UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Alternativa de comunicación inalámbrica para una aplicación mecatrónica con microcontroladores PIC

FI- UNAM

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERA EN COMPUTACIÓN

PRESENTA:

Lidia Viridiana Amador González

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECATRÓNICO

PRESENTA:

Gianni Xavier Jacobo Hidalgo

DIRECTOR DE TESIS: Ing. María Eugenia Macías Ríos

2013.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos

A mis padres Crescencia González y Javier Amador, quienes han dado todo por ayudarme a alcanzar mis metas además de llenar mi vida de amor, alegría y disciplina.

A mis hermanos Javier y Jennifer por demostrarme lo importante que es la familia y llenarme de alegría todos los días de sus vidas.

A mis pequeños primos Emilio, Nidia, Jesús, Mónica, Lourdes y Ángel por recordarme con su inocencia el verdadero valor de la vida.

A mis primas Evelin y Zuliquey por ser mis hermanas postizas y amigas de toda la vida.

A mi abuelo Emilio por su disciplina que me guió a seguir por el camino correcto.

A mis abuelitas Nieves y Glafira por consentirme todo el tiempo.

A mis tíos, Pilar, David, Guillermina, Georgina, Martha, Miguel y Rodrigo por alentarme a seguir.

A mis tíos, María de los Ángeles y Romeo por todo su apoyo y por demostrarme que todo esfuerzo tiene sus recompensas.

A Gianni Xavier por ser mi compañero, amigo y novio, quien me recuerda todo el tiempo que soy capaz de lograrlo todo, por brindarme todo su cariño y comprensión, por ser coautor de este trabajo que significa el punto final de esta etapa de nuestras vidas y a la vez la primera de las grandes cosas que haremos juntos.

A mis mosqueteros Bruno, Jorge y Miguel por llenarme de alegría durante mi estadía en la Facultad, nuestra segunda casa.

A mis amigos Guillermo, Rodrigo, Nayely, Erika, Verónica y Fernando por brindarme sus consejos y apoyo.

A Maru por todo lo que me ha enseñado y por brindarme su amistad tan desinteresadamente.

A Aldo, Magda, Rafa, Gaby, por compartir su conocimiento conmigo.

Y a todas las personas que me apoyaron y que no mencioné.

Mi más sincero agradecimiento.

Viri.

A mi mamá, Laura Hidalgo Justo… Una madre maravillosa, y una mujer excepcional

A mi padre, Javier Jacobo

A mis tíos, Sandra Hidalgo y Brian Ehlke: sin sus diarios apoyos, jamás hubiera podido realizar este trabajo

A mi tío Gustavo Hidalgo, por toda su enorme paciencia y su gran comprensión

A mis abuelos, Bertha Justo, Taurina Rosas, Juan Hidalgo y Miguel Jacobo, de quienes soy un profundo admirador (abuelos, ¡quisiera tanto ser como ustedes!)

A Lidia Viridiana y su familia, a quienes debo infinidad de favores

A los doctores Alejandro Lara y Gaudencio Paz, por todo el material que desinteresadamente han puesto a mi alcance

A todas las personas que alegraron aquellos momentos mientras elaboraba este trabajo

A todos ellos:

Muchísimas gracias...

Gianni X. Jacobo Hidalgo

Contenido

Índice de figuras

Índice de tablas

Convenciones empleadas en el documento

Fuentes

Tabla. Fuentes empleadas en el presente documento

Convención	Descripción
Tw Cen MT	Texto normal
Tw Cen MT (cursiva)	*Concepto, definición, palabra clave, palabra coloquial o variable de una expresión matemática*
`Courier New`	`Código fuente o variable de código fuente`
`Courier New (cursiva)`	*`Archivo de código fuente`*
Times New Roman (cursiva)	*Expresión matemática o sintaxis de una instrucción*
Times New Roman	Aclaración o palabra destacable dentro de una expresión matemática o sintaxis de una instrucción

Abreviaciones

SO Operating System (Sistema Operativo)

IP Internet Protocol (Protocolo de Internet)

PCB Printed Circuit Board (Tabla del circuito impreso)

MAC Media Access Control Address (Direccion de Control de Acceso al Medio)

ADSL Asymetric Digital Subscriber Line (Linea Asimetrica Digital)

GNU GNU is Not Unix

LAN Local Area Network (Red de Area Local)

WLAN Wireless Local Area Network (Red de Area Local Inalambrica)

PC Personal Computer (Computadora Personal)

ISO International Organization for Standarization (Organización Internacional para la Estandarización)

USB Universal Serial Bus (Bus Serial Universal)

RX Recepcion Electronica Digital

OSI Open System Interconnection (interconexión de Sistemas Abiertos)

WPAN Wireless Personal Área Network (Red de Área Personal Inalámbrica)

DS Distribution System (Sistema de Distribución)

SSID Service Set Identifier (Identificador de Servicio)

SPI Serial Peripheral Interfaz (Interface Periferica Serial)

OTG On The Go (Sobre la Marcha)

HID Human Interface Device (Dispositivo de Interfaz Humana)

IRQ Interrupt Request (Solicitud de interrupción)

IPv#	Internet Protocol version # (Protocolo de Internet)	**ISR**	Interrupt Service Routine (Rutina de Servicio de Interrupción)
TTL	Transistor-Transistor Logic (Lógica Transistor-Transistor)	**OPAM**	Operational Amplifier (Amplificador Operacional)
AP	Access Point (Punto de Acceso)	**MAL**	Microchip Application Libraries (Bibliotecas de Aplicaciones de Microchip)

1. En el presente documento se emplearán las unidades de las magnitudes físicas especificadas en el Sistema Internacional.

2. Se emplearán las notaciones 10^n, o bien, la expresión E^n.

Capítulo 1
Generalidades

En este capítulo se describirá el objetivo de este proyecto, definido por una problemática real igualmente descrita, y el panorama amplio de las metas que se pretendieron lograr de acuerdo a la hipótesis planteada también en esta sección.

1.1 Introducción

Un sistema mecatrónico consiste de diversos componentes o módulos de naturaleza electrónica, mecánica e informática. No es solamente el conglomerado en sí: los módulos mantienen un control, una sinergia que lleva a la obtención de la realización de una o varias tareas.

Los sistemas mecatrónicos han cobrado mayor auge desde la aparición de los equipos computacionales personales. Gracias a ellos, un complejo sistema mecánico-electrónico puede ser controlado con relativa facilidad por un sector muy extendido de la población sin requerir conocimientos técnicos o tener grandes antecedentes de dicho sistema. Cada vez, y con mayor fuerza, el uso masivo de estos equipos obliga a que se evolucione hacia mejores métodos, mejor equipo, mayores facilidades y comodidades de uso, y por lo general, que representen costos lo más bajos posibles. Una impresora es un sistema típico mecatrónico. El usuario puede no conocer de su funcionamiento interno, pero gracias al software, puede emplearlo de manera muy sencilla (figura 1.1).

Figura 1.1 Impresora como sistema mecatrónico.

En la actualidad, una de las necesidades que se requieren en el mercado es la capacidad de controlar remotamente diversos servicios o labores. Una vez más, entran en juego los sistemas computacionales.

Para dar satisfacción a esta necesidad, se ha creado otra clase de dispositivos electrónicos: los *adaptadores de red*, los cuales tienen varias clasificaciones; una de ellas es la manera en la que hacen su conexión entre ellos y otros dispositivos; o bien poseen conexión física (alámbricos), o bien, no la requieren (inalámbricos).

Los adaptadores de red alámbricos presentan una serie de desventajas, las principales son la ocupación de un espacio físico, y el costo de la instalación.

