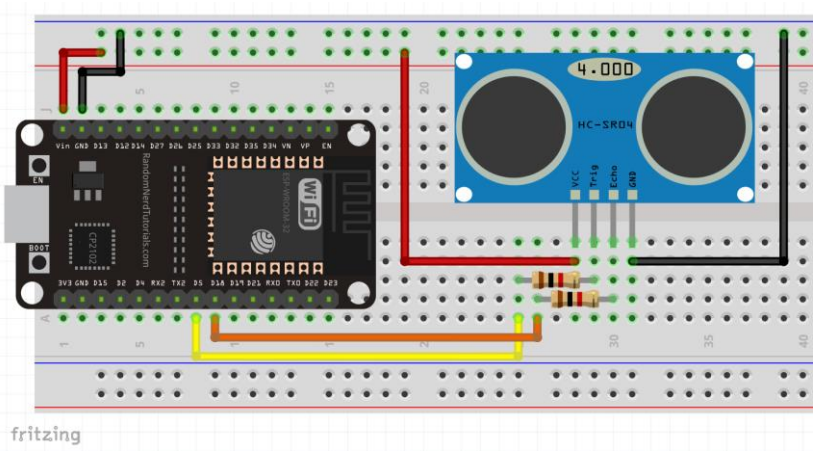


Ilustración 110: Diagrama del Ejercicio 2

Nota: Usaremos resistencias entre la comunicación del sensor y el ESP32 para evitar dañarlo, como el sensor trabaja con 5v y el ESP32 su voltaje máximo de entrada es de 3.3v corremos el riesgo de dañar al ESP32

Una vez transcrito el código lo cargamos al ESP32 ponemos un objeto enfrente al sensor ultrasónico y abrimos el monitor serial, nos debe mostrar la siguiente información

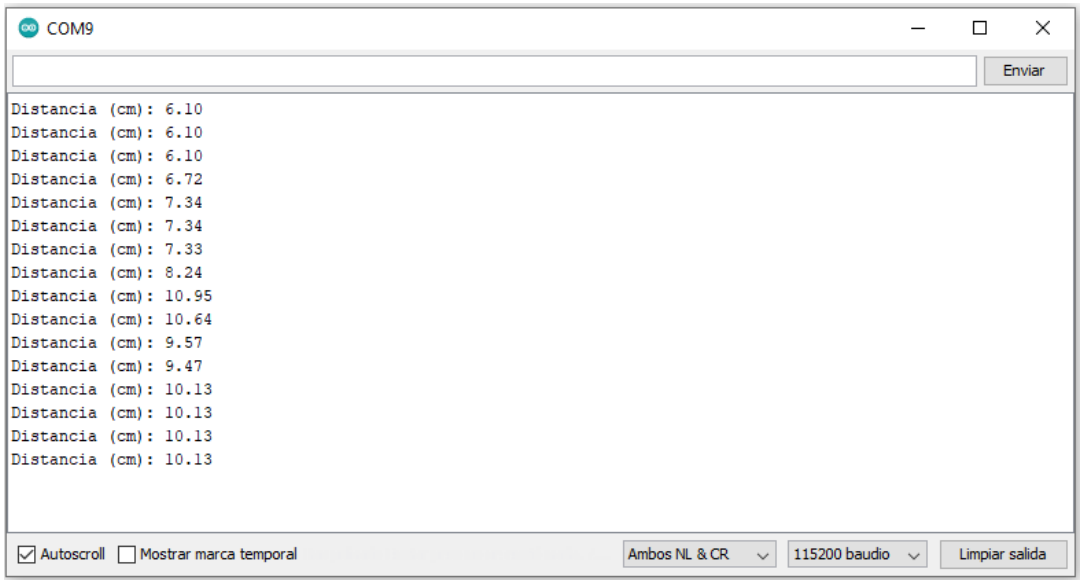


Ilustración 111: Comprobación del sensor ultrasónico HC-SR04

Al mover el objeto el valor de la distancia deberá de cambiar.

Ejercicio 3 sensor barométrico BMP280:

Primero tenemos que descargar las librerías del sensor junto con una librería de sensores para eso nos dirigimos a la pestaña de "programa" después en "incluir librería" y seleccionamos "administrar bibliotecas"

Nota: Para estos pasos es necesario tener una conexión a internet estable.

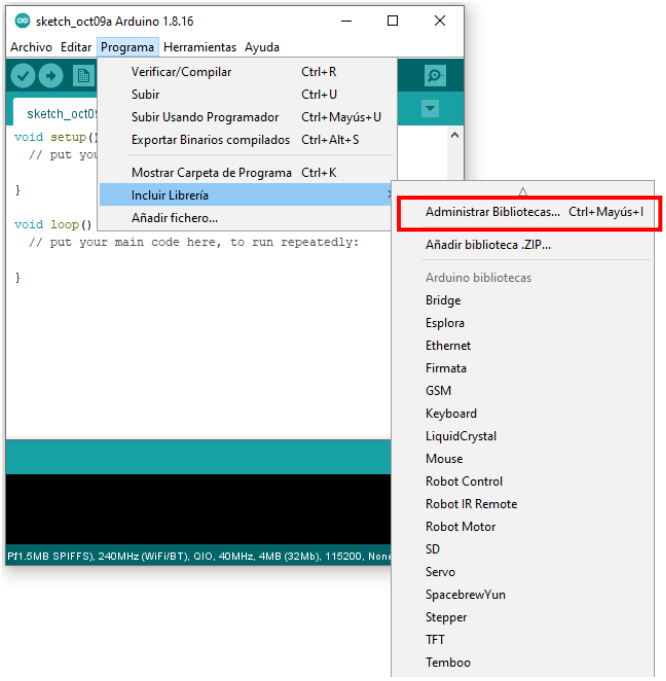


Ilustración 112: Menú de instalación de librerías

Nos abrirá una nueva ventana como esta:

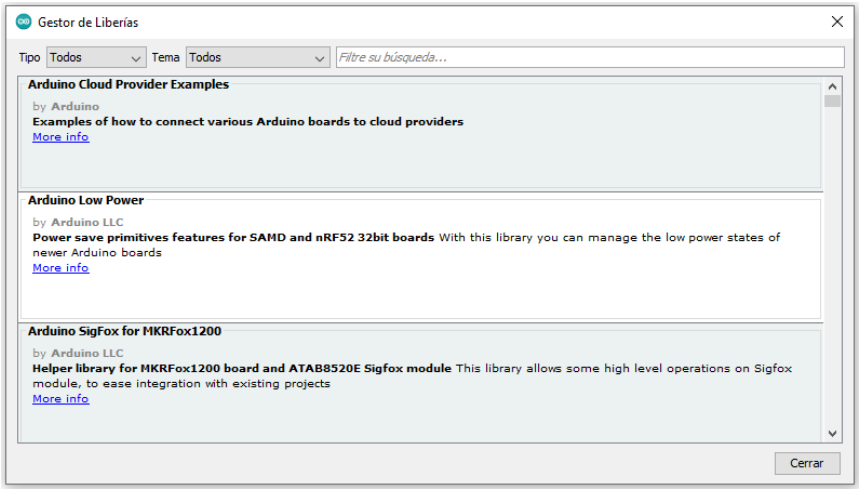


Ilustración 113: Ventana del Gestor de Librerías

En el cuadro de búsqueda ponemos “adafruit bmp280” presionamos Enter y seleccionamos la opción marcada en rojo.

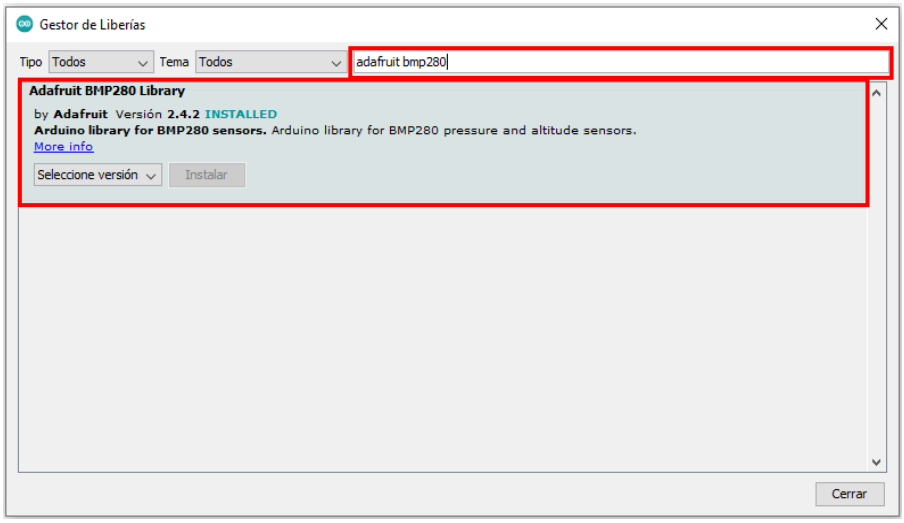


Ilustración 114: Instalación de la librería "adafruit bmp280"

Para el uso del sensor BMP280 usaremos el siguiente código:

Nota: Asegúrese que el sensor sea el BMP280 y no el BME280 de lo contrario este código no funcionara.

```
//Incluiamos las librerias para el sesnor BMP280 conectado
con I2C del ESP32
#include <Wire.h>
#include <Adafruit_BMP280.h>
//Esta varaibale es la precion al nivel del mar en
hectopascales
#define SEALEVELPRESSURE_HPA (1013.2)
//Creamos el objeto llamado bmp
Adafruit_BMP280 bmp;

void setup() {
  Serial.begin(115200); //Iniciamos el monitor serial
  Serial.println(F("Testeo del sensor BMP280"));
  //checamos si el sensor se incio y comunico correctamente
  if (!bmp.begin(0x76,0x58)) {
    Serial.println(F("No se pudo encontrar el sensor BMP280,
revise la conexion!"));
    while (1) delay(10);
```

```
  }
  //Configuraciones por defecto del sensor
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Modo
de operaciom */
                  Adafruit_BMP280::SAMPLING_X2,     /*
Sobremuestreo de temperatura */
                  Adafruit_BMP280::SAMPLING_X16,    /*
Sobremuestreo de presion */
                  Adafruit_BMP280::FILTER_X16,      /*
Filtracion */
                  Adafruit_BMP280::STANDBY_MS_500); /* Modo
de espera */
}

void loop() {
  //Leemos e imprimimos las variables del sensor
  Serial.print("Temperatura = ");
  Serial.print(bmp.readTemperature());
  Serial.println(" *C");
  Serial.print("Presion = ");
  Serial.print(bmp.readPressure() / 100.0F);
  Serial.println(" hPa");
  Serial.print("Altitud aproximada = ");
  Serial.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");
  Serial.println();
  delay(1000);
}
```

¿Como funciona el código?

Primero incluimos las librerías para una comunicación I2C con el ESP32 y las librerías del sensor BME280

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
```

Usamos una variable para la presión a nivel del mar en hectopascal medida de la CDMX

```
#define SEALEVELPRESSURE_HPA (1013.2)
```

Como usaremos la conexión I2C solo necesitamos usar la siguiente instrucción
`Adafruit_BMP280 bmp;`

En el setup() iniciamos con el monitor serial

```
Serial.begin(115200);
```

Después inicializamos el sensor y revisamos si está bien conectado al ESP32

```
 if (!bmp.begin(0x76,0x58)) {
    Serial.println(F("No se pudo encontrar el sensor BMP280,
revise la conexion!"));
    while (1) delay(10);
  }
```

En el loop() leemos los valores con las siguientes instrucciones y las imprimimos

- bmp.readTemperature()- Lectura de temperatura en Celsius
- bmp.readPressure() - lectura de la presión en hPa (hectoPascal = millibar)
- bmp.readAltitude(SEALEVELPRESSURE_HPA) - Estimación de la altitud en metros basados en la presión a nivel del mar

Antes de cargar el programa al ESP32 armaremos el siguiente circuito:

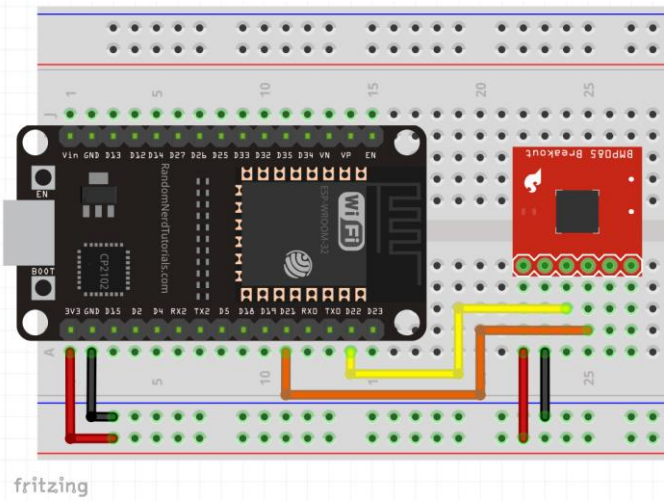


Ilustración 115: Diagrama del Ejercicio 3

Después de armar el circuito, cargaremos el programa al ESP32, abrimos el monitor serial y presionamos el botón “EN” del ESP32, nos deberá mostrar las lecturas como en la siguiente imagen:

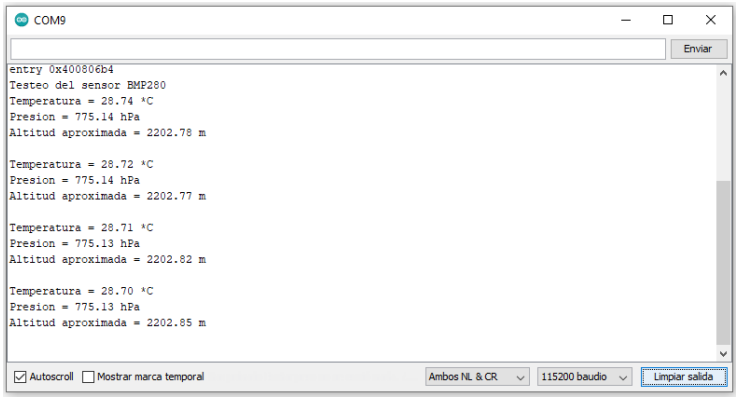


Ilustración 116: Comprobación del sensor de presión barométrica BMP280

Nota: Si presenta el mensaje de "¡No se pudo encontrar el sensor BMP280, revise la conexión!", verifique que el módulo sea el BMP280, si conecta el BME280 este código no funcionara.

Trabajo de casa 7 de la practica 6: Comunicación bidireccional del ESP32 a un smartphone con Android usando Bluetooth

1. ¿Qué es domótica?
2. Describa que es el IoT (Internet de las cosas)
3. ¿Cuáles son las versiones del Bluetooth?
4. Mencione las topologías de redes del Bluetooth
5. ¿Cuáles son las otras maneras de transmitir información inalámbricamente de corto alcance?
6. Mencione dos aplicaciones del uso del Bluetooth en la domótica
7. Aparte de su uso con Arduino, mencione otras plataformas o microcontroladores en los que se puede usar el Bluetooth

Practica 6: Comunicación bidireccional del ESP32 a un smartphone con Android usando Bluetooth

Objetivos:

- Que el alumno conozca una forma de enviar datos de sensores a un smartphone
- Hacer que el alumno aprenda a controlar los puertos del ESP32 con comandos enviados por Bluetooth
- Usar el doble núcleo del ESP32 para enviar datos de sensores y recibir comandos de un smartphone con Android

Introducción:

Desde un inicio el Bluetooth fue creado con el propósito de eliminar los cables que se usaban para transferir información de un teléfono móvil a una computadora por ejemplo, la principal competencia de esa tecnología fue el IrDA el cual básicamente era la transmisión de datos vía infrarrojo, que aunque en sus inicios era muy usada, al final se volvió relativamente obsoleta, con la aparición de documentos más pesados el tiempo de transmisión del IrDA se volvían cada vez más largos, tenían que tener los dos dispositivos pegados literalmente y cualquier movimiento podía dañar el archivo que se estaba transfiriendo.

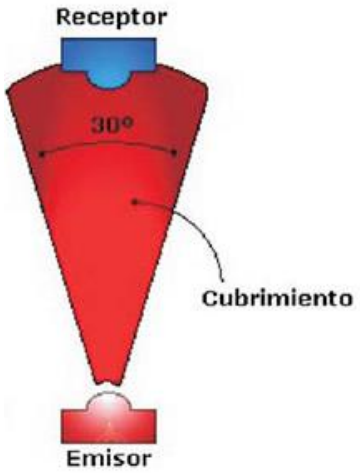


Ilustración 117: Funcionamiento del IrDA

Con el Bluetooth las cosas eran más sencillas ya que la transferencia de datos podía hacerse sin problemas y relativamente rápido, el único requisito era que el dispositivo no saliera de un rango mayos a 5 o 10 metros.

Al tratarse de una forma de transferir archivos sin usar cables y a una distancia considerable, fue muy usado por teléfonos móviles y laptops, pero la desventaja fue otra vez el tamaño de los archivos, por lo que el Bluetooth se limitaba a compartir archivos de bajo peso, como imágenes, documentos de texto, juegos sencillos compilados en java, y videos cortos de baja calidad.

También cuando se creó el Bluetooth su finalidad era conectar periféricos como manos libres o audífonos para transmitir música o audio sin la necesidad de cables, pero la calidad era muy inferior a unos tradicionales audífonos.

En años recientes, con la llegada de los microcontroladores PIC se pudo observar otra función para el Bluetooth, su uso para transferir datos de sensores a un dispositivo móvil como un celular o una PC, esto ayudando a la recolección de datos para su uso en casas u oficinas.

Con la placa de desarrollo de Arduino también el Bluetooth se volvió una de las principales razones de su utilización en la automatización o en la domótica, su fácil programación de estas placas y con la ayuda de los smartphones con sistema operativo Android, se volvió muy sencillo desarrollar proyectos con la utilización del Bluetooth, como lo son el control de luces o monitorización de sensores.

Ilustración 118: Logos de Arduino y Bluetooth

Aunque hoy en día el Bluetooth se sigue usando para la transmisión de audio sin cables y en él envió de información, su uso en la automatización o domótica se volvió vital su estudio y aplicación para diversos proyectos.

Material para el desarrollo de la práctica:

- Sensor de temperatura y humedad DHT22
- 2 resistencias de 220 ohm ½ watt
- 1 Resistencia de 10k Ohm a ½ watt
- Placa ESP32
- Protoboard
- Jumpers
- Cable micro-USB
- Celular con Android 6.0 o superior

Desarrollo:

En la practicas anteriores aprendimos a como realizar una conexión entre un smartphone con Android y el ESP32, pero solo nos enfocamos a mandar texto, ahora aprenderemos a cómo utilizar comandos para controlar los puertos GPIO del ESP32 y como mandar datos de un sensor cada cierto tiempo.

Ejercicio 1 Control de puertos GPIO con Bluetooth:

Para controlar los puertos GPIO del ESP32 usaremos el siguiente código:

```
#include "BluetoothSerial.h" //Incluimos la libleria del
Bluetooth Serial
//Checamos si el Bluetooth se habilito correctamente
#if !defined(CONFIG_BT_ENABLED) ||
!defined(CONFIG_BLUEDROID_ENABLED)
#error ¡Bluetooth no está habilitado! Ejecute `make
menuconfig` y habilítelo
#endif
#define LED 23 //Definimos el pin del led como el pin 23 del
ESP32

BluetoothSerial SerialBT; //llamamos al Bluetooth Serial como
"Serial BT"

//Variables en donde guardamos los datos recibidos
String message = "";
char incomingChar;
//Creamos la función callback
```

```
void callback_function(esp_spp_cb_event_t event,
esp_spp_cb_param_t *param) {
  if (event == ESP_SPP_START_EVT) {
    Serial.println("Inicializado SPP"); //Mensaje que el
Bluetooth se inició correctamente
  }
  else if (event == ESP_SPP_SRV_OPEN_EVT ) {
    Serial.println("Cliente conectado"); //Mensaje que un
cliente se conectó correctamente
  }
  else if (event == ESP_SPP_CLOSE_EVT  ) {
    Serial.println("Cliente desconectado"); //Mensaje que el
cliente se desconecto
  }
  else if (event == ESP_SPP_DATA_IND_EVT ) {
    //Imrimimos si hay datos recibidos
    Serial.println("Datos recibidos");

    while (SerialBT.available()) { // Mientras haya datos por
recibir
      char incoming = SerialBT.read(); // Lee los datos
recibidos
      if (incoming != '\n'){
      message += String(incoming);
    }
    //En caso de no haber datos limpiamos la variable message
    else{
      message = "";
    }
    //Imrpimimos el mensaje recibido
      Serial.print("Recibido: ");
      Serial.println(incoming);
      //Si el mensaje recibido fue "led_on" mandamos un pulso
alto al pin 23
      if (message =="led_on") {
        digitalWrite(LED, HIGH); // Encender el LED
        SerialBT.println("LED encendido"); // Envía el texto
a través del puerto Serial del BT
      }
      //Si el mensaje recibido fue "led_on" mandamos un pulso
bajo al pin 23
      else if (message =="led_off") {
        digitalWrite(LED, LOW); // Apagar el LED
        SerialBT.println("LED apagado"); // Envía el texto a
través del puerto Serial del BT
      }
    }
```

```
  }
}
void setup() {
  Serial.begin(115200); // Inicializando el monitor serial
  SerialBT.begin("ESP32"); // Nombre de tu Dispositivo
Bluetooth y en modo esclavo
  Serial.println("El dispositivo Bluetooth está listo para
emparejar");
  SerialBT.register_callback(callback_function); //
Registramos la función "callback_function" como función
callback.
  pinMode (LED, OUTPUT); // Configuramos el pin 23 como
salida
}
void loop() {
}
```

¿Como funciona el código?

Incluimos la librería del Bluetooth Serial

```
#include "BluetoothSerial.h"
```

Checamos si el Bluetooth se habilito correctamente

```
#if !defined(CONFIG_BT_ENABLED)
|| !defined(CONFIG_BLUEDROID_ENABLED)

#error ¡Bluetooth no está habilitado! Ejecute `make
menuconfig` y habilítelo

#endif
```

Definimos el pin GPIO 23 del ESP32 como el pin del Led

```
#define LED 23
```

Creamos el objeto SerialBT para más comodidad, pero se puede cambiar por el que sea

```
BluetoothSerial SerialBT;
```

Creamos las variables en donde se guardarán los datos recibidos, las guardaremos como String (Cadena de caracteres) y char (caracteres)

```
String message = "";
```

```
char incomingChar;
```

Creamos la función callback esto con el fin de en lugar de estar preguntando a cada rato si hay datos, esto crea como una alerta y en caso de haber datos ejecuta la acción, en caso contrario, no hará nada, usando el callback optimizamos y se ve mucho más organizado nuestro código.