Por ello, los adaptadores de red inalámbricos son cada vez acogidos por un mayor número de personas para el uso doméstico y en el ámbito laboral.

Los adaptadores emplean estándares para establecer la comunicación, siendo el más común el estándar 802.11, cuya certificación es la reconocida mundialmente Wi-Fi. Su uso más extendido es para computadoras que no pueden tener una conexión física a una red informática cableada, pero sí a una inalámbrica que use alguna versión del estándar 802.11. Existen diferentes modelos y marcas como las que muestra la figura 1.2, donde (a) es una ENUWI G2 de Encore Electronics y (b) es una TL 722n de TPLINK.

(a) (b)

Figura 1.2 Adaptadores USB de red (o tarjeta USB de red).

Entre las computadoras y sus accesorios de entrada y salida, una de las interfaces de comunicación más empleadas es la USB, cuyo estándar ha sido aceptado e integrado también en estos adaptadores de red. Gracias a esta interfaz, son bastantes los dispositivos que pueden ser conectados a una computadora y ser utilizados para diferentes actividades.

1.2 Planteamiento del problema

Un grupo de trabajo denominado "Valtroniks", ubicado en la ciudad de Cuernavaca, en Morelos, México, ha adquirido material electrónico para futuros proyectos. Principalmente ha adquirido interés por microcontroladores de alta gama, especialmente los creados por Microchip Technologies Inc. (denominados PIC). Con intensiones de aprender el uso de los que emplean arquitectura de 32 bits, han adquirido un circuito entrenador de desarrollo, el PIC32 USB Starter Kit II (figura 1.3).

Figura 1.3 PIC32 USB Starter Kit II.

También han deducido que las aplicaciones mecatrónicas que se desarrollarán en algún futuro emplearán comunicaciones inalámbricas. Sin embargo, han descartado tecnologías usuales como Bluetooth, ZigBee y MiWi. Las únicas redes con las que se contarían son las Wi-Fi, empleadas en *Puntos de Acceso* caseros o de oficina.

Diversas compañías han resuelto el problema de entablar estas comunicaciones por medio de circuitos especiales (módulos) para hacer la conexión de sus propios microcontroladores con redes inalámbricas Wi-Fi.

El grupo de trabajo espera no tener que emplear dichos módulos por diversas razones: sólo cuentan con redes WLAN y computadoras con sistemas operativos GNU/Linux y Windows; esperarían también no usar dispositivos físicos ni software adicionales en sus propias computadoras para el control de las aplicaciones mecatrónicas. Para entablar comunicación de forma inalámbrica sólo cuentan con adaptadores de red inalámbrica para PC con interfaz USB.

Entonces, se tienen solamente tres componentes para realizar alguna aplicación mecatrónica controlada inalámbricamente: un circuito de desarrollo (USB Starter Kit II) capaz de entablar comunicación USB con dispositivos, adaptadores de red inalámbrica con interfaz USB y una red con estándar 802.11 con versiones *a*, *b*, *g* y *n* (en adelante 802.11x).

1.3 Pregunta de investigación

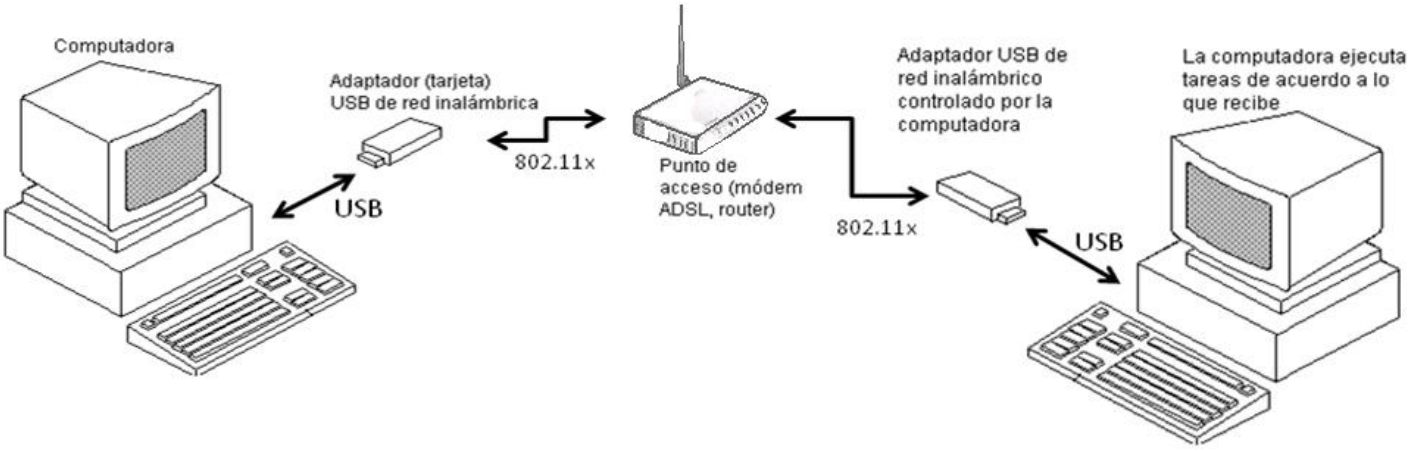
Lo anterior llevó a plantear la siguiente pregunta:

¿Puede hacerse funcionar un adaptador de red inalámbrica de interfaz USB por medio de un microcontrolador PIC, con el fin de recibir inalámbricamente información a través del estándar 802.11 para manejar alguna aplicación mecatrónica?

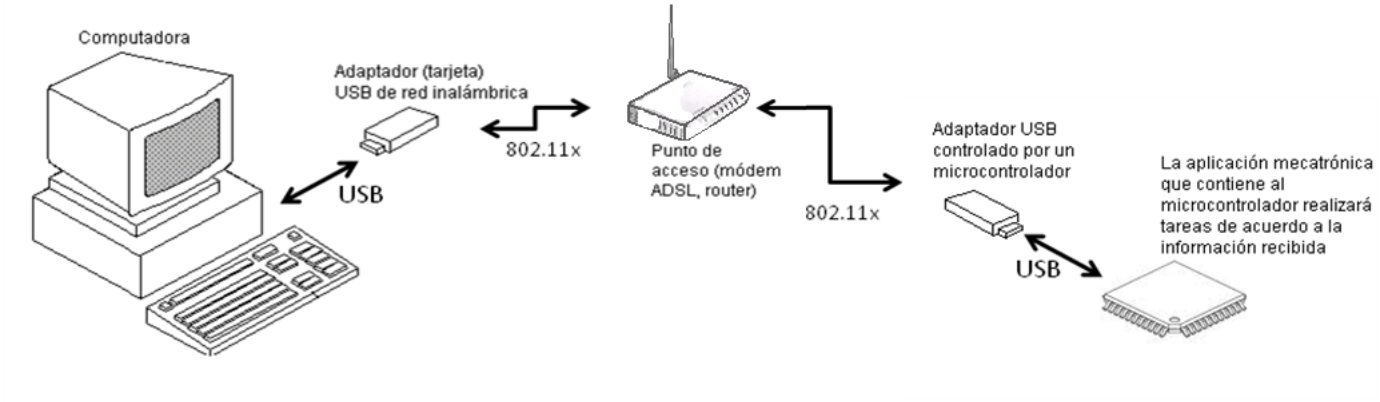
1.4 Hipótesis y justificación

Puede diseñarse la aplicación mecatrónica para recibir comandos de cualquier computadora, usando al microcontrolador. El mismo microcontrolador hará funcionar al adaptador (tarjeta) USB de red, de la misma forma que una PC lo haría.

Para hacer que la tarjeta de red funcione como si estuviera enchufada a una computadora, es necesario que sean *replicados* los comandos que intercambia (por interfaz USB) una PC con la tarjeta. De esta manera la tarjeta realizaría normalmente su función principal: enviar y recibir información por el estándar 802.11. Gráficamente esto se ilustra en la figura 1.4, donde (a) muestra una WLAN típica y (b) es la hipótesis del presente trabajo.



(a)



(b)

Figura 1.4 El microcontrolador manipulando a un adaptador de red inalámbrica en una WLAN.

Se evitaría adquirir módulos de comunicación inalámbrica especiales, que solamente funcionan para los microcontroladores, sin poderse emplear para algún otro fin. Los adaptadores USB, en caso de no requerirse en la aplicación, pueden ser desmontados y emplearse para las computadoras.

A diferencia de Bluetooth y otros estándares, la mayor parte de los componentes son fáciles de adquirir. Emplear un PIC32 es recomendado para aplicaciones que requieran de mucha memoria y altas capacidades de procesamiento y darían la facilidad de interactuar con otros dispositivos en caso de que sea requerido, especialmente aquellos que empleen interfaz USB.

Cualquier computadora dentro de la red podría interactuar con el microcontrolador sin importar el sistema operativo ni el idioma.

De resultar este proceso, se podría retomar para lograr hacer funcionar algún otro componente que también emplee interfaz USB, especialmente las tarjetas de red inalámbricas.

1.5 Estado del arte

Diferentes aparatos electrónicos hacen uso de las redes inalámbricas para sus fines. Ejemplos de ellos son las impresoras inalámbricas, las televisiones inteligentes, sistemas de seguridad, etc.

El uso que hacen de la red inalámbrica es igualmente por el estándar 802.11, pero la circuitería o sus módulos inalámbricos están ya inmersos en dichos dispositivos.

Para el desarrollo de proyectos aleatorios, Microchip ha creado circuitos complementarios que permiten la conexión de sus microcontroladores con redes Wi-Fi.

Por otro lado, estos circuitos o módulos tienen la gran ventaja del ahorro de energía, una documentación lo suficientemente descriptiva, y la facilidad de emplear lenguajes de programación de alto nivel en el desarrollo del *firmware* de algún proyecto. Un gran número de trabajos han empleado dichos módulos que comprobaron el adecuado manejo del estándar inalámbrico.

En un número reducido, pero de igual importancia, otras empresas han elaborado módulos tanto para sus mismos microcontroladores como para todos en general.

Sin embargo, una simple necesidad de entablar comunicaciones inalámbricas con el estándar 802.11 (aún para cuando el intercambio de información sea mínimo) podría implicar la necesidad de la obtención de estos módulos inalámbricos. Un gran inconveniente es que dichos módulos no son fabricados en la República Mexicana.

1.6 Solución propuesta

La comunicación inalámbrica propuesta requiere entonces emplear dos estándares, el estándar USB y el estándar 802.11 en alguna de sus versiones (*a,b,g* ó *n*). Como se planteó anteriormente, se requiere *replicar* los comandos que una PC cualquiera mandaría a una tarjeta de red para hacerla funcionar.

Como solamente se tiene al USB Starter Kit II, se debe programar su microcontrolador principal para que mande estos mismos comandos a la tarjeta de red, logrando, teóricamente, que realice sus funciones principales a semejanza de como si lo hiciera conectada a una PC.

El siguiente paso es analizar dichos comandos, e identificar aquellos que se requieren para que la tarjeta se asocie a una WLAN. Nuevamente estos comandos se replican y deben ser mandados por USB y, de funcionar la hipótesis, la tarjeta realizará el proceso de asociación en una red inalámbrica con estándar 802.11.

Si se asocia el microcontrolador (a través de la tarjeta) a una WLAN, obtendrá una dirección IP, misma que será almacenada. Con una dirección IP, cualquier computadora dentro de la red puede mandar información hacia dicha IP y, por lo tanto, si el microcontrolador es parte de una aplicación mecatrónica, podría responder con acciones específicas ante tal información.

Con el fin de probar esta comunicación, se hará el diseño de un modelo de aplicación mecatrónica. Cualquier computadora podrá mandar información, que la aplicación interpretará particularmente como un comando para realizar alguna actividad.

1.7 Objetivos

1. Lograr que la tarjeta de red reconozca al PIC32 USB Starter Kit II como una computadora, para que pueda mandar y recibir información inalámbricamente con el estándar 802.11.

   Esto se lograría con:

   - Recopilar el conocimiento necesario del estándar USB.
   - Recopilar el conocimiento necesario del estándar 802.11.
   - Comprender el funcionamiento y las herramientas para emplear al PIC32 USB Starter Kit II con interfaz USB.
   - Observar y analizar el tráfico USB entre una computadora y la tarjeta de red, e identificar la información que corresponda al estándar 802.11.
   - Replicar la información USB recabada en el kit mencionado.
   - Probar que el adaptador de red funcione con el USB Starter Kit II.
   - Verificar que sea exitosamente transferida alguna información (permitida por el estándar 802.11) entre una computadora y la tarjeta de red manejada por el USB Starter Kit II.
   - Realizar repeticiones de seguridad y comprobar el funcionamiento correcto en todas ellas.

2. Diseñar la aplicación mecatrónica que emplee la comunicación desarrollada.
    - Seleccionar una aplicación adecuada para la comunicación.
    - Seguir la metodología de diseño mecatrónico para su creación.
    - De funcionar la comunicación, implementar físicamente el modelo diseñado.

1.8 Metas

Esta comunicación podría emplearse para otras aplicaciones de la misma naturaleza o con características similares, bajo las restricciones que se encuentren durante su desarrollo.

Por otra parte, el diseño de la aplicación mecatrónica es una base para construir nuevos proyectos que tengan como finalidad lograr el objetivo final en caso de fallar la presente hipótesis.

Capítulo 2

Marco teórico

En el presente capítulo se expone con mayor precisión los conceptos que se requerirán conocer para desarrollar la comunicación y proceder el diseño de la aplicación objetivo.

La primera parte del capítulo explica, a grandes rasgos, acerca de las características comunes de las diferentes tecnologías de transmisión de información.

La segunda parte del capítulo expone las principales características de la comunicación inalámbrica, principalmente el estándar 802.11.

La tercera parte explica brevemente la definición de microcontrolador, los denominados PIC y sus características, y algunas aplicaciones que incluyen conceptos que se utilizarán durante la metodología de diseño del producto mecatrónico.

La cuarta parte expone el funcionamiento y los atributos más esenciales del estándar USB (hasta la versión 2.0).

Finalmente, en la quinta parte, se introduce al empleo del firmware desarrollado por Microchip para manejar el estándar USB en sus microcontroladores PIC32.

2.1 Comunicaciones electrónicas de redes informáticas

Con el fin de estudiar a las comunicaciones, ISO diseñó el modelo OSI, el cual las divide en 7 capas funcionales, mostrado en la figura 2.1.