```
void callback_function(esp_spp_cb_event_t event,
esp_spp_cb_param_t *param) {

//Evento Cuando se inicializa el servidor SPP
  if (event == ESP_SPP_START_EVT) {
    Serial.println("Inicializado SPP");
  }
//Evento Cuando un cliente SPP abre una conexión
  else if (event == ESP_SPP_SRV_OPEN_EVT ) {
    Serial.println("Cliente conectado");
  }
//Evento Cuando se cierra una conexión SPP
  else if (event == ESP_SPP_CLOSE_EVT  ) {
    Serial.println("Cliente desconectado");
  }
//Evento al recibir datos a través de una conexión SPP
  else if (event == ESP_SPP_DATA_IND_EVT ) {

    Serial.println("Datos recibidos");

    while (SerialBT.available()) { // Mientras haya datos por
recibir
      char incoming = SerialBT.read(); // Lee los datos recibidos
      if (incoming != '\n'){
      message += String(incoming);
    }
    //En caso de no haber datos limpiamos la variable message
    else{
      message = "";
    }
    //Imrpimimos el mensaje recibido
      Serial.print("Recibido: ");
      Serial.println(incoming);
      //Si el mensaje recibido fue "led_on" mandamos un pulso alto
al pin 23
      if (message =="led_on") {
        digitalWrite(LED, HIGH); // Encender el LED
        SerialBT.println("LED encendido"); // Envía el texto a
través del puerto Serial del BT
      }
      //Si el mensaje recibido fue "led_on" mandamos un pulso bajo
al pin 23
```

```
      else if (message =="led_off") {
        digitalWrite(LED, LOW); // Apagar el LED
        SerialBT.println("LED apagado"); // Envía el texto a
través del puerto Serial del BT
      }
    }
  }
}
```

La lista completa de las funciones callbak que posee el ESP32 es la siguiente:[66]

- ESP_SPP_INIT_EVT: Cuando el modo SPP es iniciado
- ESP_SPP_UNINIT_EVT: Cuando el modo SPP es terminado
- ESP_SPP_DISCOVERY_COMP_EVT: Cuando se completa el descubrimiento de servicios
- ESP_SPP_OPEN_EVT: Cuando un cliente SPP abre una conexión
- ESP_SPP_CLOSE_EVT: Cuando se cierra una conexión SPP
- ESP_SPP_START_EVT: Cuando se inicia el servidor SPP
- ESP_SPP_CL_INIT_EVT: Cuando un cliente SPP inicia una conexión
- ESP_SPP_DATA_IND_EVT: Al recibir datos a través de una conexión SPP
- ESP_SPP_CONG_EVT: Cuando cambia el estado de congestión en una conexión SPP
- ESP_SPP_WRITE_EVT: Al enviar datos a través de SPP.
- ESP_SPP_SRV_OPEN_EVT: Cuando un cliente se conecta al servidor SPP
- ESP_SPP_SRV_STOP_EVT: Cuando el servidor SPP se detiene

En el setup() iniciamos con el monitor serial para debug

```
Serial.begin(115200);
```

Llamamos al Bluetooth como “ESP32”

```
SerialBT.begin("ESP32");
```

Mandamos un mensaje que el dispositivo está listo para la comunicación:

```
Serial.println("El dispositivo Bluetooth está listo para
emparejar");
```

Registramos la función “callback_funcion” como callcack

```
SerialBT.register_callback(callback_function);
```

Configuramos el pin Led como salida

```
pinMode (LED, OUTPUT);
```

Para que no marque error de compilación el IDE de Arduino ponemos el loop() vacio

```
void loop() {
}
```

Después armaremos el siguiente circuito:

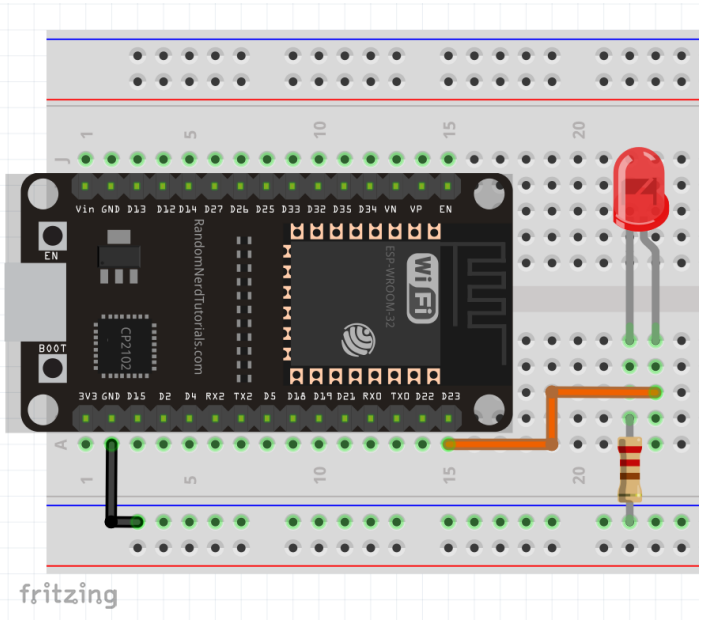


Ilustración 119: Diagrama del Ejercicio 1

Ahora cargamos el código al ESP32 y nos conectamos con la app “Serial Bluetooth” (Seguir los pasos de la practica 3)

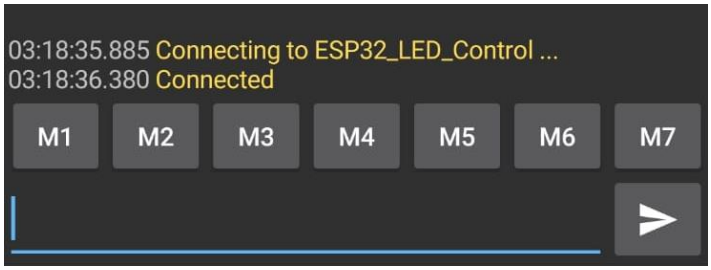


Ilustración 120: conexión del smartphone al ESP32

Mandamos el comando “led_on”, se debe encender el led conectado y nos retornara un mensaje que el led se encendió

Ilustración 121: Envió del comando "led_on"

Y cuando mandamos el comando “led_off” el led se deberá apagar y nos retornará un mensaje de que el led se apago

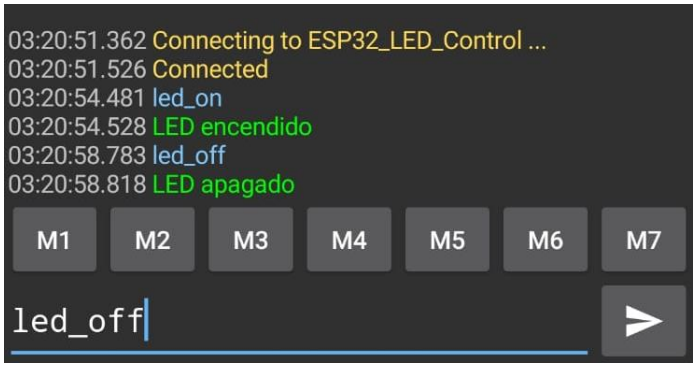


Ilustración 122: Envió del comando "led_off"

También podemos comprobar los mensajes recibidos en el monitor serial

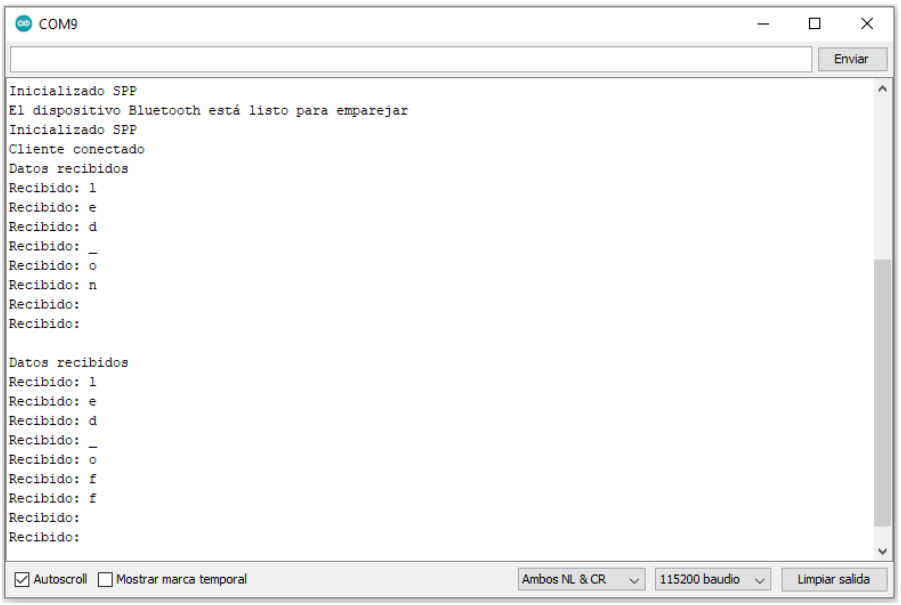


Ilustración 123: Recepción de los comandos atraves del monitor serial

En caso de querer controlar más leds solo añadimos más condiciones en la parte de lectura del mensaje, por ejemplo, en el código colocamos esto:

```
if (message =="led_on") {
digitalWrite(LED, HIGH);
SerialBT.println("LED encendido");
}
else if (message =="led_off") {
digitalWrite(LED, LOW);
SerialBT.println("LED apagado");
```

Si queremos agregar más leds podríamos poner algo como lo siguiente:

```
if (message =="led1_on") {
digitalWrite(LED1, HIGH);
SerialBT.println("LED1 encendido");
}
else if (message =="led1_off") {
digitalWrite(LED1, LOW);
SerialBT.println("LED1 apagado");
}
else if (message =="led2_on") {
digitalWrite(LED2, HIGH);
SerialBT.println("LED2 encendido");
}
else if (message =="led2_off") {
digitalWrite(LED2, LOW);
```

```
SerialBT.println("LED2 apagado");
}
```

Nota: No olvidar definir el pin de cada led a usar al inicio del código y en el setup() configurarlo como salida.

```
#define LED1 22
#define LED2 23
pinMode (LED1, OUTPUT);
pinMode (LED2, OUTPUT);
```

Ejercicio 2 Envió de datos del Sensor DTH22:

No solo podemos enviar comandos para controlar los puertos GPIO del ESP32, también podemos hacer que el ESP32 envié datos de un sensor cada cierto tiempo, como dejamos desocupado el loop() podemos poner un código ahí para la lectura de datos de un sensor y mandarlos cada 2 segundo al smartphone a la vez que podemos controlar los puertos del ESP32.

Para ello usaremos el siguiente código:

```
#include "BluetoothSerial.h" //Incluimos la librería del
Bluetooth Serial
#include "DHT.h" //Incluimos la librería del sensor DHT
#define DHTPIN 4     // Pin para conectarlo al sensor DHT
#define DHTTYPE DHT22   // Modelo del sensor
//Checamos si el Bluetooth se habilito correctamente
#if !defined(CONFIG_BT_ENABLED)
|| !defined(CONFIG_BLUEDROID_ENABLED)
#error ¡Bluetooth no está habilitado! Ejecute `make
menuconfig` y habilítelo
#endif
#define LED1 22 //Definimos el pin del led1 como el pin 21
del ESP32
#define LED2 23 //Definimos el pin del led2 como el pin 23
del ESP32

DHT dht(DHTPIN, DHTTYPE); //Inicializamos el DHT sensor
BluetoothSerial SerialBT; //llamamos al Bluetooth Serial como
"Serial BT"

//Variables en donde guardamos los datos recibidos
String message = "";
char incomingChar;
```

```
//Creamos la funcion callback
void callback_function(esp_spp_cb_event_t event,
esp_spp_cb_param_t *param) {
  if (event == ESP_SPP_START_EVT) {
    Serial.println("Inicializado SPP"); //Mensaje que el
Bluetooth se inicio correctamente
  }
  else if (event == ESP_SPP_SRV_OPEN_EVT ) {
    Serial.println("Cliente conectado"); //Mensaje que un
cliente se conecto correctamente
  }
  else if (event == ESP_SPP_CLOSE_EVT  ) {
    Serial.println("Cliente desconectado"); //Mensaje que el
cliente se desconecto
  }
  else if (event == ESP_SPP_DATA_IND_EVT ) {
    //Imrimimos si hay datos recibidos
    Serial.println("Datos recibidos");

    while (SerialBT.available()) { // Mientras haya datos por
recibir
      char incoming = SerialBT.read(); // Lee  los datos
recibidos
      if (incoming != '\n') {
        message += String(incoming);
      }
      //En caso de no haber datos limpiamos la variable
message
      else {
        message = "";
      }
      //Imrpimimos el mensaje recibido
      Serial.print("Recibido: ");
      Serial.println(incoming);
      //Si el mensaje recibido fue "led1_on" mandamos un
pulso alto al pin 22
      if (message == "led1_on") {
        digitalWrite(LED1, HIGH);
        SerialBT.println("LED1 encendido");
      }
      //Si el mensaje recibido fue "led1_off" mandamos un
pulso bajo al pin 22
      else if (message == "led1_off") {
        digitalWrite(LED1, LOW);
        SerialBT.println("LED1 apagado");
      }
```

```
      //Si el mensaje recibido fue "led2_on" mandamos un
pulso alto al pin 23
      else if (message == "led2_on") {
        digitalWrite(LED2, HIGH);
        SerialBT.println("LED2 encendido");
      }
      //Si el mensaje recibido fue "led2_off" mandamos un
pulso bajo al pin 23
      else if (message == "led2_off") {
        digitalWrite(LED2, LOW);
        SerialBT.println("LED2 apagado");

      }
    }
  }
}
void setup() {
  Serial.begin(115200); // Inicializando el monitor serial
  SerialBT.begin("ESP32"); // Nombre de tu Dispositivo
Bluetooth y en modo esclavo
  Serial.println("El dispositivo Bluetooth está listo para
emparejar");
  dht.begin(); //Inicializamos el sesnor DHT
  SerialBT.register_callback(callback_function); //
Registramos la función "callback_function" como función
callback.
  pinMode (LED1, OUTPUT); // Configuramos el pin LED1 como
salida
  pinMode (LED2, OUTPUT); // Configuramos el pin LED2 como
salida
}
void loop() {
  delay(2000);
  // Lectura de la humedad y la guardamos en h
  float h = dht.readHumidity();
  // Lectura de temperatura y lo guardamos en t
  float t = dht.readTemperature();
  // Pequeño chequeo de si las lecturas anteriores se
hicieron correctamente, si no, regresa a hacerlas de nuevo
  if (isnan(h) || isnan(t)) {
    SerialBT.println(F("Error! Lecura del sensor
incorrecta"));
    return;
  }
  //Enviamos los datos del sensor al smarthphone
  SerialBT.print(F("Humedad: "));
  SerialBT.print(h);
```

```
  SerialBT.print(F("%  Temperatura: "));
  SerialBT.print(t);
  SerialBT.println(F("°C "));

}
```

El código es muy similar al anterior por lo que solo explicaremos las líneas añadidas

Primero incluimos la librería, pin a usar y el modelo del sensor DHT:

```
#include "DHT.h"
#define DHTPIN 4
#define DHTTYPE DHT22
```

Después configuramos el sensor

```
DHT dht(DHTPIN, DHTTYPE);
```

En el setup() iniciamos el sensor

```
dht.begin();
```

Después en el loop() ponemos las instrucciones para leer los datos e enviarlos al smartphone por Bluetooth

Esperamos dos segundos antes de realizar la lectura de datos

```
delay(2000);
```

Lectura de la humedad y la guardamos en h como flotante

```
float h = dht.readHumidity();
```

Lectura de temperatura y lo guardamos en t como flotante

```
float t = dht.readTemperature();
```

Checamos de si las lecturas anteriores se hicieron correctamente, si no, regresa a hacerlas de nuevo

```
if (isnan(h) || isnan(t)) {
    SerialBT.println(F("Error! Lectura del sensor
incorrecta"));
    return;
 }
```

Enviamos los datos del sensor al smartphone por Bluetooth