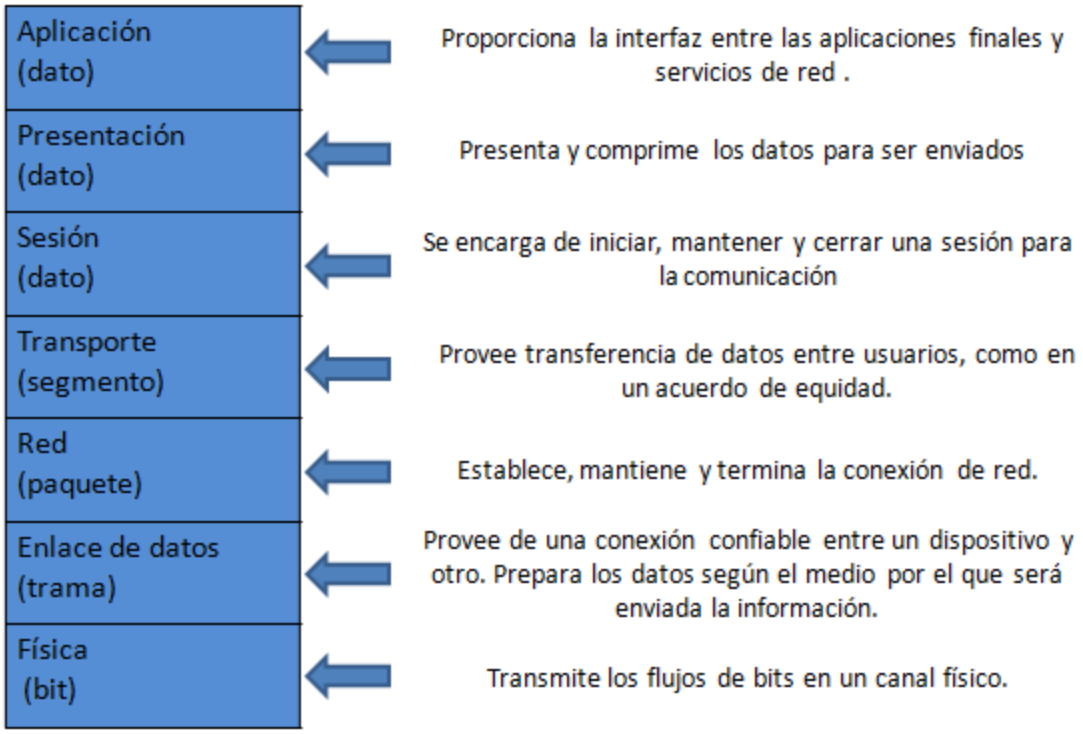


Figura 2.1 Modelo OSI.

El presente proyecto se pretende desarrollar en toda la pila del modelo OSI, tomando en cuenta el encapsulamiento adecuado en cada una de las capas con el fin de lograr un correcto intercambio de información. En las primeras 2 capas se encuentra la principal problemática a tratar: la transmisión de datos, la cual se abordará más adelante.

2.1.1 Principales protocolos de las capas de red y transporte

- **2.1.1.1 Capa de red**

El protocolo de la capa de red más utilizado es *IPv4*, que tiene información necesaria para asignar una ruta a los paquetes de red y es documentado en el RFC (Request For Coment) 791. Tiene dos responsabilidades principales:

1. Entregar paquetes a través de la red basado en el mejor esfuerzo.

2. Fragmentar y reensamblar paquetes para soportar transferencias de más de un kilobyte.

El encabezado del paquete IPv4 es mostrado en la figura 2.2.

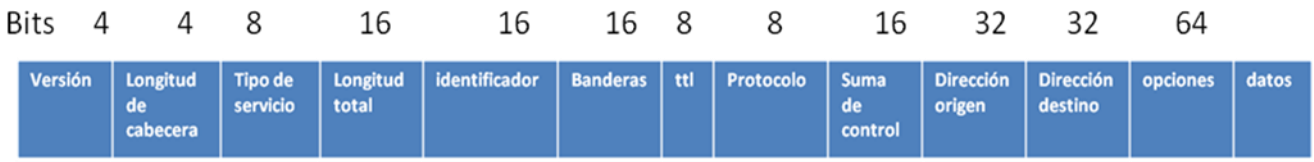


Figura 2.2 Encabezado de paquete IPv4.

El encabezado de los paquetes IPv4 tienen campos que se definen a continuación:

Versión: Define la versión de IP que se está utilizando, 0100b para IPV4 o 0110b para IPV6.

Longitud de cabecera: Indica la longitud del encabezado en palabras de 32 bits.

Tipo de Servicio: Permite al host indicar el tipo de servicio de red que quiere.

Largo o longitud total: Este campo indica la longitud total del paquete IP en bytes.

Identificación: Este campo lleva un número generado por el nodo origen para que el host destino identifique a qué paquete pertenece el fragmento recién llegado (cuando existe fragmentación en el trayecto del paquete).

Banderas: Para fines de fragmentación.

Offset o Desplazamiento del Fragmento: Indica qué tanto se ha desplazado el fragmento en relación con el comienzo de los datos originales (cuando el paquete tuvo que ser fragmentado).

Time To Live: Este campo indica el número máximo de *saltos* que un paquete puede dar (255). En cada salto el número de este campo se decrementa en 1. Cuando tiene un valor 0 y no se ha alcanzado el destino el paquete se descarta y el nodo fuente es informado.

Protocolo: Identifica el protocolo de capa superior que está encapsulado en el paquete IP según la numeración del RFC-1700.

Suma de control: Sólo para la cabecera IP, y es útil para la detección de errores generados por palabras de memoria erróneas en los routers.

Dirección Fuente: Dirección IP origen.

Dirección Destino: Dirección IP destino.

Opciones: Son de longitud variable y se utiliza el campo cuando se considera necesario.

- **2.1.1.2 Capa de transporte**

Existen 2 protocolos principales asociados a la capa de transporte (ver tabla 2.1): TCP (Transmission Control Protocol) y UDP (User Datagram Protocol).

Tabla 2.1 Comparativa general entre TCP y UDP.

Característica	TCP	UDP
Tamaño mínimo	21 bytes	8 bytes
Orientado a conexión	Sí	No
Tiempo de respuesta	Largo	Corto

Dado que se requiere el mínimo de información en el presente proyecto es conveniente usar UDP. Tiene las siguientes características:

UDP asume que el IP se utiliza como protocolo subyacente, utiliza un mínimo de mecanismo de protocolo. El protocolo se orienta a transacciones, y tanto la entrega como la protección ante duplicados no se garantizan.

Figura 2.3 Formato de la cabecera del protocolo de datagrama de usuario.

Los campos de la cabecera UDP son los siguientes:

Puerto de Origen: Indica el puerto lógico del proceso emisor. Si no se utiliza, se inserta un valor cero.

Puerto de Destino: Indica el puerto lógico del proceso destino.

Longitud: Representa la longitud en octetos de este datagrama de usuario, incluyendo la cabecera y los datos. La longitud mínima son 8 bytes, que es la longitud de la cabecera. La longitud máxima teórica del datagrama son 65535 bytes pero esto queda limitado a 65507 bytes si se usa IPv4 (campo obligatorio)

Suma de Control: Es construida con información de la cabecera IP, la cabecera UDP y los datos, y es útil para detectar errores.

2.2 Transmisión en una red de datos

Para poder enviar información de un dispositivo a otro es necesario contar con un sistema para transportarla ya sea mediante dispositivos de almacenamiento o de todo un sistema de transmisión, los cuales pueden dividirse en alámbricos e inalámbricos.

2.2 .1 Alámbrica

Cable Coaxial (Coaxial Cable): El cable coaxial está formado por un alambre de cobre rígido como núcleo, rodeado por un material aislante, a su vez forrado con un conductor

cilíndrico, frecuentemente es una malla de tejido fuertemente trenzado. El conductor externo se cubre de una envoltura protectora de plástico.

Par Trenzado (Twister-Pair Cabling): Consta de un par de alambres de cobre aislados de 1mm de grosor generalmente, trenzados en forma helicoidal, con el objetivo de reducir la interferencia eléctrica de pares similares cercanos. Existen dos tipos de par trenzado: UTP (Unshielded Twister Pair) y el STP (Shielded Twister Pair). El conector para el cable par trenzado más utilizado es el RJ45.

Fibra Óptica: Es un medio de vidrio, forrado por un aislante plástico, en el cual la información viaja en forma de luz. Tiene alta inmunidad al ruido, baja pérdida de información y confiabilidad de la misma a velocidades luz, cuenta con alta velocidad de propagación, poca atenuación, inmunidad a interferencias electromagnéticas y resiste la corrosión. Los dos conectores para fibra óptica más comunes son el ST (Straight Tip) y SC (Subscriber Connector).

2.2.2 Inalámbrica

La mayoría de las conexiones son cableadas, pero existen situaciones en las que esto no es posible porque los equipos se mantienen en continuo movimiento, el lugar no permite realizar modificaciones a las instalaciones o simplemente porque la comunicación inalámbrica es una alternativa menos costosa.

Existen diferentes tecnologías para la comunicación inalámbrica, algunas son mostradas en la tabla 2.2 y las otras son abordadas posteriormente.

Tabla 2.2 Tecnologías inalámbricas.

Tecnología	Descripción
Microondas	Son señales inalámbricas que viajan por encima de los 100 MHz, en línea recta. Cuyo sistema de transmisión consiste en al menos 2 torres de microondas equipadas con antenas direccionales, las cuales deben estar alineadas, formando la llamada *línea de vista*; por la curvatura de la tierra, dichas antenas están muy cerca y se requieren antenas más altas para una mayor distancia entre ellas. La distancia máxima entre 2 torres es de 50 Km y no debe de haber obstrucciones entre ellas. Transmiten a una frecuencia en el rango de los 2 a 25 GHz.
Infrarrojo	Este medio de transmisión tiene corto alcance. Son relativamente baratos y fáciles de construir. Usan una sola frecuencia. Consiste en una unidad base conectada al servidor y dispositivos, la cual tiene 2 nodos ópticos, uno para enviar y otro para recibir información. Este tipo de comunicación tiene una característica importante: no atraviesan cuerpos sólidos. Esto tiene la desventaja de no poder existir una comunicación entre paredes, pero tiene la ventaja de que no interfiere con otras comunicaciones.

- **2.2.2.1 Estándar 802.15**

Por la creciente necesidad de dispositivos que sean capaces de enviar una baja transferencia de datos con bajo consumo de energía se desarrolla un grupo de comunicaciones inalámbricas para redes que abarquen áreas muy reducidas llamadas WPAN.

Estas redes cumplen el estándar IEEE 802.15.x. Están orientadas a dispositivos como los sensores: de bajo coste, con capacidades de procesado, memoria de almacenamiento y batería limitados.

Las WPAN tienen un alcance de alrededor de los 10 m. Se definieron diferentes grupos de trabajo para la estandarización de las redes WPAN, los más comunes se muestran en la tabla 2.3

Tabla 2.3 Familia 802.15.

Características	**802.15.1 (Bluetooth)**	**802.15.3 (Transmisión de alta velocidad)**	**802.15.4 (LoWPAN Zigbee)**
Rango de datos	1-2 Mbps	11,12, 33, 44 ó 55 Mbps	20, 40 ó 250 Kbps
Potencia de salida	100 mW	6 mW	1 mW
Alcance (en interiores)	100 m	20 m	20 m
Banda de frecuencia	2.4 Ghz	2.4GHz	2.4 GHz
Canales	78	3 ó 4	26
Información técnica disponible	Media-Escasa	Adecuada	Muy escasa
Costo	Bajo	Alta	Alto
Eficiencia	Media	Buena	Buena
Compatibilidad	802.11	802.11	802.11

Es posible observar que 802.15.3 tiene un menor alcance pero generalmente tienen altas eficiencias, con lo que es notorio su objetivo.

- **2.2.2.2 Estándar 802.11**

802.11 es una extensión de los estándares IEEE 802. Utiliza direcciones de 48 bits. El estándar IEEE 802.11 utiliza para el envío de información el protocolo CSMA/CA (Control acceso al medio con detección de portadora y prevención de colisiones).

Las redes 802.11 también usan acuse de recibo de enlace de datos para confirmar que una trama se recibió con éxito. Si la estación transmisora no detecta la trama de

reconocimiento, ya sea porque la trama de datos original o el reconocimiento no se recibieron intactos, se retransmite la trama. Este reconocimiento explícito supera la interferencia y otros problemas relacionados con la radio.

Otros servicios admitidos por la 802.11 son la autenticación, asociación (conectividad a un dispositivo inalámbrico) y confidencialidad (cifrado). Se han desarrollado diferentes versiones del estándar como se puede observar en la tabla 2.4.

Tabla 2.4 Familia 802.11.

Tecnología	**802.11a**	**802.11b**	**802.11g**	**802.11n**
Banda (GHz)	5.7	2.4	2.4	5.7 y 2.4
Canales (no superpuestos)	23	3	3	2
Modulación	OFDM	DSSS	DSSS OFDM	MIMO OFDM
Velocidad máxima de los datos (Mbps)	54	11	54	300
Alcance (m)	35	35	35	70
Aprobación	Octubre de 1999	Octubre de 1999	Junio de 2003	Septiembre de 2009
Pros	Rápido, menos susceptible a interferencias	Bajo costo, buen alcance	Bajo costo, rápido y difícil de obstruir	Buena velocidad de transferencia
Contras	Costoso, poco alcance	Lenta, susceptible a interferencias	Susceptible a interferencias	Costoso

El estándar 802.11 establece un formato para la trama de datos, el cual es mostrado en la figura 2.2.