```
SerialBT.print(F("Humedad: "));
SerialBT.print(h);
SerialBT.print(F("%  Temperatura: "));
SerialBT.print(t);
SerialBT.println(F("°C "));
```

Después armaremos el siguiente circuito:

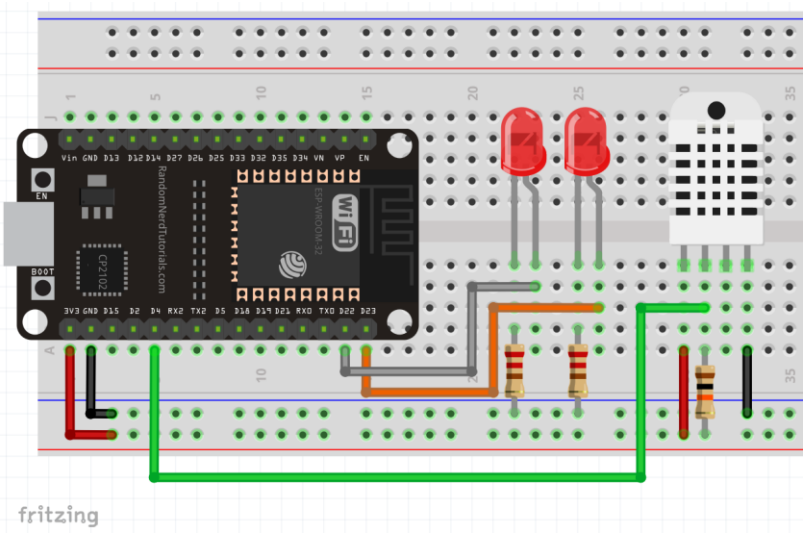


Ilustración 124: Diagrama del Ejercicio 2

Nos conectamos con el Bluetooth del ESP32 con la app "Serial Bluetooth Terminal" y si la conexión es correcta nos empezara a mandar los datos del sensor cada dos segundos:

Ilustración 125: Transmisión de datos del ESP32 al smartphone

Si mandamos los comandos led1_on encenderemos el led conectado al pin 22

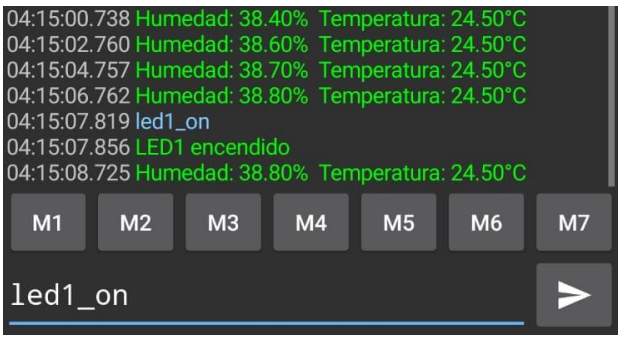


Ilustración 126: Envió del comando "led1_on"

En caso de que mandemos el comando led2_on encenderemos el led conectado al pin 23

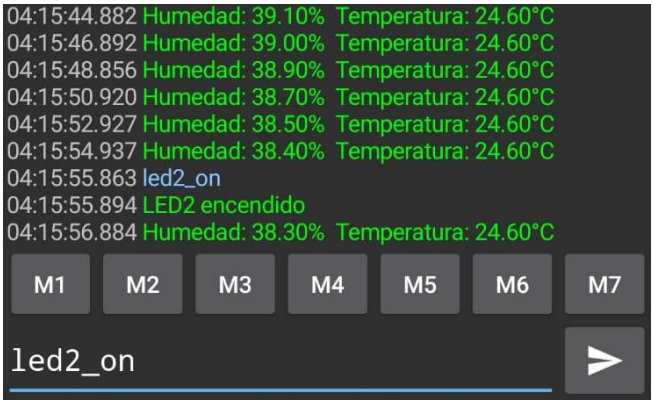


Ilustración 127: Envió del comando "led2_on"

Para apagarlos solo mandamos el comando led1_off y led2_off y los leds se deberán de apagar

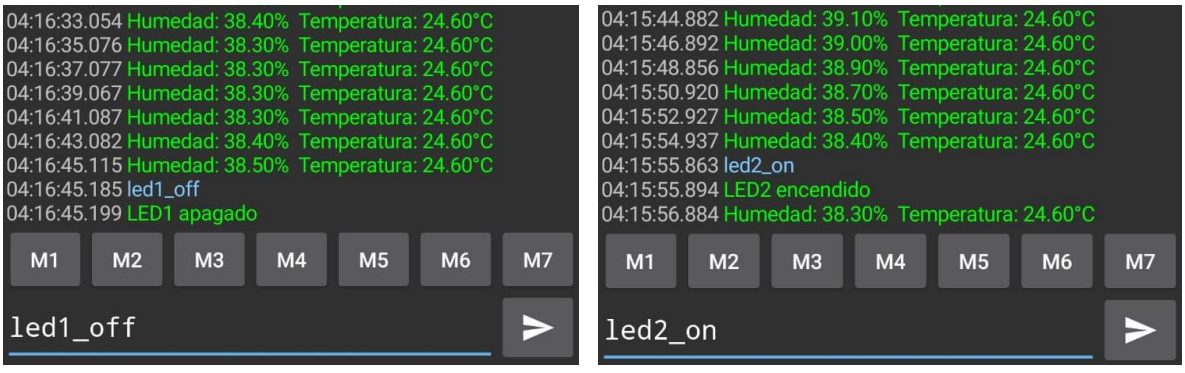


Ilustración 128: Envió de los comandos "led1_off" y "led2_off"

Trabajo de casa 8 de la practica 7: Comunicación entre el ESP32 y la app de mensajería Telegram

1. Mencione otras apps de mensajería instantánea aparte de Telegram
2. ¿Cuántos usuarios están registrados a la app de Telegram?
3. ¿Como funciona el protocolo MPTProto?
4. ¿Cuál es la diferencia entre el protocolo XMPP y MPTProto?
5. ¿En qué otras aplicaciones podemos encontrar un “bot”?
6. Describa los términos de Handler y webhook dentro de un bot en Telegram

Practica 7: Comunicación entre el ESP32 y la app de mensajería Telegram

Objetivos:

- Establecer una comunicación entre el ESP32 y un bot de Telegram usando internet
- Conocer que es un bot dentro de la app Telegram, así como su configuración
- Aprender a usar comandos para una comunicación con un bot en Telegram

Introducción:

Telegram

Es una plataforma de mensajería instantánea por internet y VOIP desarrollado por los hermanos Nikolái y Pavel Durόv, anunciada oficialmente el 14 de agosto de 2013 en Rusia, creada en un inicio por conflictos de intereses a las reformas implementadas a los servicios de internet de dicho país, por lo cual ex desarrolladores de la red social rusa VK acompañaron a Durov a la creación de Telegram.[67]

Ilustración 129: Icono de Telegram

Inicialmente fue implementado en teléfonos móviles Android y iOS, fue hasta el siguiente año que la app fue integrada a varias plataformas como macOS, Windows, GNU/Linux, Firefox OS, etc. Parte de su software es libre, por lo cual recibe mucho apoyo de la comunidad, solo la parte en el servidor es privada.

La aplicación está enfocada a la mensajería instantánea, envió de archivos y la comunicación entre varias personas. La información es cifrada mediante el protocolo MTProto en dos mecanismos por lo cual cualquier información que se transfiera atraves de la aplicación está cifrada integra e independientemente vía servidor.[68]

BOT

Es un programa informático autónomo controlado por inteligencia artificial, secuencias de texto o botones interactivos, encargados principalmente en ofrecer servicios vía internet.[69]

Telegram Bot API

Es el nombre a una interfaz de programación del servicio de mensajería de Telegram orientada en bots, cualquier persona con una cuenta de Telegram puede tener acceso a un bot solo con tener el nombre de usuario.[70]

Realizan servicios para los usuarios como lo son, administración de canales y grupos, compartir contenido, elaboración de votaciones, ejecutar juegos y realizar pagos, gracias a esto muchos países emplean estos bots para realizar consultas, pedidos a domicilio, etc.[71]

Cada bot de Telegram tiene que ser creado via @botfather, la cual tiene una serie de parámetros:

- Información básica del bot: nombre, foto de perfil y descripción
- Alias: nombre de usuario con sufijo “bot”
- Clave secreta o toker: Para su integración con servicios ajenos a Telegram

La primera versión de Bot API 1 fue anunciada en junio de 2015, el cual integraba solo bot conversacional, es decir, solo estaba diseñado a dar respuestas u ordenes instantáneas mediante comandos de texto.

La segunda versión llamada Bot API 2 fue anunciada en abril de 2016 la cual incluía varias mejoras de interfaz, dando la posibilidad de que puedan ser personalizados por el gusto del usuario, también se integró la opción de pago un año después.[71]

Material para el desarrollo de la práctica:

- Placa ESP32
- Sensor de temperatura y humedad DHT22
- Jumpers
- 1 Resistencia de 10k Ohm a ½ watt
- Protoboard

- Cable micro-USB
- Celular con Android o IPhone

Desarrollo:

Nota: Para el desarrollo de esta práctica es necesario tener acceso a internet

Antes de empezar con los ejercicios primero debemos de añadir dos librerías al IDE de Arduino

La primera será la librería Universal Telegram Bot Library creada por Brian Lough que nos proporciona una interfaz para la comunicación de Telegram Bot API.

Abrimos el siguiente link de descarga y nos mandara a un repositorio de github.com

https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot

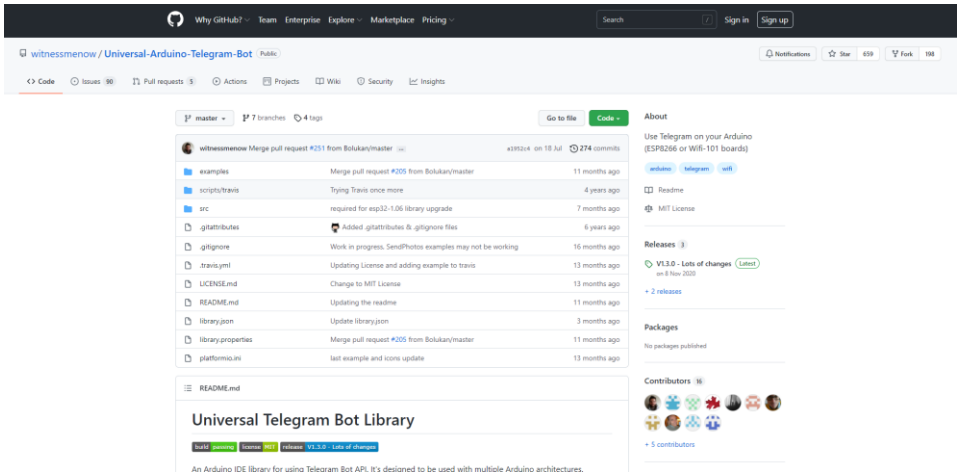


Ilustración 130: Pagina de descarga para la librería ""Universal Arduino Telegram Bot "

Le damos clic en el botón “Code” y se desplegara un submenú y seleccionamos la opción “Download ZIP”

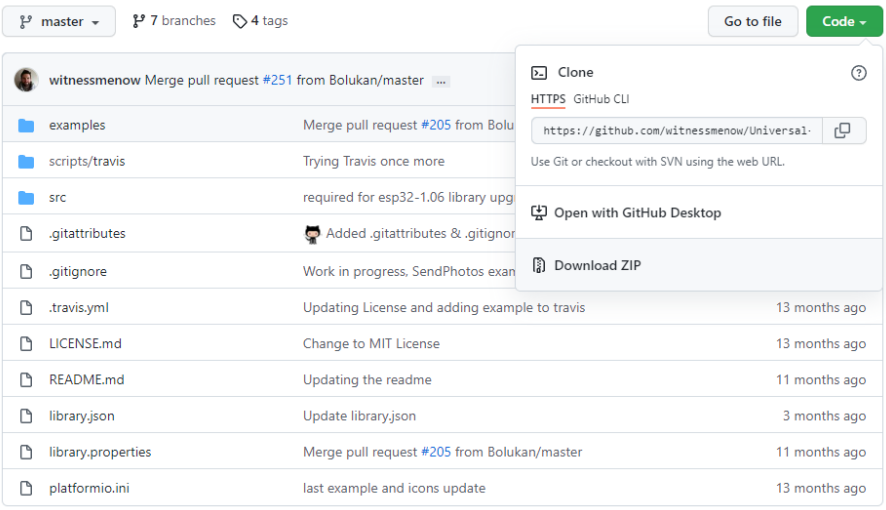


Ilustración 131: Opción de descarga para la librería ""Universal Arduino Telegram Bot "

Lo guardamos en cualquier ubicación y esperamos a que se descargue

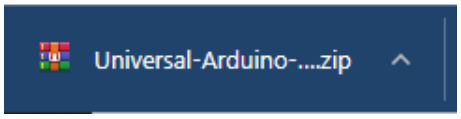


Ilustración 132: Archivo .ZIP de la librería "Universal Arduino Telegram Bot"

Luego nos vamos al IDE de Arduino y en la pestaña de “programa” - “Incluir librería” seleccionamos “Añadir Biblioteca.ZIP...”

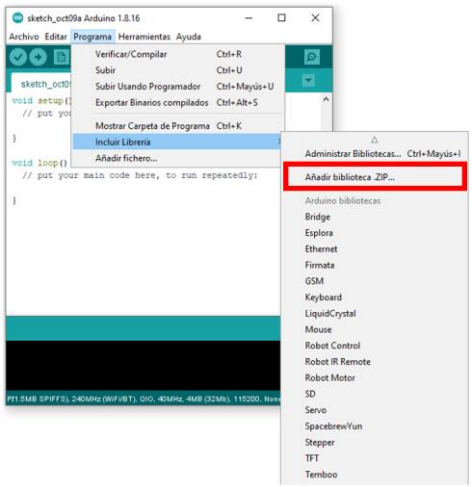


Ilustración 133: Menú para la instalación de librerías mediante archivo .ZIP

Después nos abrirá una venada de búsqueda, buscamos el archivo que descargamos y le damos clic en “abrir”

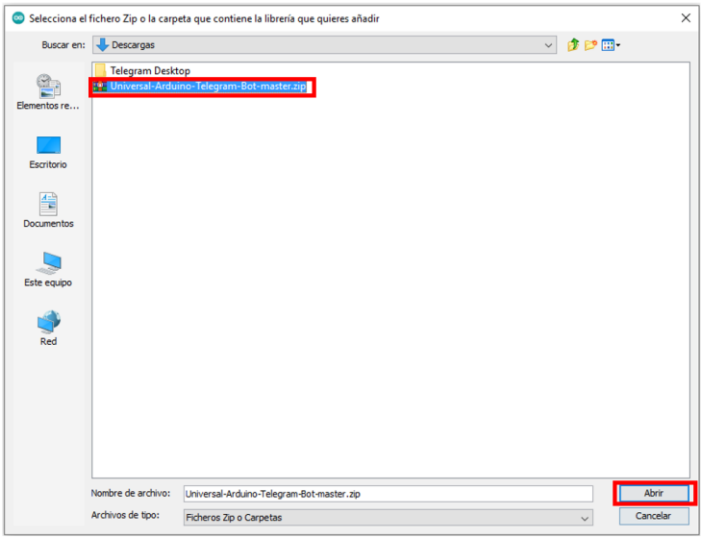


Ilustración 134: Ventana de búsqueda

Esperamos unos segundos y estará instalada.

Para la segunda librería nos dirigimos igual a la pestaña de “programa”- “incluir librería” pero esta vez seleccionamos “Administrar bibliotecas”

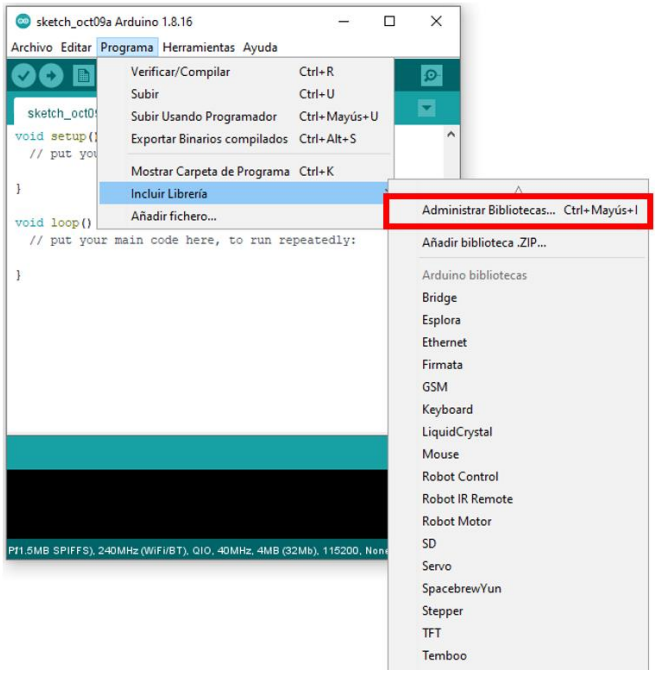


Ilustración 135: Menú para abrir la ventana de "Administrador de bibliotecas"

Se abrirá una nueva ventana como esta:

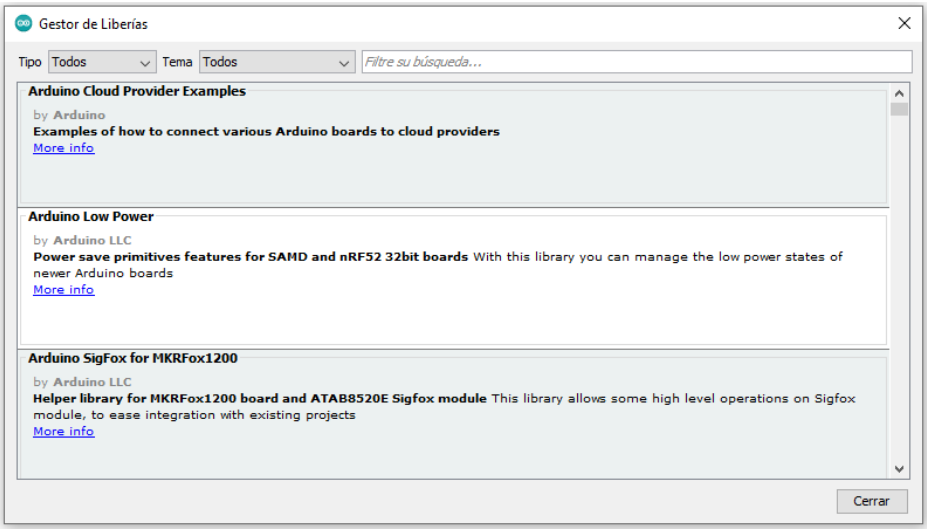


Ilustración 136: Ventana del gestor de Librerías

En el cuadro de búsqueda ponemos "ArduinoJson", presionamos Enter, y seleccionamos "Instalar" en la opción marcada en el cuadro rojo:

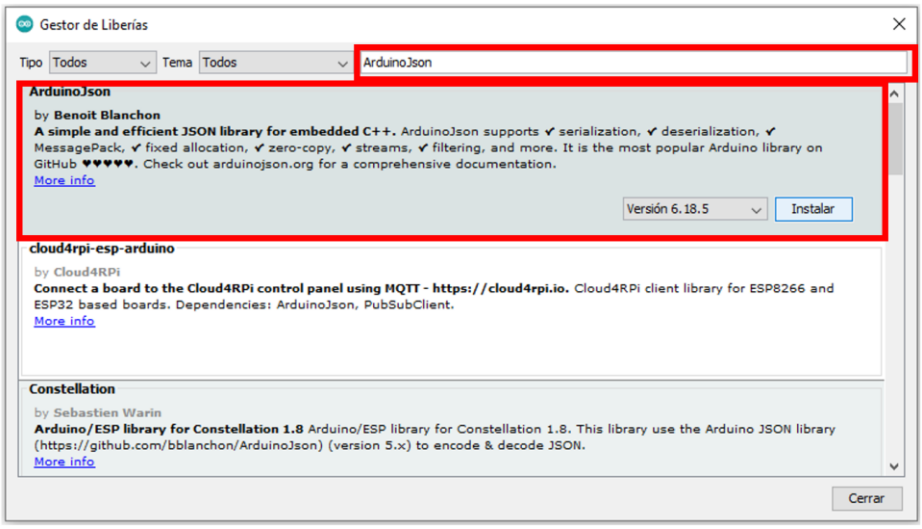


Ilustración 137: Buscando la librería "ArdionoJson" en el gestor de librerías

Esperamos a que este instalada y reiniciaos el IDE de Arduino.

Ejercicio 1 Control de puertos GIPO del ESP32:

Primero configuraremos la App de Telegram

Nota: Se debe tener una cuenta de Telegram ya echa para los siguientes pasos, si no se tiene, crearla antes de continuar.

Abrimos la App de Telegram

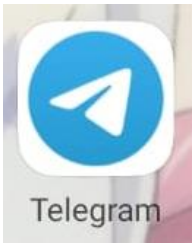


Ilustración 138: Icono de Telegram en Android

En el cuadro de búsqueda escribimos “botfather” y seleccionamos la opción que tiene una palomita azul (cuadro rojo)

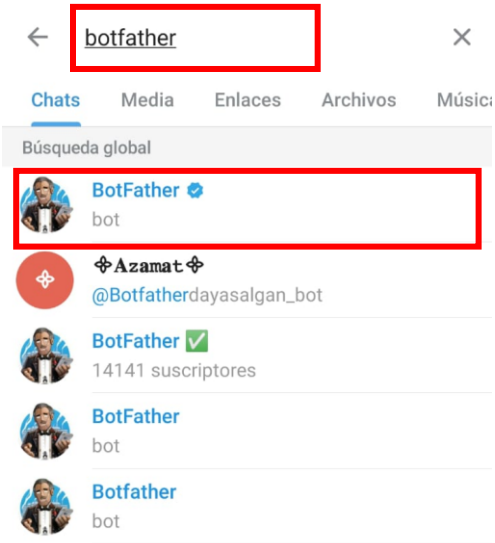


Ilustración 139: Buscar "botfather"

Se abrirá un chat nuevo, presionamos en “iniciar”

Ilustración 140: Chat con el BotFather

Nos mostrara un mensaje con los comandos que podemos usar, le damos clic en /newbot o lo escribimos manualmente

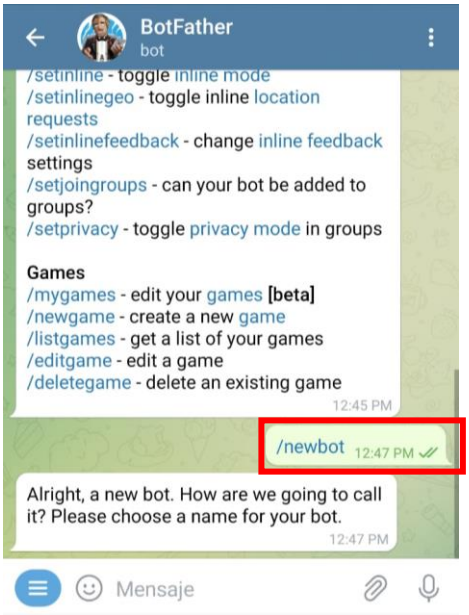


Ilustración 141: Creación de un nuevo bot

Escogemos un nombre para llamar a nuestro bot, en este caso lo llamaremos “ESP32” pero podemos el nombre que nosotros queramos

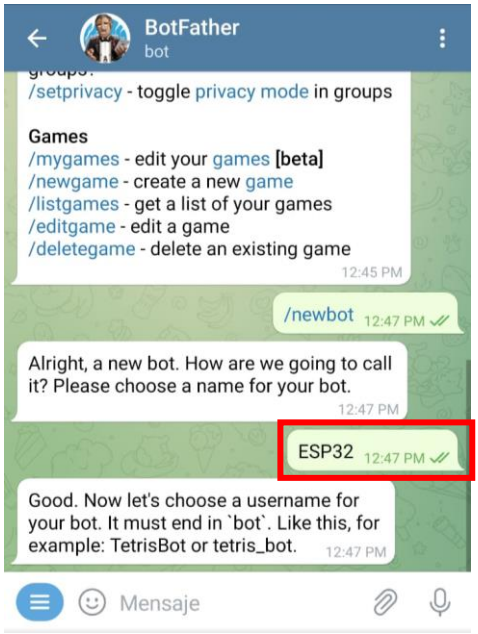


Ilustración 142: Nombre del bot "ESP32"

Luego nos pedirá un nombre de usuario, pueden poner el que sea, pero si ya está ocupado o lo usa otro Bot les dirá que no está disponible

Ilustración 143: Mensaje de error en caso de que el usuario este ocupado

Si el nombre de usuario es correcto y no está siendo usado nos mandara el siguiente mensaje con el link de nuestro bot y el token de acceso

Nota: No compartir estos datos por temas de seguridad

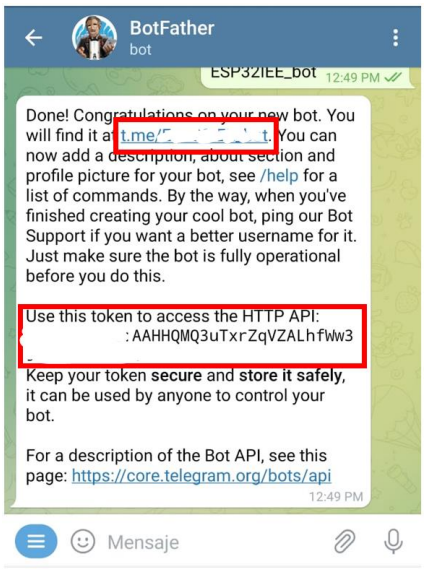


Ilustración 144: Creación exitosa de un bot en Telegram

Después necesitamos saber nuestro ID en Telegram esto para que el ESP32 solo se pueda comunicar con nosotros y no con otro usuario de Telegram

Para eso nos dirigimos al cuadro de búsqueda de Telegram y escribimos “IDBot” y seleccionamos la opción del cuadro rojo

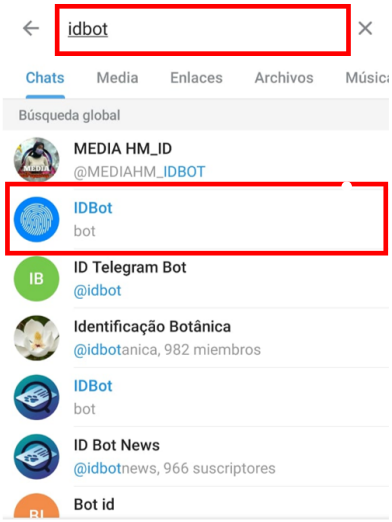


Ilustración 145: Buscar "IDbot"

Se abrirá un nuevo chat, presionamos el botón “iniciar”

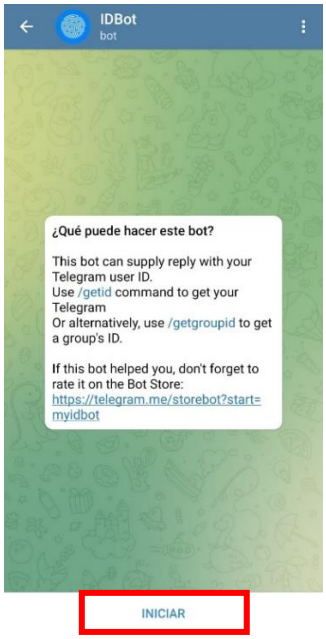


Ilustración 146: Chat con el bot "IDBot"

Luego seleccionamos el comando /getid o lo escribimos manualmente

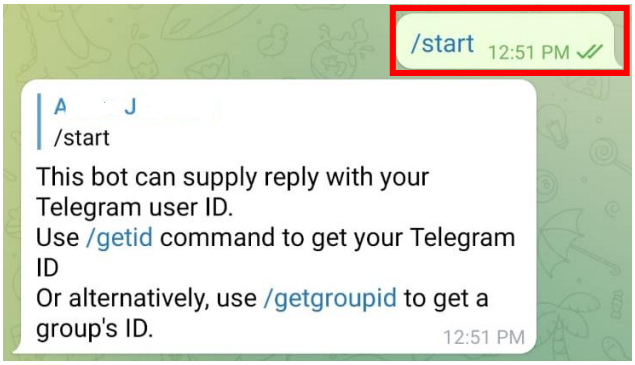


Ilustración 147: Inicio de la conversación con el bot "IDBot"

Nos mandara un mensaje con el número de nuestro ID, NO compartir este número por temas de seguridad:

Ilustración 148: Obtención de nuestro ID de Telegram

Ya tenemos todo listo, ahora para este ejercicio veremos primero el control de un led mediante comandos, para ello usaremos el siguiente código:

```
#include <Wi-Fi.h> //Incluimos la librería para usar el Wi-Fi del ESP32
#include <Wi-FiClientSecure.h>
#include <UniversalTelegramBot.h> //Librería del Bot de Telegram
#include <ArduinoJson.h> //Libreria de complemento

// SSID y Pasword de la red Wi-Fi a conectar, reemplace con su red Wi-Fi
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Inicializamos el Bot de Telegram
#define BOTtoken "XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  // Bot token, reemplácelo por su bot creado anteriormente"

//ID de Telegram para que solo se pueda comunicarse con un solo usuario
```

```
#define CHAT_ID "XXXXXXXXXX"

Wi-FiClientSecure client; //Iniciamos el Wi-Fi en modo
cliente
UniversalTelegramBot bot(BOTtoken, client); //Mandamos esa
información al Telegram Bot

// Chequeo de nuevos mensajes cada 1 segundo
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;

const int ledPin = 2; //Pin a usar del ESP32
bool ledState = LOW; //Estado inicial del Pin

// Subprograma que controla que sucede cuando un nuevo
mensaje es enviado de Telegram al ESP32
void handleNewMessages(int numNewMessages) {
  Serial.println("Nuevo mensaje: ");
  Serial.println(String(numNewMessages));

  for (int i = 0; i < numNewMessages; i++) {
    // Verificamos que coincida con nuestro ID, en caso
contrario mandara el mensaje de "Usuario no autorizado"
    String chat_id = String(bot.messages[i].chat_id);
    if (chat_id != CHAT_ID) {
      bot.sendMessage(chat_id, "Usuario no autorizado", "");
      continue;
    }

    // Imprimimos el mensaje recibido
    String text = bot.messages[i].text;
    Serial.println(text);

    String from_name = bot.messages[i].from_name;
//Almacenamos nuestro nombre de usuario de Telegram
    //Mandamos un mensaje con los comandos que se pueden
utilizar cuando se reciba el comando /start
    if (text == "/start") {
      String welcome = "Bienvenido, " + from_name + ".\n";
      welcome += "Utilice los siguientes comandos para
controlar sus salidas. \n\n";
      welcome += "/led_on Para encender GPIO \n";
      welcome += "/led_off Para apagar GPIO \n";
      welcome += "/state para solicitar el estado actual de
GPIO \n";
      bot.sendMessage(chat_id, welcome, "");
    }
```

```
    // Si recivimos el comando /led_on, mandamos pulso alto
al pin 2 y mandamos el mensaje que el "Led está encendido"
    if (text == "/led_on") {
      bot.sendMessage(chat_id, "Led encendido", "");
      ledState = HIGH;
      digitalWrite(ledPin, ledState);
    }
    // Si recivimos el comando /led_off, mandamos pulso bajo
al pin 2 y mandamos el mensaje que el "Led esta apagado"
    if (text == "/led_off") {
      bot.sendMessage(chat_id, "Led apagado", "");
      ledState = LOW;
      digitalWrite(ledPin, ledState);
    }
   // Si recivimos el comando /state, verificamos el estado
del pin, si está en estado alto mandamos el mensaje:
   //Led está encendido, si esta apagado mandamos el mensaje
"Led esta apagado"
    if (text == "/state") {
      if (digitalRead(ledPin)) {
        bot.sendMessage(chat_id, "Led esta encendido", "");
      }
      else {
        bot.sendMessage(chat_id, "Led esta apagado", "");
      }
    }
  }
}

void setup() {

  Serial.begin(115200); //Inicializamos el monitor serial
  pinMode(ledPin, OUTPUT); //configuramos el pin 2 como
salida
  digitalWrite(ledPin, ledState); //Mandamos un estado bajo
para apagar el Led para que inicie asi.
  client.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Añadimos la
certificación de Telegram
  //Conectamos a nuestra red Wi-Fi
  Wi-Fi.mode(WI-FI_STA);
  Wi-Fi.begin(ssid, password);
  //Esperamos a que se conecte a nuestra res Wi-Fi
  while (Wi-Fi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a la Red Wi-Fi...");
  }
```

```
  // En caso de conexión exitosa imprimimos la dirección IP
del ESP32
  Serial.println(Wi-Fi.localIP());
}

void loop() {
  //Revisamos si hay mensajes nuevos cada segundo
  if (millis() > lastTimeBotRan + botRequestDelay)  {
    int numNewMessages =
bot.getUpdates(bot.last_message_received + 1);

  //Cuando recibimos un nuevo mensaje llamamos al subprograma
handleNewMessages
    while (numNewMessages) {
      Serial.println("Mensaje nuevo!");
      handleNewMessages(numNewMessages);
      numNewMessages =
bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis();
  }
}
```

Cargamos el código al ESP32, después abrimos el monitor serial y presionamos el botón “EN” y esperamos hasta que se conecte a nuestra red Wi-Fi

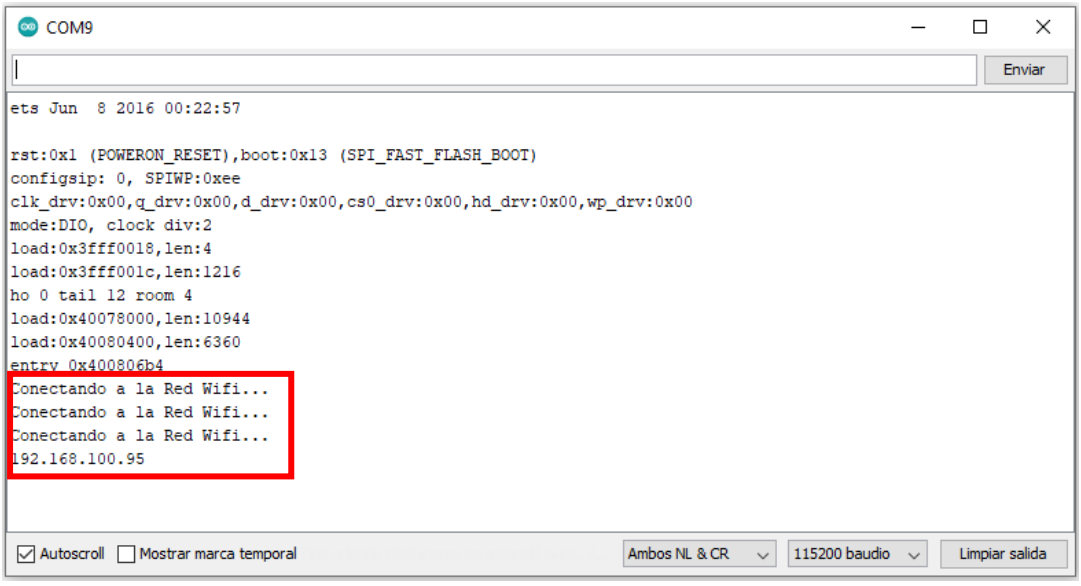


Ilustración 149: Ventada del monitor serial mostrando una conexión correcta a una red Wi-Fi

Ahora nos dirigimos al link de nuestro bot que creamos al inicio

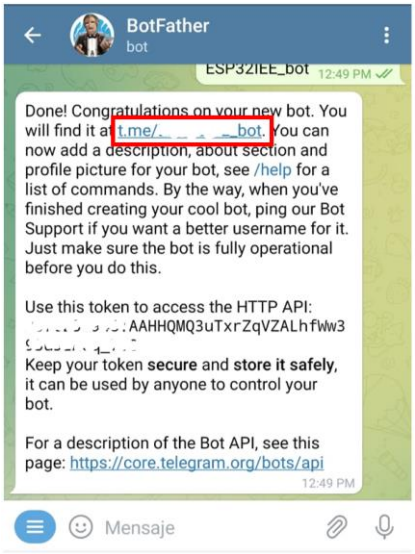


Ilustración 150: Obtención del link de nuestro bot

Se abrirá un nuevo chat y seleccionamos la opción “iniciar”

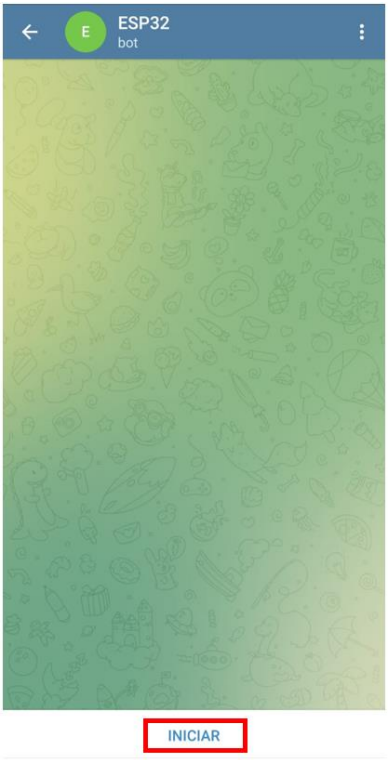


Ilustración 151: Chat de inicio con nuestro bot "ESP32"

Si colocamos nuestro token y el ID de Telegram correctamente nos debe de mostrar un mensaje como el siguiente:

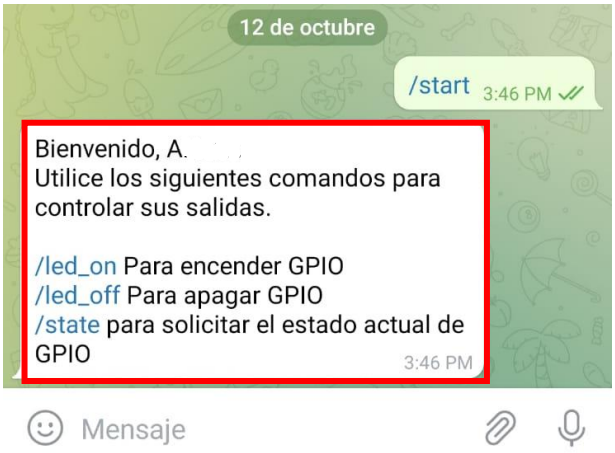


Ilustración 152: Mensaje de bienvenida del ESP32

Ahora solo seleccionamos el comando o lo escribimos manualmente, si seleccionamos el comando /led_on encenderá el led que tiene la board del ESP32 y nos mostrara un mensaje que el led ha sido encendido

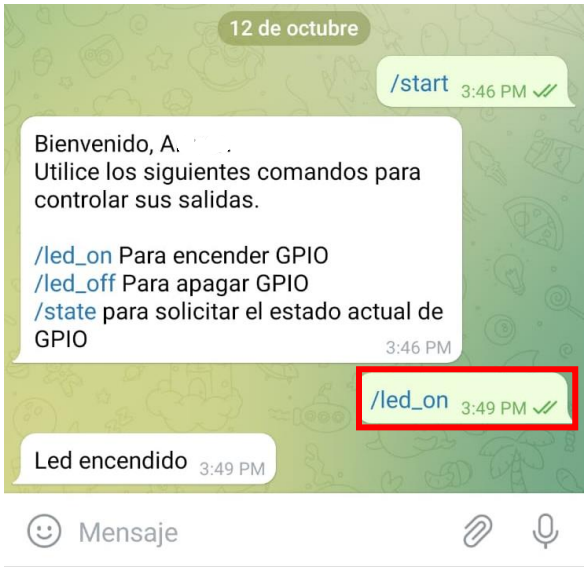


Ilustración 153: Envió del comando "/led_on"

En caso del comando /State nos mostrara un mensaje del estado del Led, encendido o apagado

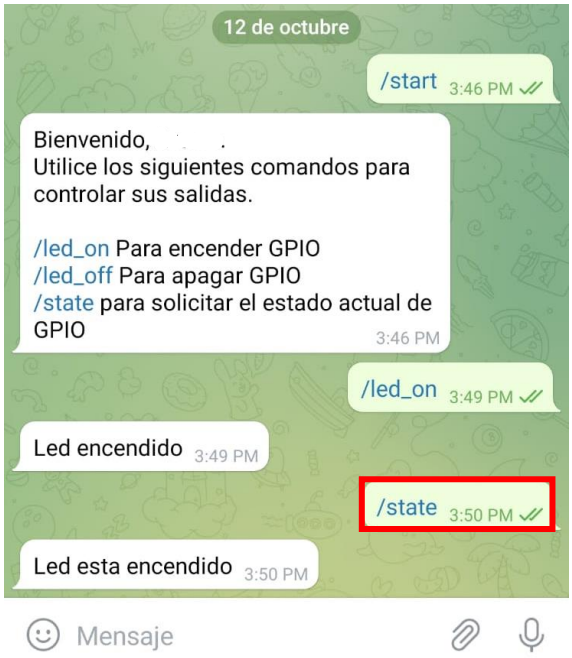


Ilustración 154: Envió del comando "/state"

También podemos comprobar los mensajes que se reciben mediante el monitor serial

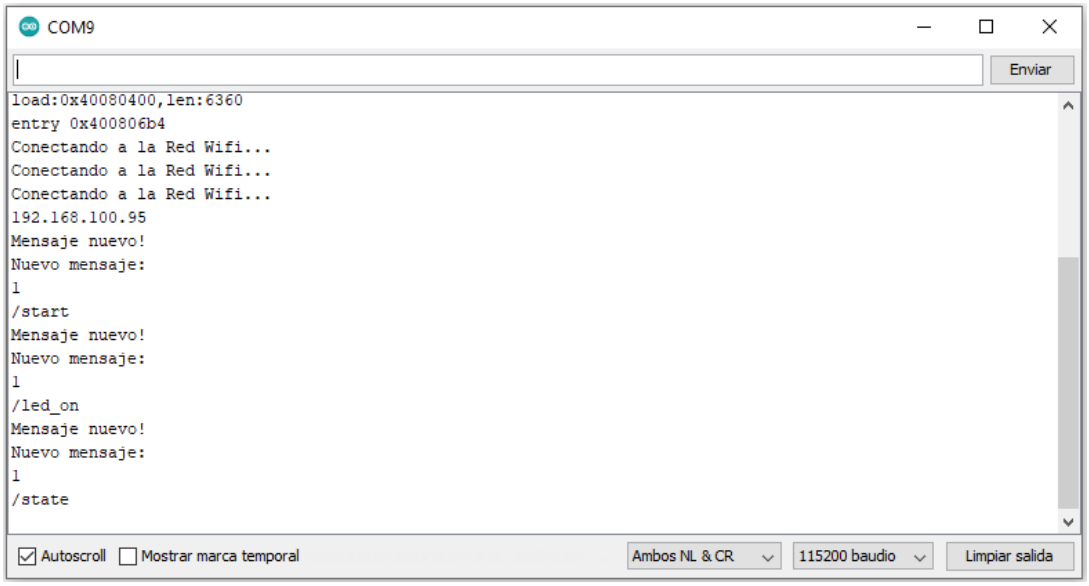


Ilustración 155: Comprobación de mensajes recibidos mediante el monitor serial

Si el bot interactúa con otro usuario de Telegram o nuestro Id que colocamos es incorrecto nos retornara un mensaje de “Usuario no autorizado”

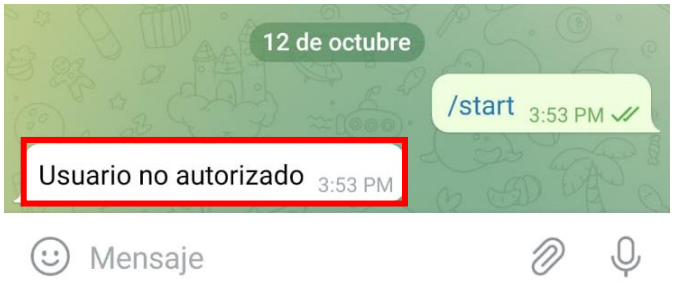


Ilustración 156: Mensaje de "error"

Ejercicio 2 Envió de datos del ESP32 a Telegram:

No solo podemos mandar comandos para controlar las salidas del ESP32, también podemos crear comandos para que nos envié los datos de un sensor, en este caso usaremos el sensor DHT22, cada vez que mandemos el comando el ESP32 nos mandara los datos de temperatura y humedad.

Nota: Seguir los pasos de la practica 5 para instalar las librerías del sensor DHT22

Primero transcribiremos el siguiente código:

```
#include <Wi-Fi.h> //Incluimos la librería para usar el Wi-Fi
del ESP32
#include <Wi-FiClientSecure.h>
#include <UniversalTelegramBot.h> //Librería del Bot de
Telegram
#include <ArduinoJson.h> //Librería de complemento
#include "DHT.h" //Incluimos la librería del sensor DHT
#define DHTPIN 4     // Pin para conectarlo al sensor DHT
#define DHTTYPE DHT22   // Modelo del sensor


// SSID y Pasword de la red Wi-Fi a conectar, reemplace con
su red Wi-Fi
const char* ssid = "REPLACE_WITH_YOUR_SSID ";
const char* password = " REPLACE_WITH_YOUR_PASSWORD ";

//ID de Telegram para que solo se pueda comunicarse con un
solo usuario
#define CHAT_ID " XXXXXXXXXX "

// Inicializamos el Bot de Telegram
```

```
#define BOTtoken "
XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX "  // Bot
token, reemplácelo por su bot creado anteriormente

Wi-FiClientSecure client; //Iniciamos el Wi-Fi en modo
cliente
UniversalTelegramBot bot(BOTtoken, client); //Mandamos esa
información al Telegram Bot

// Chequeo de nuevos mensajes cada 1 segundo
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;

DHT dht(DHTPIN, DHTTYPE); //Inicializamos el DHT sensor

// Leemos los valores del sensor DHT y regresamos el valor en
un string
String getReadings() {
  // Lectura de la humedad y la guardamos en h
  float h = dht.readHumidity();
  // Lectura de temperatura y lo guardamos en t
  float t = dht.readTemperature();
  //Guardamos los valores en un string
  String message = "Temperatura: " + String(t) + " °C \n";
  message += "Humedad: " + String (h) + " % \n";
  return message;
}

// Subprograma que controla que sucede cuando un nuevo
mensaje es enviado de Telegram al ESP32
void handleNewMessages(int numNewMessages) {
  Serial.println("Nuevo mensaje: ");
  Serial.println(String(numNewMessages));

  for (int i = 0; i < numNewMessages; i++) {
    // Verificamos que coincida con nuestro ID, en caso
contrario mandara el mensaje de "Usuario no autorizado"
    String chat_id = String(bot.messages[i].chat_id);
    if (chat_id != CHAT_ID) {
      bot.sendMessage(chat_id, "Usuario no autorizado", "");
      continue;
    }

    // Imprimimos el mensaje recibido
    String text = bot.messages[i].text;
    Serial.println(text);
```

```
    String from_name =
bot.messages[i].from_name;  //Almacenamos nuestro nombre de
usuario de Telegram

    //Mandamos un mensaje con los comandos que se pueden
utilizar cuando se reciba el comando /start
    if (text == "/start") {
      String welcome = "Bienvenido, " + from_name + ".\n";
      welcome += "Utilice el siguiente comando para realizar
una lectura. \n\n";
      welcome += "/readings \n";
      bot.sendMessage(chat_id, welcome, "");
    }
    // Si recibimos el comando /readings leemos las lecturas
del sensor DHT y las enviamos
    if (text == "/readings") {
      String readings = getReadings();
      bot.sendMessage(chat_id, readings, "");
    }
  }
}
void setup() {
  Serial.begin(115200); //Inicializamos el monitor serial
  dht.begin(); //Inicializamos el sensor DHT
  //Conectamos a nuestra red Wi-Fi
  Wi-Fi.mode(WI-FI_STA);
  Wi-Fi.begin(ssid, password);

  client.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Añadimos la
certificación de Telegram

  //Esperamos a que se conecte a nuestra red Wi-Fi
  while (Wi-Fi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a la Red Wi-Fi...");
  }
  // En caso de conexión exitosa imprimimos la dirección IP
del ESP32
  Serial.println(Wi-Fi.localIP());
}
void loop() {
  //Revisamos si hay mensajes nuevos cada segundo
  if (millis() > lastTimeBotRan + botRequestDelay)  {
    int numNewMessages =
bot.getUpdates(bot.last_message_received + 1);
```