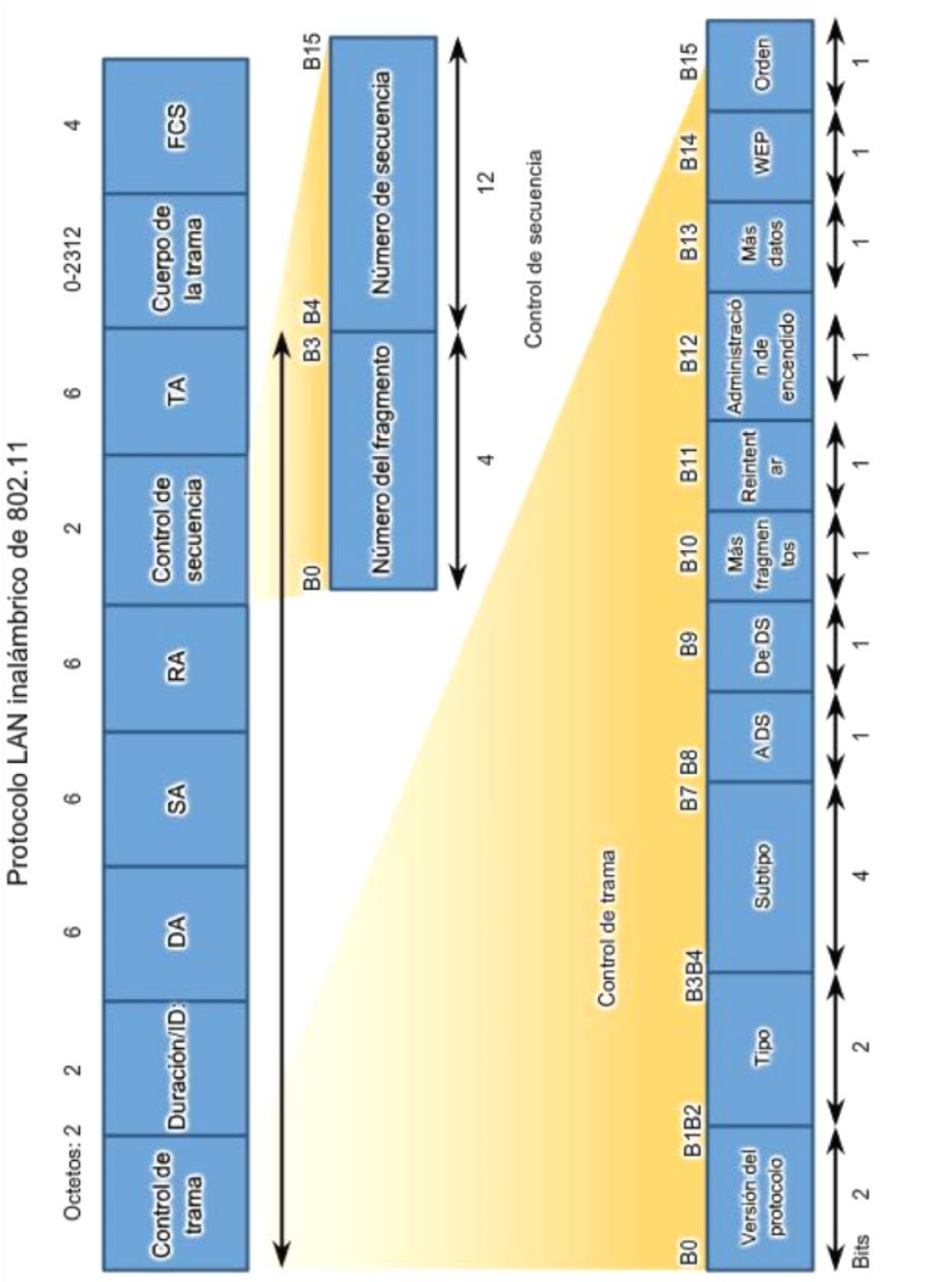


Figura 2.4 Estructura de la trama de datos del estándar 802.11.

Los campos de la trama son los siguientes:

Versión del protocolo: la versión de la trama 802.11 en uso. Hasta la fecha de elaboración de este trabajo, se usa siempre un 0.

Tipo y subtipo: identifica una de las tres funciones y subfunciones de la trama, puede tomar los valores que muestra la tabla 2.5.

Tabla 2.5 Tipo y subtipo de la trama 802.11.

Tipo (binario)	Descripción	Subtipo (binario)	Descripción
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	0110-0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Desauthentication
00	Management	1101-1111	Reserved
01	Control	0000-1001	Reserved
01	Control	1010	PS-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF End
01	Control	1111	CF End + CF-ACK
10	Data	0000	Data
10	Data	0001	Data + CF- Ack
10	Data	0010	Data + CF- Poll
10	Data	0011	Data + CF- Ack + CF-Poll
10	Data	0100	Null Fuction
10	Data	0101	CF-Ack
10	Data	0110	CF-Poll
10	Data	0111	CF-Ack +CF-Poll
10	Data	1000-1111	Reserved
11	Reserved	0000-1111	Reserved

A DS: Establecido en 1 en las tramas de datos destinadas al *sistema de distribución* (dispositivos en la estructura inalámbrica).

Desde DS: Establecido en 1 en tramas de datos que salen del sistema de distribución.

Más fragmentos: Establecido en 1 para tramas que tienen otro fragmento.

Reintentar: Establecido en 1 si la trama es una retransmisión de una trama anterior.

Administración de energía: Establecido en 1 para indicar que un nodo estará en el modo ahorro de energía.

Más datos: Establecido en 1 para indicar a un nodo en el modo ahorro de energía que más tramas se guardan en la memoria del búfer de ese nodo.

WEP (Privacidad equivalente por cable): Establecido en 1 si la trama contiene información cifrada por seguridad.

Orden: Establecido en 1 en una trama de tipo datos que utiliza la clase de servicio Estrictamente ordenada (no requiere reordenamiento).

Duración/ID: Según el tipo de trama, representa el tiempo, en microsegundos, requerido para transmitir la trama o una identidad de asociación (AID) para la estación que transmitió la trama.

Dirección de destino (DA): Dirección MAC del nodo de destino final en la red.

Dirección de origen (SA): Dirección MAC del nodo que inició la trama.

Dirección del receptor (RA): Dirección MAC que identifica al dispositivo inalámbrico que es el receptor inmediato de la trama.

Dirección del transmisor (TA): Dirección MAC que identifica al dispositivo inalámbrico que transmitió la trama.

Número de secuencia: Indica el número de secuencia asignado a la trama; las tramas retransmitidas se identifican por números de secuencia duplicados.

Número de fragmento: Indica el número de cada fragmento de la trama.

Cuerpo de la trama: Contiene la información que se está transportando; para tramas de datos, generalmente un paquete IP.

FCS: Contiene una verificación por redundancia cíclica (CRC) de 32 bits de la trama.

El proceso que se sigue para asociar un dispositivo a un DS es el que se muestra en la figura 2.5. Consta principalmente en 6 pasos si la red es abierta, es decir si no cuenta con un sistema de cifrado.

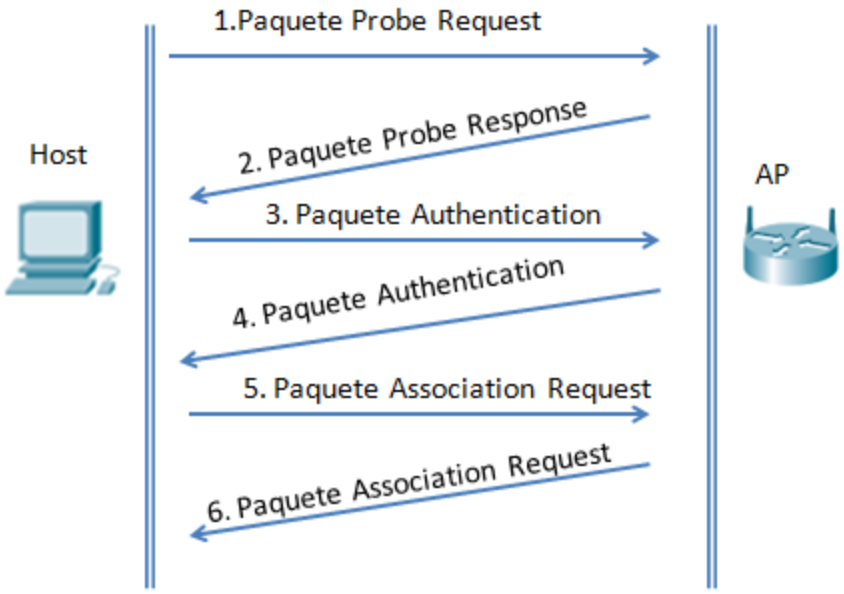


Figura 2.5 Proceso de asociación de un host a un DS.

Paso 0. El DS envía periódicamente tramas denominadas *Beacons* (opcionalmente se puede configurar el DS para que no envíe Beacons) que contienen información acerca de sí mismo. Estos sólo demuestran que el DS está activo, ya que no son necesarios para establecer la comunicación.

Estos paquetes contienen información acerca de todas las capacidades y características que se deben cumplir para establecer la comunicación. Al ser un anuncio a todos los dispositivos la dirección de destino es la de *broadcast*. La figura 2.6 representa el esquema de la trama Beacon.