```
    //Cuando recibimos un nuevo mensaje llamamos al
subprograma handleNewMessages
    while (numNewMessages) {
      Serial.println("Mensaje nuevo!");
      handleNewMessages(numNewMessages);
      numNewMessages =
bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis();
  }
}
```

Cargamos el código el ESP32, después abrimos el monitor serial y presionamos el botón “EN” esperamos hasta que se conecte a nuestra red Wi-Fi

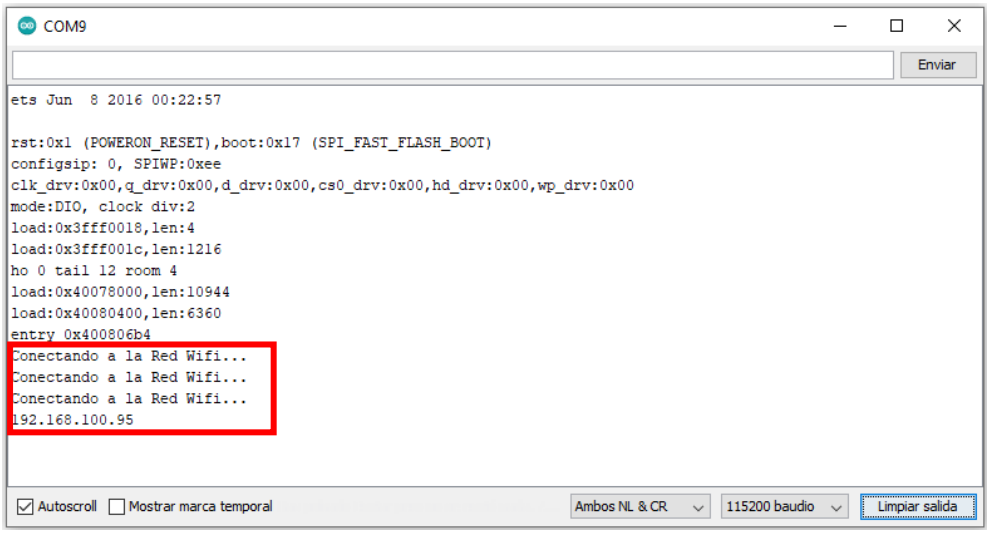


Ilustración 157: Ventada del monitor serial mostrando una conexión correcta a una red Wi-Fi

Ahora nos dirigimos al link de nuestro bot que creamos al inicio

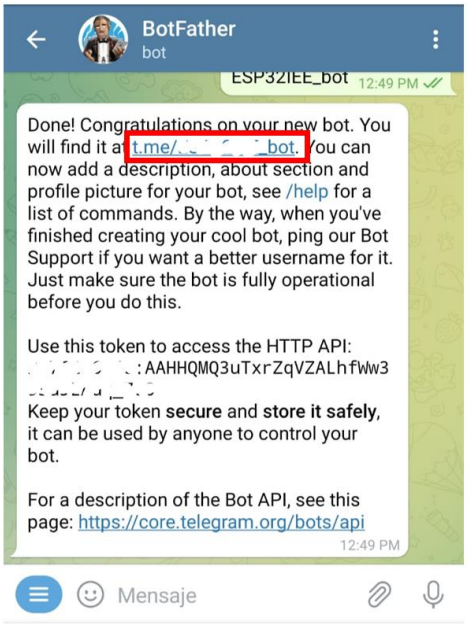


Ilustración 158: Obtención del link de nuestro bot

Se abrirá un nuevo chat y seleccionamos la opción “iniciar”

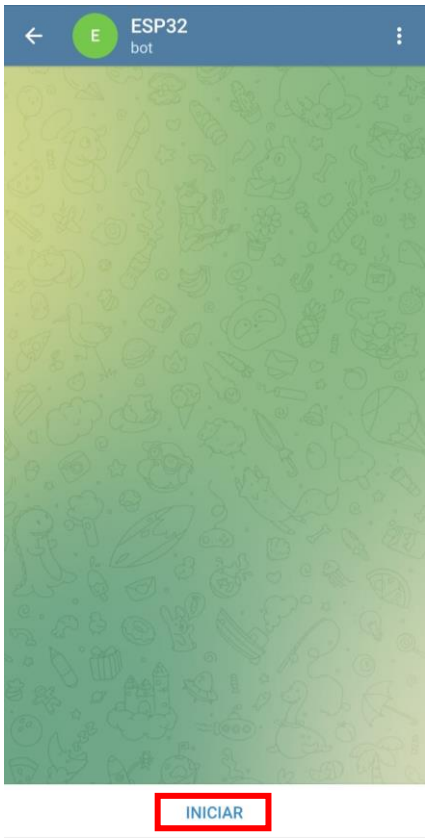


Ilustración 159: Chat con el bot "ESP32"

Si colocamos nuestro token y el ID de Telegram correctamente nos debe de mostrar un mensaje como el siguiente:

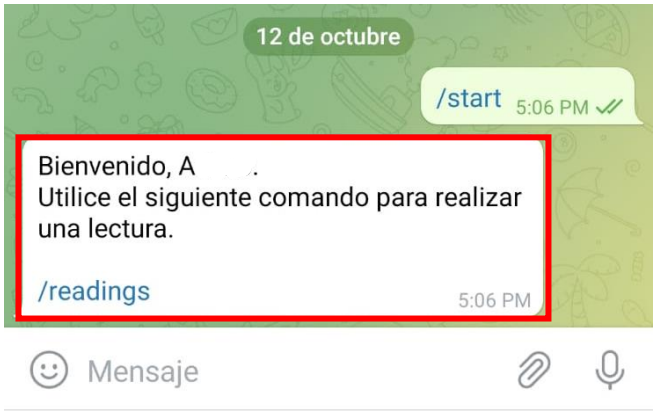


Ilustración 160: Mensaje de bienvenida del ESP32

Ahora solo seleccionamos el comando /readings o lo escribimos manualmente y nos retornara un mensaje con las variables del sensor DHT22

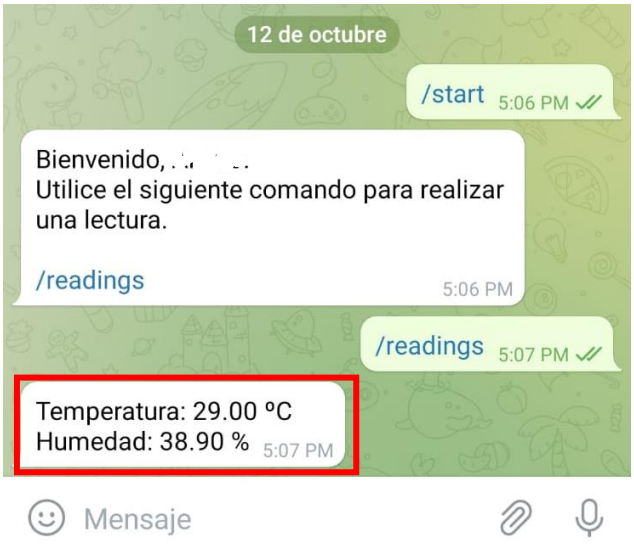


Ilustración 161: Mensaje con datos enviados del ESP32

También podemos comprobar los mensajes que se reciben mediante el monitor serial

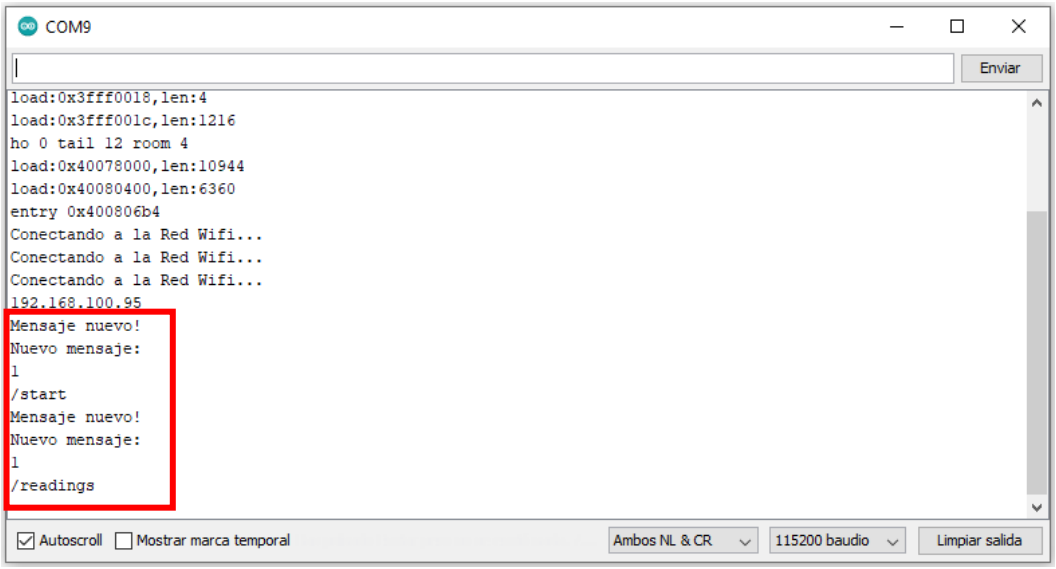


Ilustración 162: Comprobación de mensajes recibidos mediante el monitor serial

Trabajo de casa 9 de la Practica 8: Conexión del ESP32 a la plataforma ThingSpeak

1. ¿Qué es un servidor en la nube?
2. ¿Qué es MATLAB?
3. Como funciona el protocolo MQTT
4. Defina que es un Broker cloud
5. Mencione que es un muestreo de datos
6. Aparte de ThingSpeak, ¿qué otras plataformas existen para el almacenamiento de datos para el internet de las cosas?
7. ¿Cuáles son las ventajas de usar servidores en la nube para el internet de las cosas?
8. ¿Qué aplicaciones hay para usar la plataforma de ThingSpeak con el uso de sensores?

Practica 8: Conexión del ESP32 a la plataforma ThingSpeak

Objetivos:

- Entablar una comunicación entre el ESP32 y ThingSpeak
- Conocer una manera de enviar datos de sensores a un "servidor en la nube" como ThingSpeak
- Ver las ventajas y desventajas de usar ThingSpeak para futuros proyectos
- Como Configurar ThingSpeak para que sea capaz de recibir los datos de nuestro ESP32

Introducción:

ThingSpeak:

Es una plataforma de código abierto diseñada para el internet de las cosas, es decir, permite la comunicación entre dispositivos y personas, la plataforma permite en análisis y visualización de datos en tiempo real a través de un servidor en la nube.

Ilustración 163: Logo de ThingSpeak

Su desarrollo se remonta desde el año de 2011 en la plataforma de Github, está basada en la plataforma *Ruby on Rail*s (RoR), su arquitectura está basada en el Modelo Vista Controlador el cual es una arquitectura de software que separa la información en una aplicación, interfaz de usuario y lógica de control en partes distintas.[72]

- Modelo: Solo representa los datos
- Vista: Es la interfaz de usuario del sistema
- Controlador: Es el intermediario entre el Modelo y la Vista, maneja la información que se envía entre ellos y las transforma para que haya una buena comunicación entre ambos

También la plataforma está integrada con los servidores de MATLAB, la cual nos permite hacer cálculos con los datos que registra el servidor del o los sensores, como, por ejemplo, escala de tiempo, promedios, mediana, sumatoria y redondeo.[73]

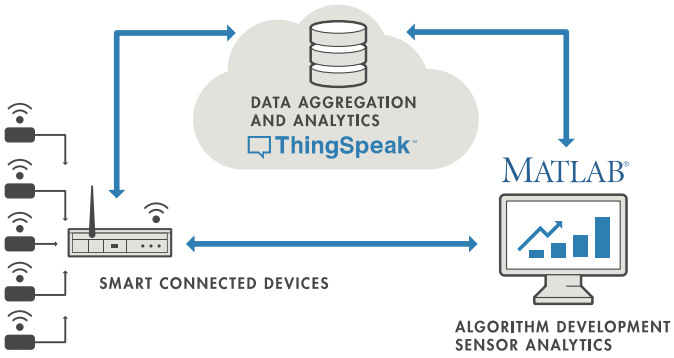


Ilustración 164: Diagrama de la plataforma ThingSpeak

Usa el protocolo de transferencia de hipertexto HTTP atraves de internet, también se pueden crear aplicaciones que registran las lecturas de sensores, localización y una red social, como por ejemplo Twitter, es decir, a través de tweets se puede comunicar un dispositivo a en este caso a los seguidores de Twitter que tenga esa cuenta, toda esa información se muestra atraves de gráficas, la cuales también podemos editar.[74]

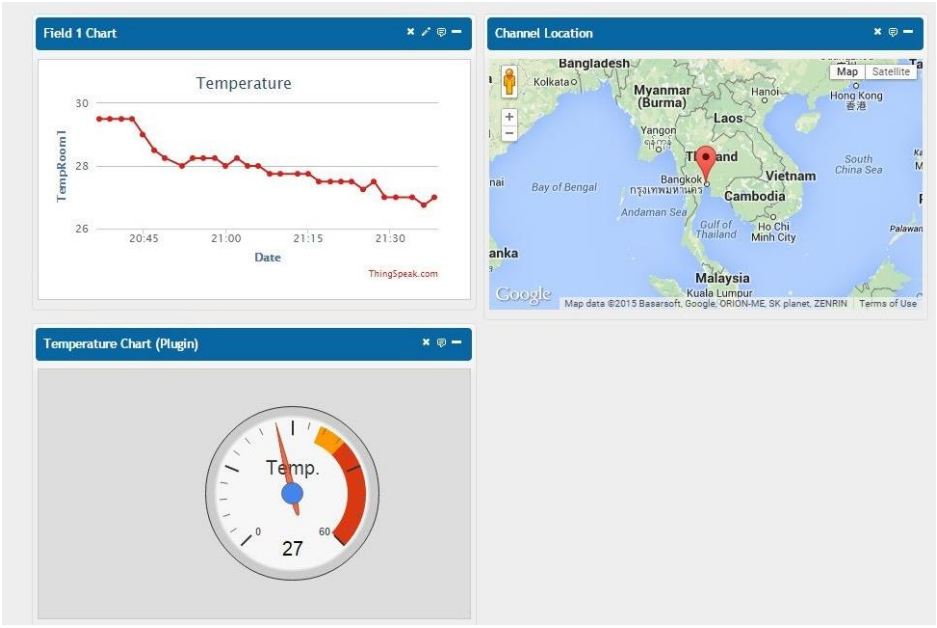


Ilustración 165: Interfaz de usuario de la plataforma ThingSpeak

Al ser de código abierto el soporte de ThingSpeak cuenta con un blog, foro, documentación y tutoriales que la comunidad de internet puede editar para mejorar su estabilidad de la plataforma o agregar funciones nuevas.

El uso de esta plataforma es gratuito, pero contiene limitaciones como, por ejemplo: el envío de datos es lento y solo hay un máximo de datos que se pueden almacenar en el servidor, también en la cantidad de dispositivos que se pueden conectar a una sola cuenta, pero para el uso educativo nos sirve para adentrarnos a el mundo de estos servicios en la nube, si se desea quitar esas limitaciones se puede contratar un plan por año, el cual va de 95 a 650 dólares por año, dependiendo si es para uso personal, educativo o por una institución.

Material para el desarrollo de la práctica:

- Placa ESP32
- Protoboard
- Jumpers
- Sensor BMP280
- Cable micro-USB

Desarrollo:

Antes de la realización del primer ejercicio primero configuraremos un nuevo canal en la plataforma de TingSpek

Nota: Primero crear una cuenta en dicha plataforma antes de continuar

En la página principal, parte superior nos dirigimos a la pestaña de “channels” y seleccionamos la opción de “my channels”

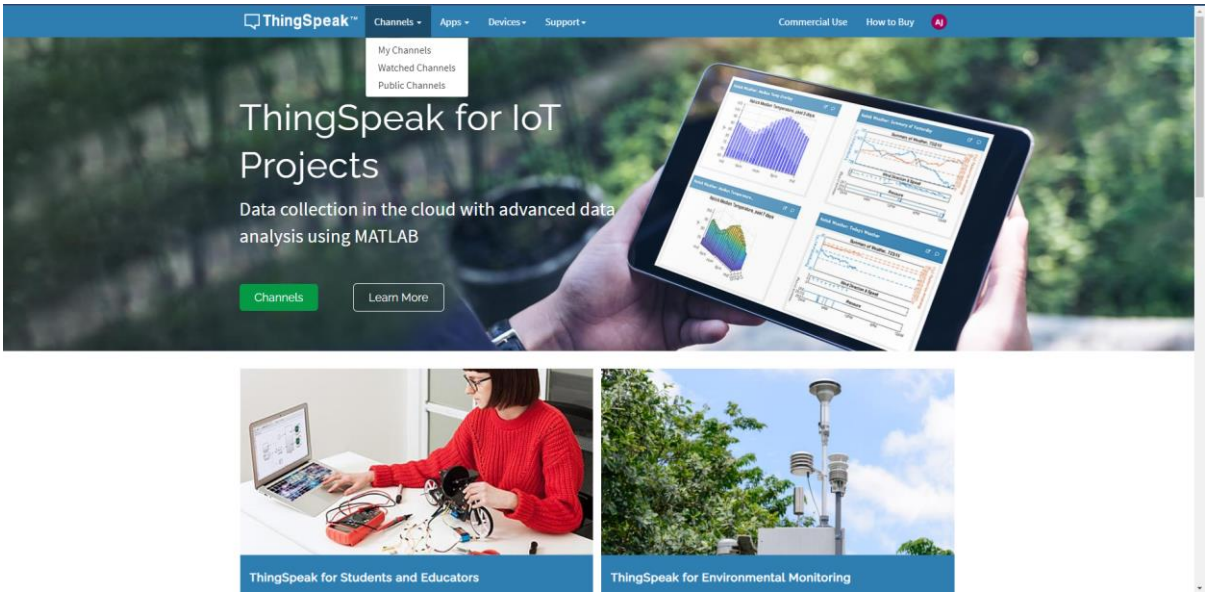


Ilustración 166: Página principal de ThingSpeak

Nos dirigirá al panel de nuestros canales como no tenemos ninguno le damos clic en "New Channels"

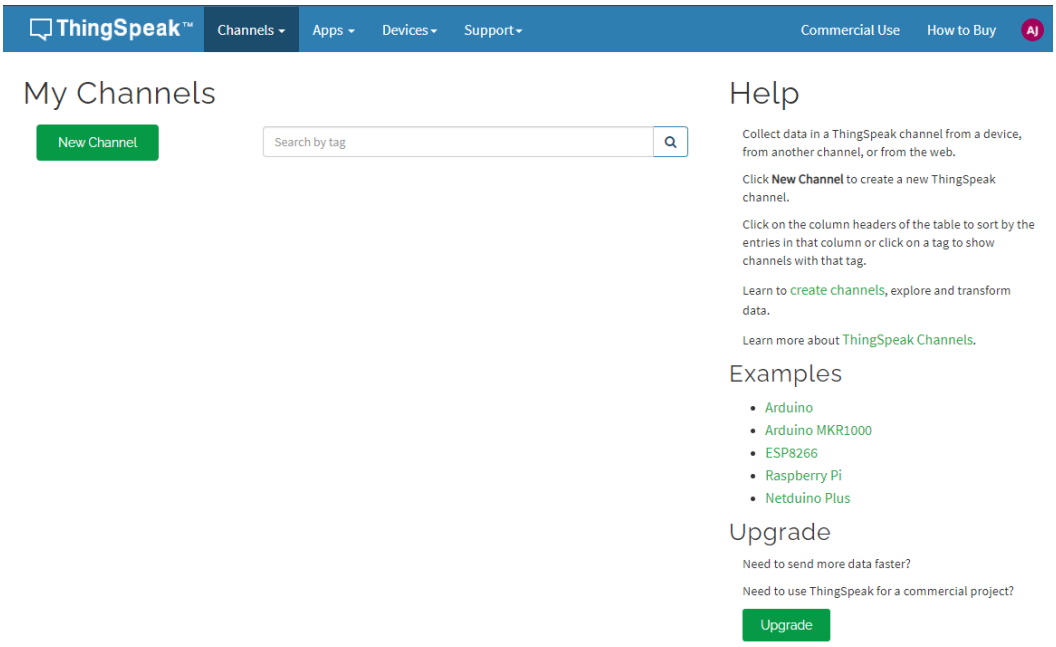


Ilustración 167: Creación de nuevo "Canal"

Después nos mandara al panel de configuración del canal

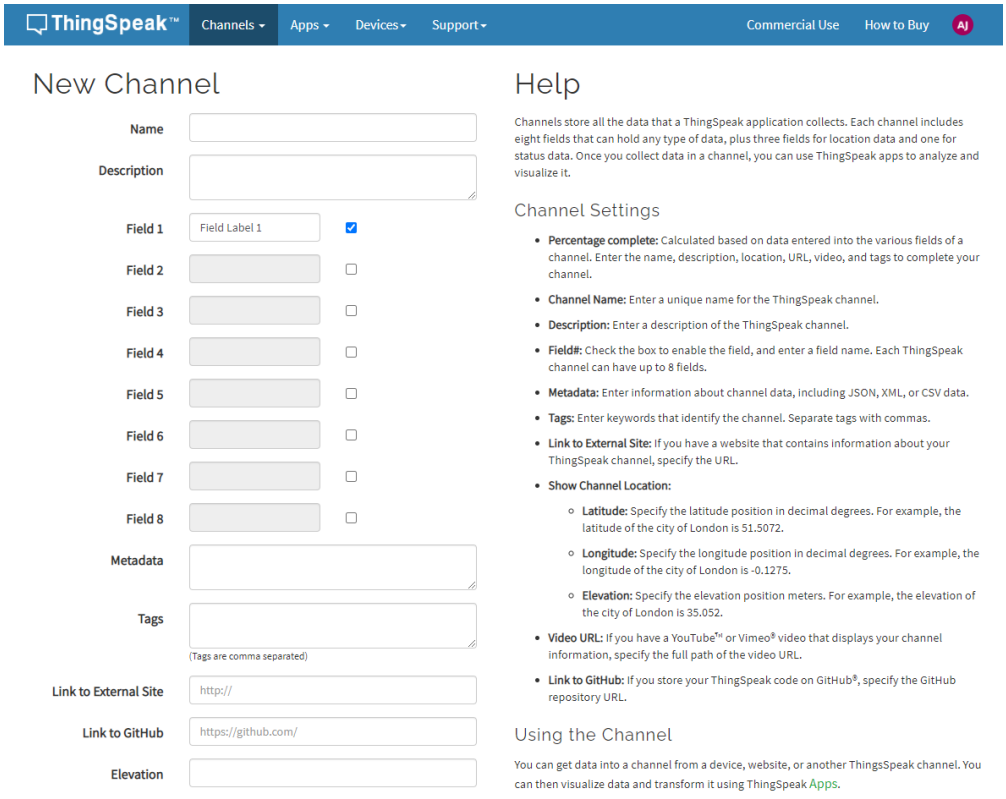


Ilustración 168: Configuración del "Canal"

En el nombre le ponemos “BMP280, en descripción ponemos “Sensor BMP280” y en los recuadros de Field, seleccionamos hasta el número 3 y en cada uno los nombramos como “Temperatura”, “Presión” y “Altitud” respectivamente

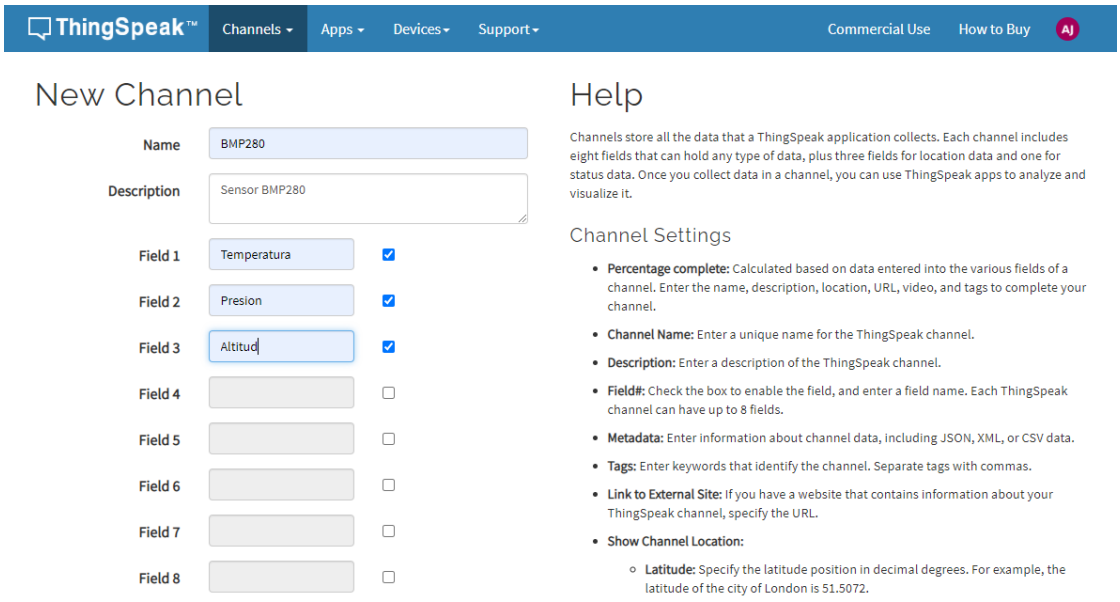


Ilustración 169: Configuración del "Canal" para el sensor BMP280

Después nos vamos al final de la página y damos clic en “Save Channel”

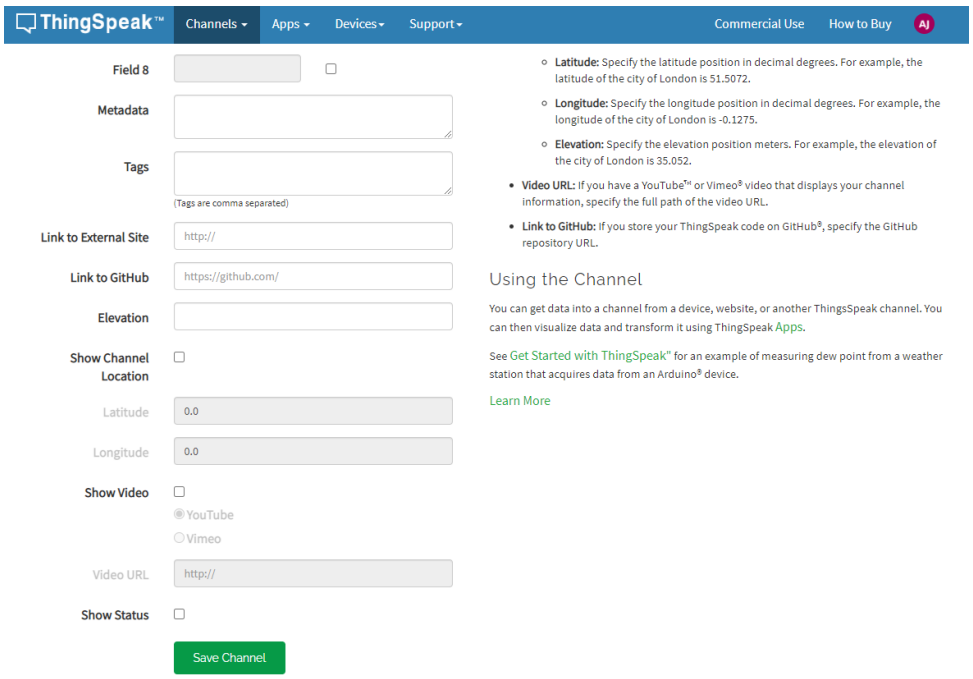


Ilustración 170: Guardado de configuración del "Canal" creado

Se mostrarán el canal creado con varias gráficas, en cada una le damos clic en la opción del recuadro rojo.

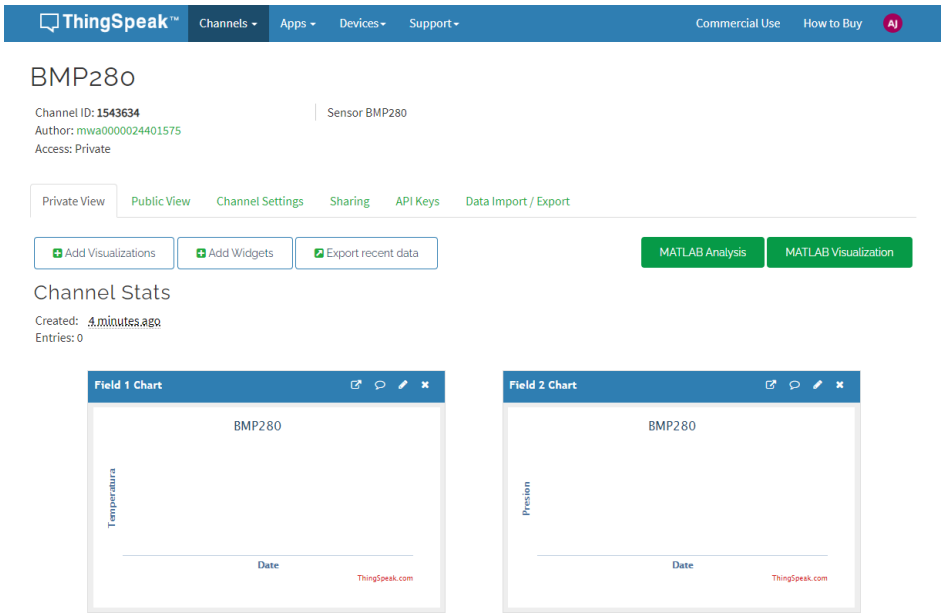


Ilustración 171: Interfaz del "canal" del sensor BMP280

Cambiamos el título a “BMP280 Temperatura”, “BMP280 Presión” y “BMP280 Altitud” respectivamente y le damos clic en “save”

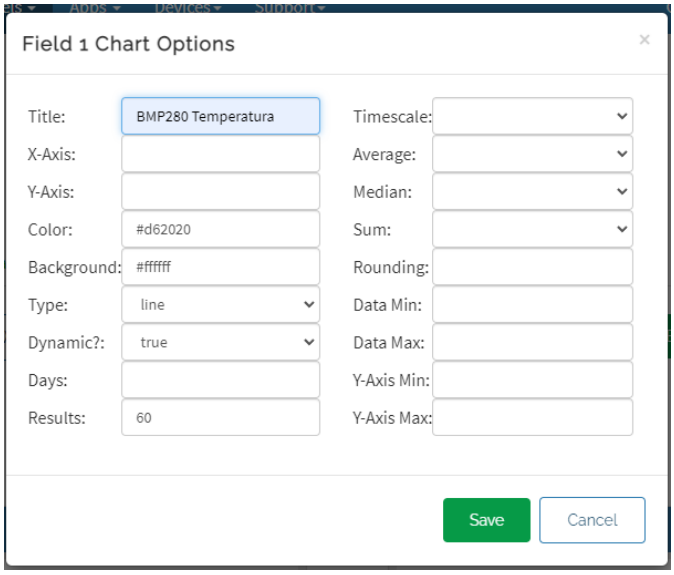


Ilustración 172: Configuración de cada grafica para el valor de cada lectura del sensor BMP280

Deben de quedar como se muestra en la siguiente imagen

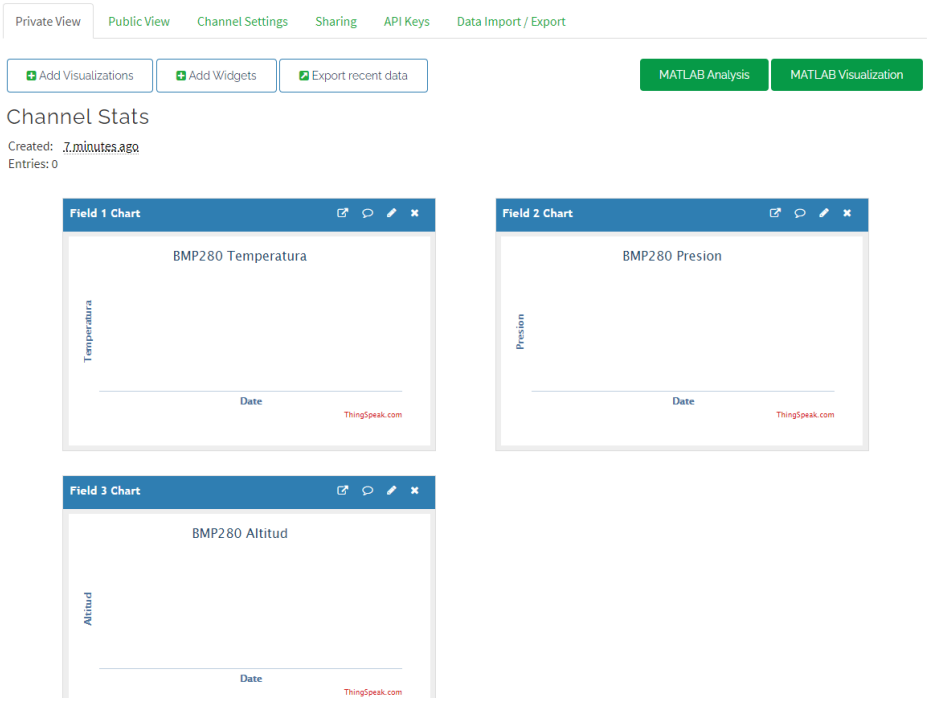


Ilustración 173: Interfaz de las gráficas de las lecturas del sensor BMP280

Necesitaremos una APIkey para que el ESP32 pueda comunicarse con los servidores de ThingSpeak, vamos a la pestaña de “APIkeys”

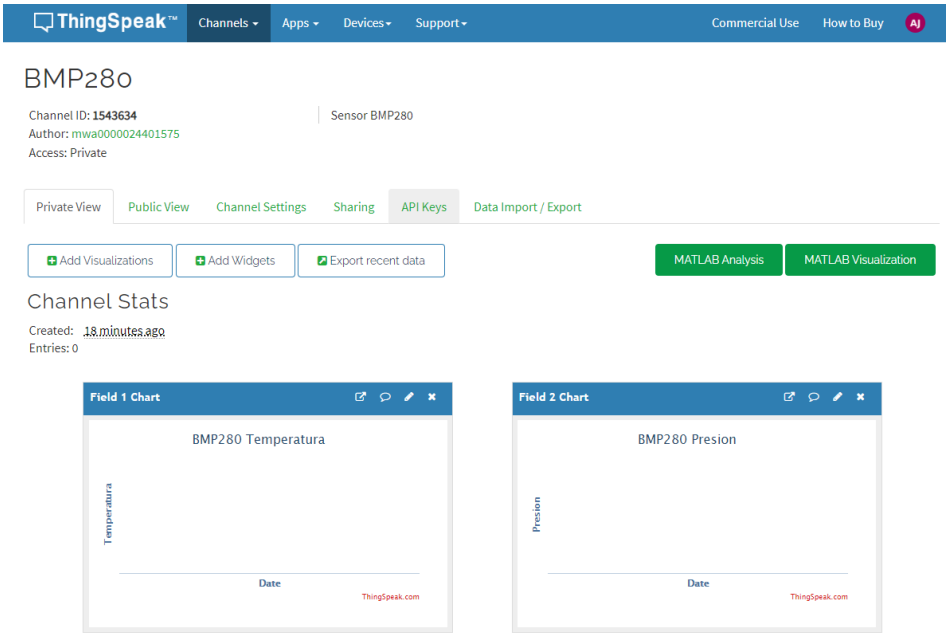


Ilustración 174: Obtención de la APIkey

Nos dirigirá a nuestra APIKey

Nota: No compartir esta key con ningún otro usuario por seguridad

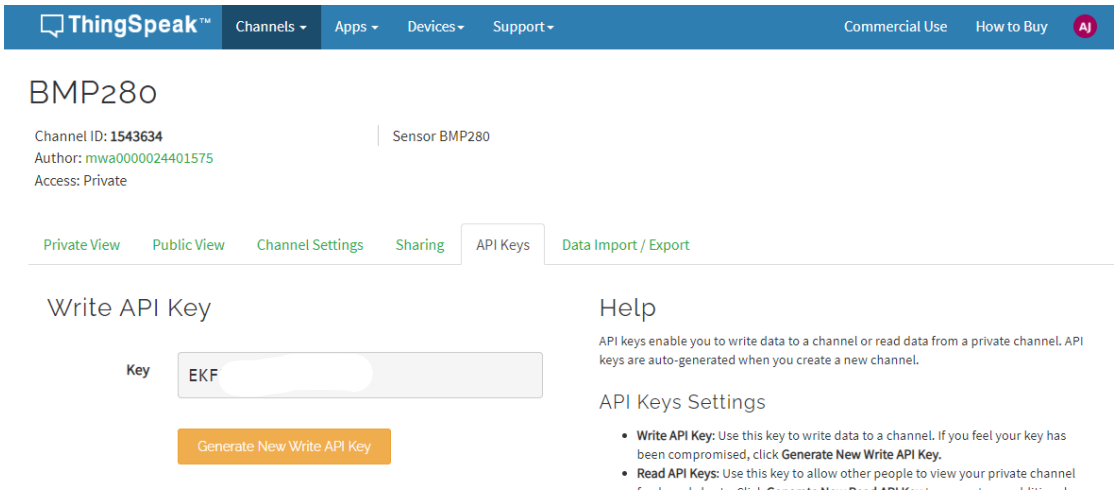


Ilustración 175: APIkey que genera la plataforma de ThingSpeak

Después abrimos al IDE de Arduino y descargamos la librería de “ThingSpeak”, seleccionamos la pestaña de “programa” después en “incluir librería” y seleccionamos “Administrador de Bibliotecas”

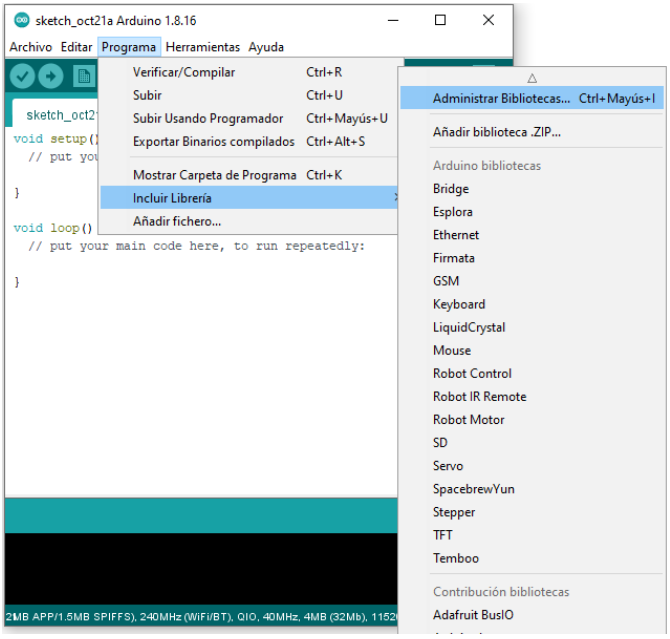


Ilustración 176: Menú para abrir la ventana de "Administrador de bibliotecas"

Se abrirá una nueva ventana, en el cuadro de búsqueda escribimos “ThingSpeak” e instalamos la librería marcada en el recuadro rojo

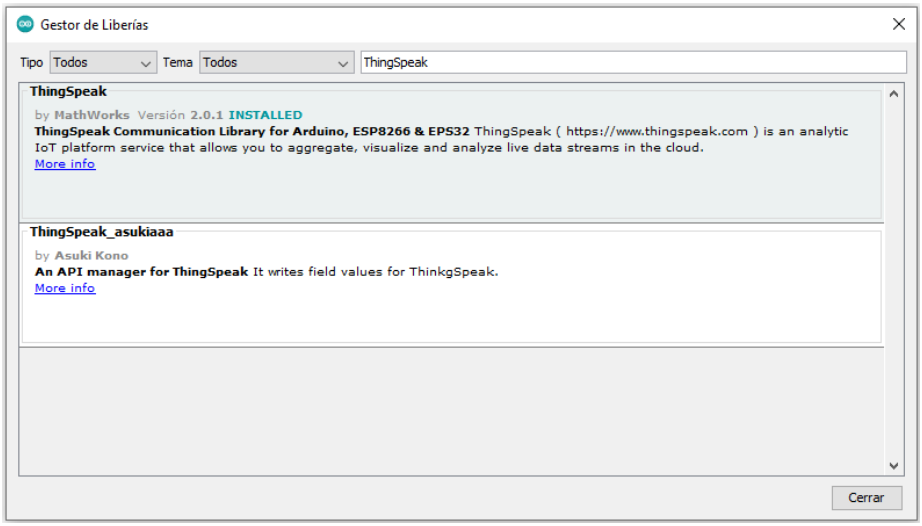


Ilustración 177: Ventana del "Gestor de Librerias"

Esperamos unos segundos y reiniciamos el IDE de Arduino

Ejercicio 1 ThingsSpeak:

Para hacer la comunicación del ESP32 con la plataforma ThingSpeak utilizaremos el siguiente código:

```
//Incluimos las librerías a usar
#include <Wi-Fi.h> //librería del Wi-Fi del ESP32
#include <Wire.h> //Librería para la comunicación I2C
#include "ThingSpeak.h" //Librería de la plataforma
ThingSpeak
#include <Adafruit_BMP280.h> //Librería del sensor BMP280
#define SEALEVELPRESSURE_HPA (1013.2) //Valor de la presión
en HPA de la CDMX

const char* ssid = "";   // SSID de la red Wi-Fi (Escriba el
SSID de su red entre las comillas)
const char* password = "";   // Contraseña de la red Wi-Fi
(Escriba la contraseña de la red Wi-Fi entre las comillas)

Wi-FiClient  client; //Inicializamos en Wi-Fi

unsigned long myChannelNumber = 1;  //Canal de escucha de
ThingSpeak (Reemplaze el numero por el canal de escucha que
desee enviar los datos del sensor)
const char * myWriteAPIKey = "XXXXXXXXXXXXX"; //APIkey de
nuestro perfil en ThingSpeak (Reemplaze las “XXXXX” por su
APIkey)

// Variables de tiempo para mandar la información cada cierto
tiempo
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;

//Variables donde guardaremos las lecturas del sensor
float temperatura;
float presion;
float altitud;

// creamos el objeto del sensor BMP280 y lo llamamos bmp
Adafruit_BMP280 bmp;
//Subprograma de la configuración del sensor BMP280
void initBMP(){

   if (!bmp.begin(0x76,0x58)) {
    Serial.println(F("No se pudo encontrar el sensor BMP280,
revise la conexion!"));
```

```
    while (1) delay(10);
  }
  //Configuraciones por defecto del sensor
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Modo
de operación */
                  Adafruit_BMP280::SAMPLING_X2,     /* Sobre
muestreo de temperatura */
                  Adafruit_BMP280::SAMPLING_X16,    /* Sobre
muestreo de presión */
                  Adafruit_BMP280::FILTER_X16,      /*
Filtración */
                  Adafruit_BMP280::STANDBY_MS_500); /* Modo
de espera */

}

void setup() {
  Serial.begin(115200);  //Iniciamos el monitor serial
  Wi-Fi.mode(WI-FI_STA);   //Configuramos el Wi-Fi del ESP32
en modo STA
  ThingSpeak.begin(client);  // Iniciamos ThingSpeak

  //Conexcion a la red Wi-Fi
  Wi-Fi.begin(ssid, password);
  Serial.print("Conectando..");
    while (Wi-Fi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }
    Serial.println("");
    Serial.println("Conectado a la red Wi-Fi");

 //Iniciamos al sensor con las configuraciones
  initBMP();
}

void loop() {


  if ((millis() - lastTime) > timerDelay) {

    // Leemos la temperatura del sensor
    temperatura = bmp.readTemperature();
    Serial.print("Temperature (°C): ");
    Serial.println(temperatura);

    // Leemos la presión del sensor
```

```
    presion = (bmp.readPressure() / 100.0F);
    Serial.print("Presión (hPa): ");
    Serial.println(presion);

    // Leemos la altitud del sensor
    altitud = (bmp.readAltitude(SEALEVELPRESSURE_HPA));
    Serial.print("Altitud (m): ");
    Serial.println(altitud);

    // Configuramos los campos en donde se mostrarán las
variables en la plataforma ThingSpeak
    ThingSpeak.setField(1, temperatura);
    ThingSpeak.setField(2, presion);
    ThingSpeak.setField(3, altitud);

    //Mandamos la información a ThingSpeak (Podemos mandar
hasta un máximo de 8 variables)
    int x = ThingSpeak.writeFields(myChannelNumber,
myWriteAPIKey);