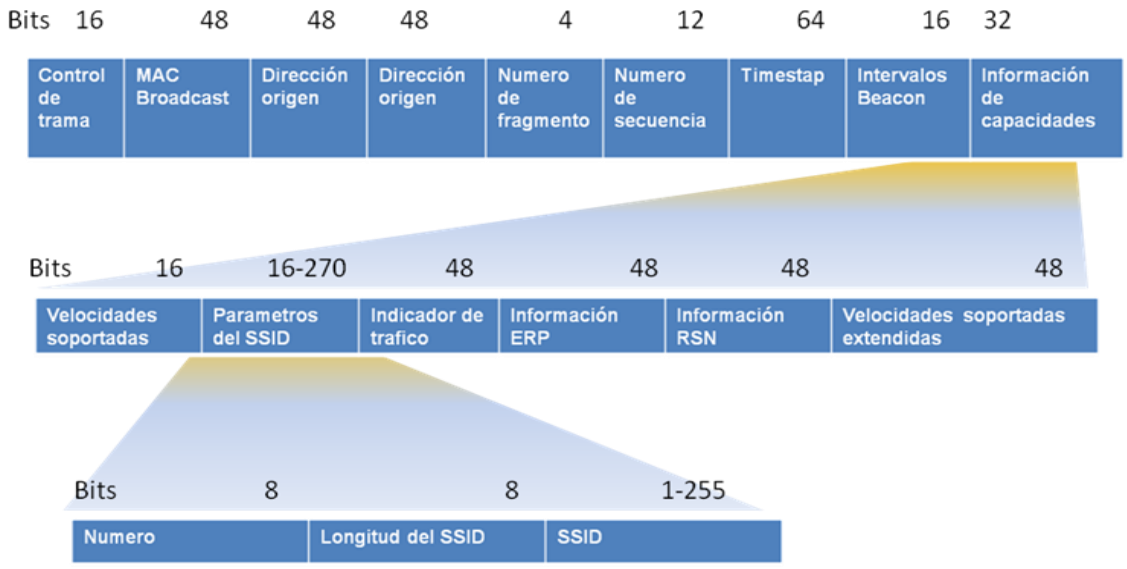


Figura 2.6 Estructura de trama Beacon.

Paso 1. El host envía un trama denominada Probe Request, la cual es una solicitud del host al DS para que le envíe características específicas para establecer la comunicación.

La figura 2.7 representa el esquema de la trama Probe Request, aunque la dirección de destino es broadcast lleva información de un DS en específico, como son los parámetros del SSID.

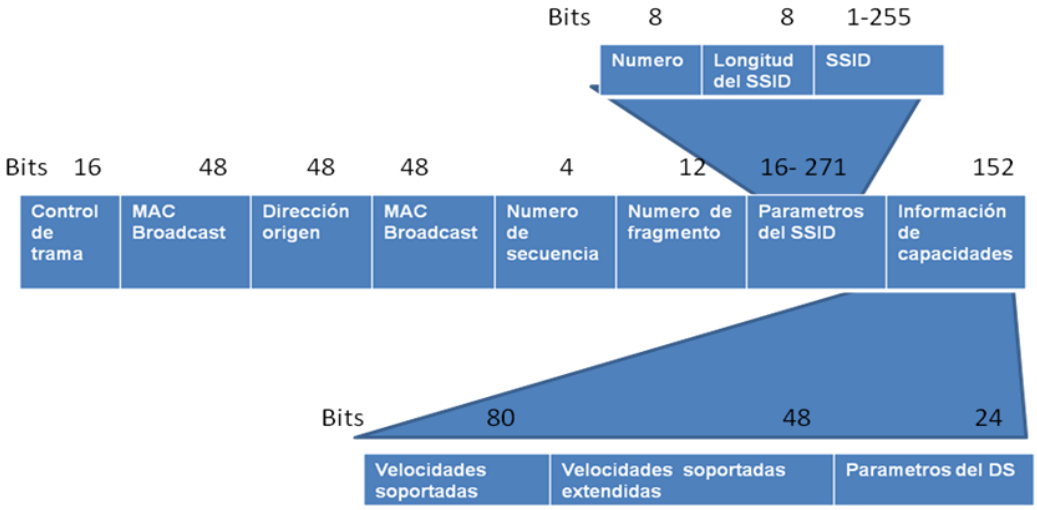


Figura 2.7 Estructura de una trama Probe Request.

Paso 2. El DS regresa una trama del tipo Probe Response con toda la información acerca de sí mismo para poder entablar la comunicación. La figura 2.8 representa el esquema de la trama Probe Response. Como se observa la estructura de las tramas 802.11 son muy similares, pero cambian algunos campos y principalmente se distingue el control de trama 0x50 00 3A 01.

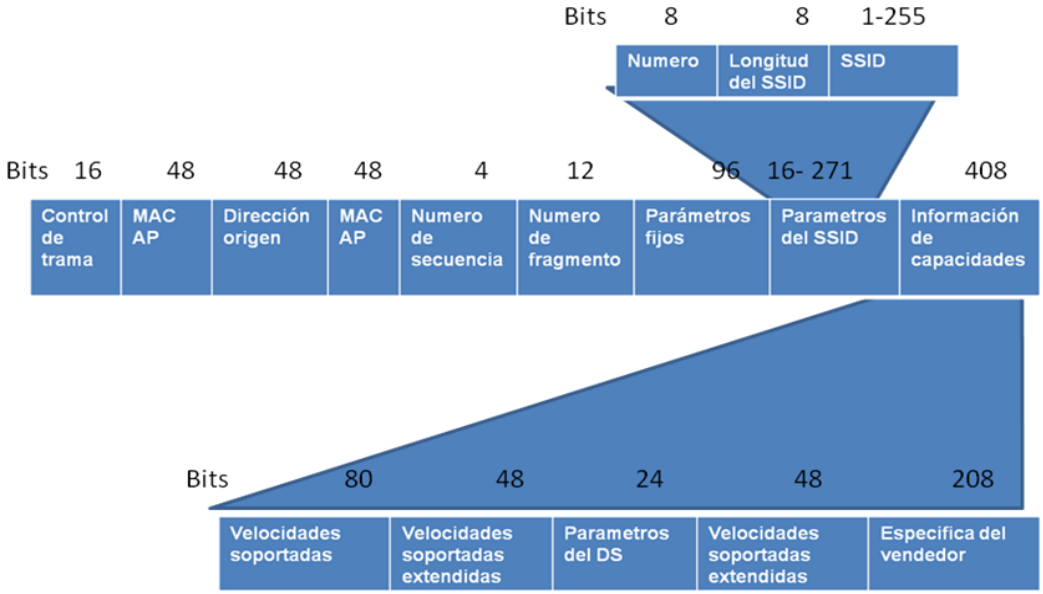


Figura 2.8 Estructura de trama Probe Response.

Paso 3. El host envía una trama denominada Authentication para solicitar al DS que lo reconozca. La figura 2.9 representa el esquema de la trama Authentication, esta es la única trama que contiene los mismos datos tanto del host al DS como del DS al host. Como se trata de una red abierta el algoritmo de autenticación se fija en 00.

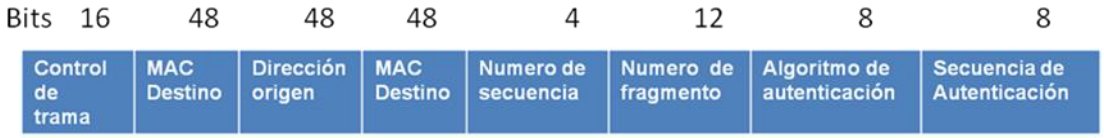


Figura 2.9 Estructura de trama Authentication.

Paso 4. El DS envía una trama Authentication al host, para informarle que lo ha reconocido.