    //Checamos si la información se envió correctamente
    if(x == 200){
      Serial.println("Canal actualizado correctamente ");
    }
     //Si no se envio, muestra un mensaje de error con el
código en HTTP
    else{
      Serial.println("Problema al actualizar canal. HTTP
error code " + String(x));
    }
    lastTime = millis();
  }
}
```

Antes de cargar el código al ESP32 armaremos el siguiente diagrama:

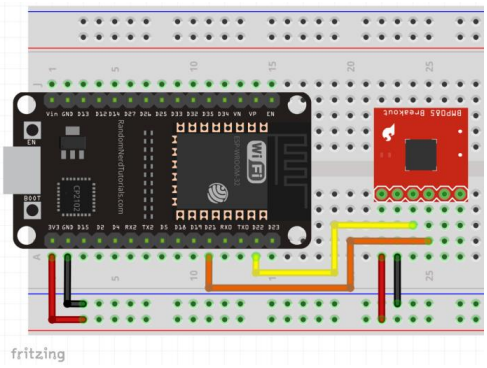


Ilustración 178: Diagrama del Ejercicio 1

Después cargaremos el código al ESP32, abrimos el monitor serial y presionamos el botón “EN” del ESP32 y esperamos unos segundos a que nos muestre que los datos fueron enviados exitosamente

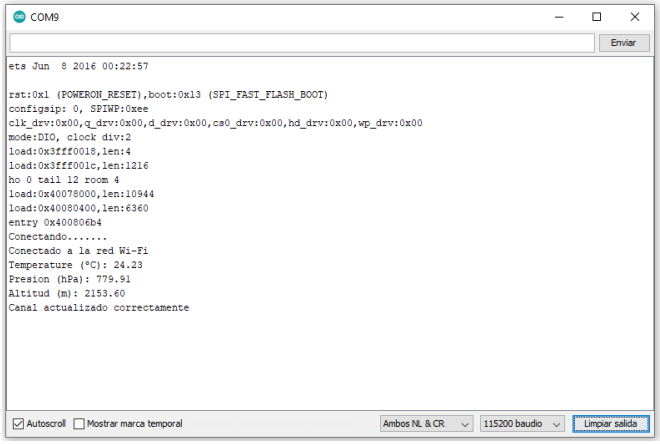


Ilustración 179: Comprobación del envío de datos del ESP32 a la plataforma ThingSpeak

Nota: Si recibimos el mensaje de “Problema al actualizar el canal” verifique que el “canal” no haya cambiado de APIkey o que lo haya escrito incorrectamente, cada canal tiene una key diferente.

Después nos dirigimos en la página de ThingSpeak y nos dirigimos al canal que creamos al inicio, se deberán de mostrar datos del sensor en las gráficas como se muestra en la siguiente imagen:

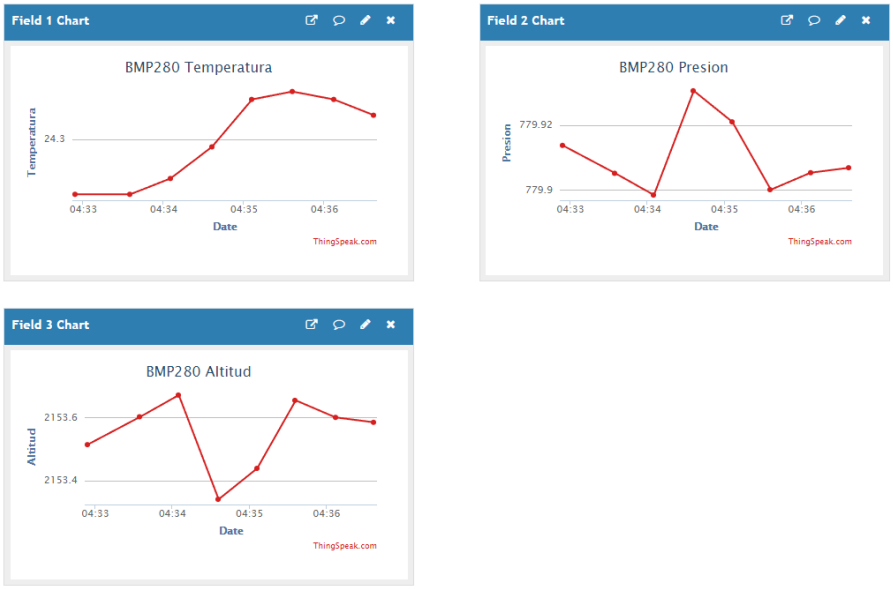


Ilustración 180: Graficas con los datos del sensor BMP280

Los datos pueden tardar unos segundos en verse reflejados en las gráficas desde que se mandó la información del ESP32.

Conclusiones

El presente trabajo es una propuesta que se planteó con el objetivo de tener un manual de prácticas a modo de guía para complementar la enseñanza del Laboratorio de Diseño de Sistemas con Microprocesadores basándonos en la placa de desarrollo ESP32, al ser el primer trabajo para dicho laboratorio se pueden presentar mejoras o cambios para una óptima realización de las practicas.

Fomentamos el uso de nuevas placas de desarrollo, en este caso la tarjeta ESP32 para futuros proyectos de electrónica enfocándose en una amplia gama de posibilidades como por ejemplo en el ámbito de IoT, por sus conectividades inalámbricas, su facilidad de programación con el IDE de Arduino, mejores características a comparación de otras placas de desarrollo y su bajo coste.

Se pudo observar cuales son las ventajas y desventajas del chip del ESP32, conocimos una manera en la que se puede programar con un entorno de programación que es uno de los más populares dentro del estudio de la electrónica, respaldado por una gran comunidad que sigue dando ideas nuevas y soluciones al momento de usar el software, talvez el principal problema es que cierta información solo se encuentra en inglés o en foros de habla inglesa, pero esto no tiene que ser un impedimento para usar esta tarjeta de desarrollo, incluso puede ser una manera en que más personas tengan el interés de estudiar el idioma y así aprender otras maneras de como programar el ESP32.

Utilizamos un entorno de programación que es comúnmente usado cuando se quiere iniciar en el ámbito de la programación de microcontroladores, el IDE de Arduino provee de una comunidad la cual cada día brindan soluciones o información acerca de cómo usar el software, aunque el IDE no está específicamente desarrollado para que se programen diferentes tarjetas de desarrollo diferentes a la familia Arduino y la empresa Espressif Systems dieron la posibilidad de que pueda ser posible usarlo para programar el ESP32.

Trabajo futuro

Este trabajo presenta un inicio para que, si un futuro otro alumno de la facultad quiera realizar prácticas para el laboratorio o un profesor que imparte la materia, tengan este manual como base o una guía para así complementar la enseñanza de la asignatura.

También sirve como base para que se analice si el usar la placa de desarrollo del ESP32 es viable para el laboratorio, observando las ventajas y desventajas de usarla, para que, si lo requiere el laboratorio o el profesor, se cambie o desarrollen otras prácticas usando otras placas de desarrollo existentes en el mercado o bien, cambiando el entorno de programación a los diferentes que hay disponibles y no solo usar el IDE de Arduino.

Se pueden presentar deficiencias en algunos ejercicios o que resultan relativamente sencillos de hacer, esto se dio por lo sucedido en los años 2020 a 2022 que por una pandemia global se tuvo que quedar en aislamiento total y, por ende, se dejó de ir a las aulas, por ese motivo los ejercicios se tuvieron que realizar de una manera no tan complicada y explicados de una manera que resultara sencillo entender para el alumno.

Por ese mismo motivo de la pandemia se tuvo que descartar la idea de poner a prueba este manual de prácticas en un grupo de alumnos y que a base de ello, se pudiera dar una mejor retroalimentación para dar una mejor propuesta.

Basándome en las opiniones, estudios y experiencia propia al tomar clases en línea, se llega a una conclusión de que estos temas al ser prácticos deben de tratarse de una manera presencial o bien, de una manera en que el alumno pueda realizar los ejercicios de manera física. Por ello, esta propuesta de prácticas considero en mi opinión personal que debe de ser mejorado o adaptado a un entorno en donde las practicas se puedan realizar de una manera ya sea presencial o hibrida. [75]

Referencias de imágenes

Ilustración 1: SoC del ESP32. Recuperado el 8 de enero de 2022, de https://uy.mouser.com/new/espressif/espressif-esp32-soc/

Ilustración 2: Nomenclatura de los diferentes Soc´s del ESP32. Recuperado el 9 de enero de 2022, de https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Ilustración 3: Modulo ESP32-WROOM-32D. Recuperado el 9 de enero de 2022, de https://docs.espressif.com/projects/esp-idf/en/v4.3.1/esp32/hw-reference/modules-and-boards.html

Ilustración 4: Modulo ESP32-WROOM-32. Recuperado el 9 de enero de 2022, de https://docs.espressif.com/projects/esp-idf/en/v4.3.1/esp32/hw-reference/modules-and-boards.html

Ilustración 5: Modulo ESP32-WROOVER. Recuperado el 9 de enero de 2022, de https://docs.espressif.com/projects/esp-idf/en/v4.3.1/esp32/hw-reference/modules-and-boards.html

Ilustración 6: Placa de desarrollo ESP32 CAM. Recuperado el 9 de enero de 2022, de https://hetpro-store.com/esp32-cam/

Ilustración 7: Placa de desarrollo ESP32-DevKitC. Recuperado el 9 de enero de 2022, de https://docs.espressif.com/projects/esp-idf/en/v4.3.1/esp32/hw-reference/modules-and-boards-previous.html#esp-modules-and-boards-esp32-devkitc-v2

Ilustración 8: Placa de desarrollo HUZZAH32. Recuperado el 9 de enero de 2022, de https://templates.blakadder.com/adafruit_HUZZAH32.html

Ilustración 9: Placa de desarrollo Node-MCU-32S. Recuperado el 9 de enero de 2022, de https://www.amazon.com.mx/Tresd-NodeMCU-32-Bluetooth-Compatible-arduino/dp/B08DFKCF6H

Ilustración 10: Placa de desarrollo ESP32 devkit v1. Recuperado el 9 de enero de 2022, de https://shopee.com.mx/ESP32-CAM-Wi-Fi-Development-Board-ESP-32S-Tablero-De-Desarrollo-FT232RL-FTDI-Alambre-De-Puente-i.653837799.14143440188

Referencias

[1] B. Kotiyal, I. Baig, M. Muzamil, y S. Dalvi, “Home automation using arduino Wi-Fi module ESP8266”, may 2016, Consultado: el 3 de mayo de 2022. [En línea]. Disponible en: http://localhost:8080/xmlui/handle/123456789/1558

[2] F. Biendicho Lletí, “Comunicación Bluetooth entre Arduino UNO y Android aplicado a un detector de mentiras”, Proyecto/Trabajo fin de carrera/grado, Universitat Politècnica de València, 2015. Consultado: el 3 de mayo de 2022. [En línea]. Disponible en: https://riunet.upv.es/handle/10251/57549

[3] R. Renteria, R. Ruelas, y G. Ochoa, “Módulo ESP8266 y sus aplicaciones en el internet de las cosas”, vol. 1, núm. 2, pp. 24–36, ago. 2017.

[4] M. Schwartz, *Internet of Things with ESP8266*. Packt Publishing Ltd, 2016.

[5] M. I. Álvarez Bautista, G. Seseña Pascasio, y U. M. Peñuelas Rivas, “Creación de biblioteca Wi-Fi ESP para microcontroladores PIC”, Universidad Nacional Autónoma de México, Ciudad Universitaria, Cd. Mx., 2020. Consultado: el 24 de enero de 2022. [En línea]. Disponible en: https://tesiunam.dgb.unam.mx/F/JSVEXMB5VGXCM9TVX4M82I718R17X6RCREI4LYJ8M8F2J1931M-53604?func=full-set-set&set_number=212628&set_entry=000001&format=999

[6] E. Crespo, “Dispositivos Hardware IoT”, *Aprendiendo Arduino*, el 15 de octubre de 2019. https://aprendiendoarduino.wordpress.com/2019/10/15/dispositivos-hardware-iot-2/ (consultado el 5 de mayo de 2022).

[7] “Internet de las cosas: Protocolos de comunicación inalámbricos más comunes en IoT? - **Electrodaddy**”. https://www.electrodaddy.com/internet-de-las-cosas-protocolos-de-comunicacion-inalambricos-mas-comunes-iot/ (consultado el 5 de mayo de 2022).

[8] P. Bertoleti, *Proyectos com ESP32 y LoRa*. Editora NCB, 2019. [En línea]. Disponible en: https://books.google.es/books?id=Doi0DwAAQBAJ

[9] “System on a chip”, *Wikipedia*. el 1 de noviembre de 2021. Consultado: el 2 de noviembre de 2021. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=System_on_a_chip&oldid=1053086814

[10] Espressif Systems, “ESP32 Series Datasheet”. el 16 de julio de 2021. Consultado: el 10 de septiembre de 2021. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[11] Á. Benito Herranz, “Desarrollo de aplicaciones para IoT con el módulo ESP32”, Trabajo Fin de Grado, Universidad de Alcalá Escuela Politécnica Superior, 2019. Consultado: el 9 de enero de 2022. [En línea]. Disponible en: https://ebuah.uah.es/dspace/bitstream/handle/10017/35420/TFG_Benito_Herranz_2019.pdf?sequence=1&isAllowed=y

[12] “ESP32 Modules and Boards - ESP32 - — ESP-IDF Programming Guide v4.3.1 documentation”, *ESP-IDF Programming Guide*. https://docs.espressif.com/projects/esp-idf/en/v4.3.1/esp32/hw-reference/modules-and-boards.html (consultado el 9 de enero de 2022).

[13] Adafruit Industries, “Adafruit HUZZAH32 – ESP32 Feather Board”, *Adafruit*. https://www.adafruit.com/product/3405 (consultado el 9 de noviembre de 2021).

[14] I. Hübschmann, “ESP32 for IoT: A Complete Guide”, *Nabto*, el 28 de agosto de 2020. https://www.nabto.com/guide-to-iot-esp-32/ (consultado el 9 de noviembre de 2021).

[15] M. Fezari y A. Al Dahoud, “Integrated Development Environment ‘IDE’ For Arduino”, oct. 2018, [En línea]. Disponible en: https://www.researchgate.net/profile/Mohamed-Fezari-2/publication/328615543_Integrated_Development_Environment_IDE_For_Arduino/links/5bd8c6d24585150b2b9206df/Integrated-Development-Environment-IDE-For-Arduino.pdf

[16] Ó. Torrente Artero, *ARDUINO. Curso práctico de formación*. RC Libros, 2013. [En línea]. Disponible en: https://books.google.es/books?id=6cZhDmf7suQC

[17] Free Software Foundation, Inc., “¿Qué es el Software Libre? - Proyecto GNU - Free Software Foundation”, *El sistema operativo GNU*. https://www.gnu.org/philosophy/free-sw.es.html (consultado el 9 de septiembre de 2021).

[18] J. Beningo, “Cómo seleccionar y usar el módulo ESP32 con Wi-Fi/Bluetooth adecuado para una aplicación de IoT industrial”, *Digi-Key*, el 21 de enero de 2020. https://www.digikey.com.mx/es/articles/how-to-select-and-use-the-right-esp32-wi-fi-Bluetooth-module (consultado el 9 de septiembre de 2021).

[19] S. Balachandran, “General Purpose Input/Output (GPIO)”. Michigan State University College of Engineering, 2009. [En línea]. Disponible en: https://www.egr.msu.edu/classes/ece480/capstone/fall09/group03/AN_balachandran.pdf

[20] Texas Instruments, “TMS320C6452 DSP General-Purpose Input/Output (GPIO)”. 2007. Consultado: el 10 de septiembre de 2021. [En línea]. Disponible en: https://www.ti.com/lit/ug/spruf95/spruf95.pdf?ts=1631285180447

[21] “GPIO”, *Wikipedia, la enciclopedia libre*. el 7 de agosto de 2019. Consultado: el 10 de septiembre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=GPIO&oldid=118066807

[22] “Universal asynchronous receiver-transmitter”, *Wikipedia*. el 23 de agosto de 2021. Consultado: el 11 de septiembre de 2021. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver-transmitter&oldid=1040256325

[23] “C. Gordon Bell”, *Wikipedia, la enciclopedia libre*. el 1 de julio de 2020. Consultado: el 11 de septiembre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=C._Gordon_Bell&oldid=127395521

[24] C. G. Bell, J. C. Mudge, y J. E. McNamara, Eds., *Computer engineering: A dec view of hardware systems design*, vol. 4, 4 vols. Bedford, Mass: Digital Press, 1978. Consultado: el 11 de septiembre de 2021. [En línea]. Disponible en: http://bitsavers.org/pdf/dec/_Books/Bell-ComputerEngineering.pdf

[25] "PDP-1", *Wikipedia, la enciclopedia libre*. el 30 de enero de 2020. Consultado: el 11 de septiembre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=PDP-1&oldid=123184672

[26] "Chapter 20: Analog to Digital Conversion [Analog Devices Wiki]", *Analog Devices*, el 20 de enero de 2021. https://wiki.analog.com/university/courses/electronics/text/chapter-20 (consultado el 5 de octubre de 2021).

[27] Solectroshop, "¿Qué es PWM y cómo usarlo?", *Solectro*, el 26 de agosto de 2020. https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38 (consultado el 5 de octubre de 2021).

[28] "Sensor táctil", *HiSoUR Arte Cultura Historia*, el 7 de noviembre de 2018. https://www.hisour.com/es/tactile-sensor-42878/ (consultado el 27 de septiembre de 2021).

[29] "Sensors - Touch Sensors | Newark", *Newark*. https://mexico.newark.com/sensor-touch-sensor-cap-res-technology (consultado el 27 de septiembre de 2021).

[30] "ESP32 Capacitive Touch Sensor Pins with Arduino IDE | Random Nerd Tutorials", *Random Nerd Tutorials*, el 31 de mayo de 2019. https://randomnerdtutorials.com/esp32-touch-pins-arduino-ide/ (consultado el 27 de septiembre de 2021).

[31] E. Cocero Navarro y J. Díaz Bizarro, "Estudio de aplicaciones de Bluetooth para móviles de la serie S60 de Nokia". junio de 2007. Consultado: el 21 de octubre de 2021. [En línea]. Disponible en: https://eprints.ucm.es/id/eprint/8914/1/memoriaChatBT.pdf

[32] I. Puy, "Bluetooth". Hochschule Furtwangen University, el 5 de mayo de 2008. [En línea]. Disponible en: https://webuser.hs-furtwangen.de/~heindl/ebte-08ss-Bluetooth-Ingo-Puy-Crespo.pdf

[33] "Ericsson T39", *Wikipedia*. el 5 de junio de 2020. Consultado: el 7 de octubre de 2021. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Ericsson_T39&oldid=960838686

[34] C. Zahumenszky, "Ericsson T68m. Teléfonos con Historia IX", *Xataka*, el 24 de febrero de 2012. https://www.xataka.com/moviles/ericsson-t68m-telefonos-con-historia-ix (consultado el 7 de octubre de 2021).

[35] A. R. Castellano, "Bluetooth. Introducción a su Funcionamiento." Universidad Pontificia Comillas, 2012 de 2011. Consultado: el 7 de octubre de 2021. [En línea]. Disponible en: http://www.sistemamid.com/panel/uploads/biblioteca/1/619/640/641/3792.pdf

[36] "Harald Blåtand", *Wikipedia, la enciclopedia libre*. el 17 de agosto de 2021. Consultado: el 7 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Harald_Bl%C3%A5tand&oldid=137729219

[37] Espressif Systems, “ESP32 Bluetooth Architecture”. noviembre de 2019. Consultado: el 7 de octubre de 2021. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_Bluetooth_architecture_en.pdf

[38] E. Pau García, “Redes Wi-Fi, ¿Realmente se pueden proteger?”, Máster Universitario en seguridad de las TIC, Universidad Oberta de Catalunya, España, 2019. Consultado: el 7 de octubre de 2021. [En línea]. Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/95427/6/epauTFM0619memoria.pdf

[39] J. Pipa Huamán, “REDES INALÁMBRICAS. Introducción a las redes inalámbricas, ventajas y desventajas de las WI-FI, estándares inalámbricos, hardware inalámbrico, diseño de una Red Inalámbrica, instalación de una red Inalámbrica, configuración de Red Inalámbrica, software para redes inalámbricas, aplicaciones.”, Tesis de licenciatura, Universidad Nacional de Educación, Lima, Perú, 2019. [En línea]. Disponible en: http://200.60.81.165/bitstream/handle/20.500.14039/5004/Redes%20inal%C3%A1mbricas.pdf?sequence=1&isAllowed=y

[40] J. Danielsen Newth, “The History of Wi-Fi”, *Eye Networks*, el 12 de junio de 2019. https://eyenetworks.no/en/wi-fi-history/ (consultado el 7 de octubre de 2021).

[41] J. Thomas, “La historia del Wi-Fi”, *purple*, el 1 de mayo de 2019. https://purple.ai/es/blogs/la-historia-del-Wi-Fi/ (consultado el 7 de octubre de 2021).

[42] The Editors of Encyclopaedia Britannica, “Wi-Fi | networking technology”, *Encyclopedia Britannica*, el 3 de febrero de 2022. https://www.britannica.com/technology/Wi-Fi (consultado el 7 de octubre de 2021).

[43] “Wi-Fi - ESP32 - — ESP-IDF Programming Guide v4.3.1 documentation”, *ESP-IDF Programming Guide*. https://docs.espressif.com/projects/esp-idf/en/v4.3.1/esp32/api-reference/network/esp_Wi-Fi.html (consultado el 7 de octubre de 2021).

[44] “ESP32 Bluetooth Classic with Arduino IDE - Getting Started | Random Nerd Tutorials”, *Random Nerd Tutorials*, el 10 de mayo de 2019. https://randomnerdtutorials.com/esp32-Bluetooth-classic-arduino-ide/ (consultado el 7 de octubre de 2021).

[45] “ESP32 Useful Wi-Fi Library Functions (Arduino IDE) | Random Nerd Tutorials”, *Random Nerd Tutorials*, el 12 de febrero de 2021. https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/ (consultado el 7 de octubre de 2021).

[46] “Central processing unit”, *Wikipedia*. el 9 de octubre de 2021. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Central_processing_unit&oldid=1049000573

[47] “Unidad central de procesamiento”, *Wikipedia, la enciclopedia libre*. el 22 de septiembre de 2021. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_central_de_procesamiento&oldid=138514148

[48] S. Muñoz, “CPU vs. microprocesador: ¿Cuáles son las diferencias?”, *ComputerWeekly.es*, el 16 de julio de 2020. https://www.computerweekly.com/es/consejo/CPU-vs-microprocesador-Cuales-son-las-diferencias (consultado el 9 de octubre de 2021).

[49] J. Velasco, “Historia de la Tecnología: Intel 8080”, *Hipertextual*, el 23 de agosto de 2012. http://hipertextual.com/2012/08/historia-intel-8080 (consultado el 9 de octubre de 2021).

[50] “Microprocesador”, *Wikipedia, la enciclopedia libre*. el 4 de octubre de 2021. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Microprocesador&oldid=138763171

[51] J. Velasco, “Historia de la Tecnología: Gun Fight”, *Hipertextual*, el 20 de julio de 2012. http://hipertextual.com/2012/07/historia-de-la-tecnologia-gun-fight (consultado el 9 de octubre de 2021).

[52] “Procesador multinúcleo”, *Wikipedia, la enciclopedia libre*. el 7 de mayo de 2021. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Procesador_multin%C3%BAcleo&oldid=135373445

[53] “Dual-core”, *Wikipedia, la enciclopedia libre*. el 16 de febrero de 2021. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Dual-core&oldid=133266445

[54] “Procesadores Dual Core | Observatorio Tecnológico”. http://recursostic.educacion.es/observatorio/web/eu/equipamiento-tecnologico/hardware/267-angel-maria-de-dios-roso (consultado el 9 de octubre de 2021).

[55] B. Gómez, “¿Qué son los núcleos de un procesador? ¿cómo funciona? ⭐”, *Profesional review*, el 13 de febrero de 2021. https://www.profesionalreview.com/2021/02/13/nucleo-procesador/ (consultado el 9 de octubre de 2021).

[56] “ESP32 Dual Core with Arduino IDE | Random Nerd Tutorials”, *Random Nerd Tutorials*, el 4 de octubre de 2018. https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/ (consultado el 9 de octubre de 2021).

[57] G. Herrero González, “Task (Tarea)”, *guillehg*. http://www.guillehg.com/index.php?option=com_content&view=article&id=68&Itemid=705 (consultado el 9 de octubre de 2021).

[58] L. Llamas, “Cómo usar FreeRTOS en Arduino”, *Luis LLamas*, el 21 de diciembre de 2021. https://www.luisllamas.es/como-usar-freertos-en-arduino/ (consultado el 20 de enero de 2022).

[59] Amazon Web Services, Inc, “FreeRTOS - Guía del usuario”. Consultado: el 20 de enero de 2022. [En línea]. Disponible en: https://docs.aws.amazon.com/es_es/freertos/latest/userguide/freertos-ug.pdf

[60] L. Thomas, “Digital humidity and temperature sensor AM2302”. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

[61] “Sensor de temperatura y humedad relativa DHT22 (AM2302)”, *Naylamp Mechatronics - Perú*. https://naylampmechatronics.com/sensores-temperatura-y-humedad/58-sensor-de-temperatura-y-humedad-relativa-dht22-am2302.html (consultado el 9 de octubre de 2021).

[62] Thomas Liu, “Capacitive-type humidity and temperature module/sensor”. Consultado: el 9 de octubre de 2021. [En línea]. Disponible en: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf

[63] “Sensor de presión BMP280”, *Naylamp Mechatronics - Perú*. https://naylampmechatronics.com/sensores-posicion-inerciales-gps/358-sensor-de-presion-bmp280.html (consultado el 9 de octubre de 2021).

[64] “ESP32 with DHT11/DHT22 Temperature and Humidity Sensor using Arduino IDE | Random Nerd Tutorials”, *Random Nerd Tutorials*, el 25 de abril de 2019. https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/ (consultado el 11 de octubre de 2021).

[65] “ESP32 with HC-SR04 Ultrasonic Sensor with Arduino IDE | Random Nerd Tutorials”, *Random Nerd Tutorials*, el 20 de julio de 2021. https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino/ (consultado el 11 de octubre de 2021).

[66] D. Carrasco, “ESP32: Empezando a usar el Bluetooth (SPP)”, *ElectroSoftCloud*, el 11 de abril de 2021. https://www.electrosoftcloud.com/esp32-empezando-a-usar-el-Bluetooth-spp/ (consultado el 13 de octubre de 2021).

[67] “Telegram”, *Wikipedia, la enciclopedia libre*. el 29 de marzo de 2022. Consultado: el 24 de enero de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Telegram&oldid=142578270

[68] “MTProto”, *Wikipedia, la enciclopedia libre*. el 22 de julio de 2021. Consultado: el 13 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=MTProto&oldid=137183896

[69] “Bot”, *Wikipedia, la enciclopedia libre*. el 7 de octubre de 2021. Consultado: el 13 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Bot&oldid=138845296

[70] “Telegram Bot API”, *Wikipedia, la enciclopedia libre*. el 11 de marzo de 2021. Consultado: el 13 de octubre de 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Telegram_Bot_API&oldid=133893888

[71] M. J. Quesada García, “BOT PARA TELEGRAM QUE EJECUTE AVENTURAS CONVERSACIONALES”, Trabajo Fin de Grado, UNIVERSIDAD DE JAÉN, España, 2017. Consultado: el 13 de octubre de 2021. [En línea]. Disponible en: https://tauja.ujaen.es/bitstream/10953.1/5944/1/TFG_Garcia-Quesada_Manuel-Jesus.pdf

[72] “Modelo vista controlador (MVC)”, *Universidad de Alicante*. https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html (consultado el 21 de octubre de 2021).

[73] A. González Fong, “Sistema de comunicación remoto para medir en tiempo real la temperatura utilizando sensores de fibra óptica.”, Reporte de Residencia, Tecnológico Nacional De México, México, 2019. [En línea]. Disponible en: http://repositoriodigital.tuxtla.tecnm.mx/xmlui/bitstream/handle/123456789/2249/MDRPIEL2018037.pdf?sequence=1&isAllowed=y

[74] F. D. Salgado Castillo y D. S. Coello Moncayo, “Prototipo de monitoreo ambiental aplicando el Internet de las cosas con Arduino y Cloud Computing”, Tesis de licenciatura, Universidad del Azuay, Ecuador, 2015. [En línea]. Disponible en: https://dspace.uazuay.edu.ec/bitstream/datos/5052/1/11491.pdf

[75] M. A. Lovón Cueva, “Repercusiones de las clases virtuales en los estudiantes universitarios en el contexto de la cuarentena por COVID19: El caso de la PUCP”, *Sep. 2020*, vol. 9, agosto de 2020. Consultado: el 3 de mayo de 2022. [En línea]. Disponible en: https://revistas.usil.edu.pe/index.php/pyr/article/download/588/1086