Paso 5. El host envía una trama Association Request. La figura 2.10 representa el esquema de la trama Association Request, con la cual el host le pide formalmente al DS que comience la comunicación.

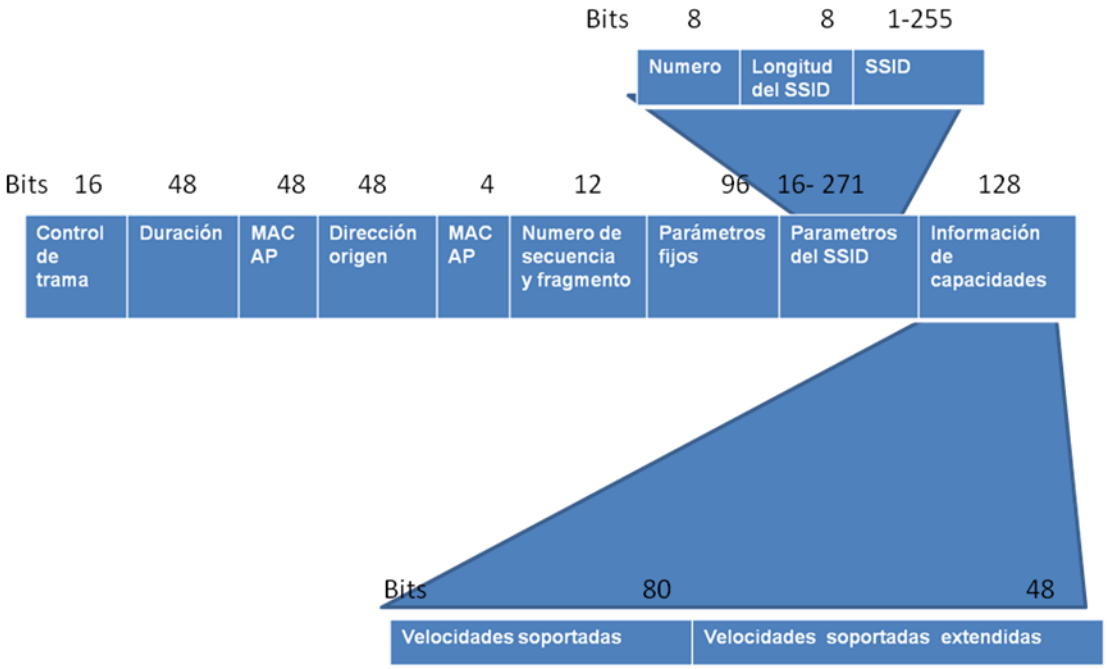


Figura 2.10 Estructura de trama Association Request.

Paso 6. El DS envía una trama Association Response. La figura 2.11 representa el esquema de la trama Asociación Response, con la cual el DS concluye el proceso de asociación.

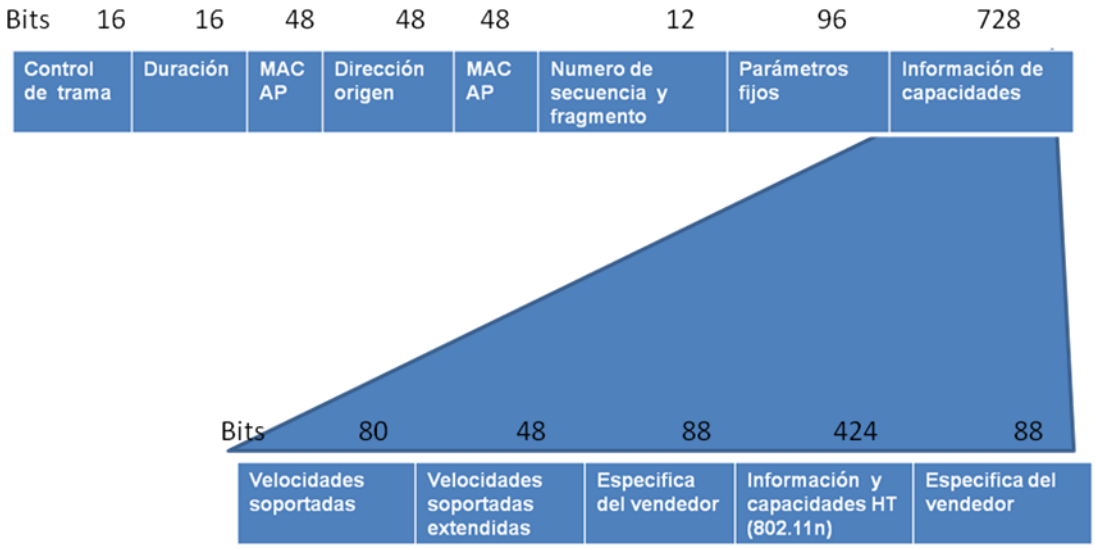


Figura 2.11 Estructura de trama Association Response.

El protocolo CSMA/CA especifica un procedimiento para el envío de datos, el cual es mostrado en la figura 2.12.

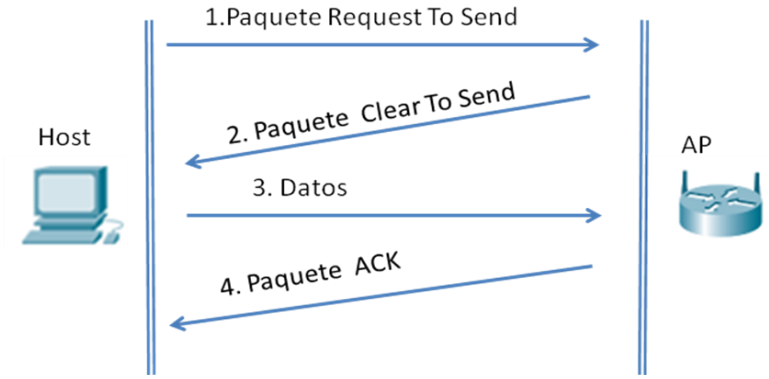


Figura 2.12 Proceso del protocolo CSMA/CA.

Paso 1. El host envía una trama de petición de envío denominada RTS (Request To Send). La figura muestra la estructura de una trama RTS.

Paso 2. Si el AP no está ocupado con otra transmisión se devuelve una trama tipo Clear To Send donde se le notifica al host que el canal esta libre para recibir datos.

Paso 3. El host envía los datos.

Paso 4. El AP envía una trama denominada ACK. Con la que notifica al host que ha recibido los datos completos.

La figura 2.13 muestra la estructura de las tramas, (a) es Trama Request To Send y (b) Trama Clear To Send, y ACK. Ambas tienen la misma estructura, la diferencia radica en la

dirección MAC destino, para la trama CTS es la dirección del host y en la ACK es la del AP.

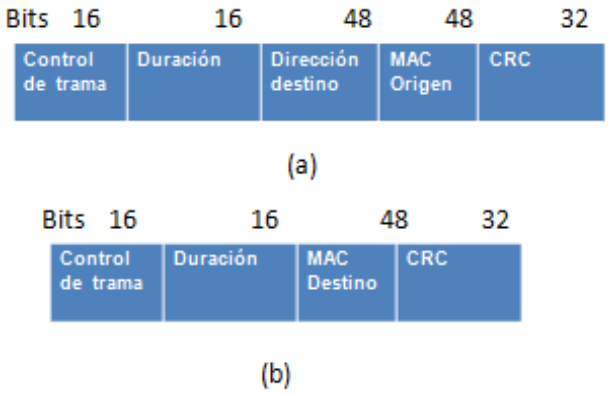


Figura 2.13 Estructura de tramas RTS, CTS y ACK.

2.2.3 Ping ICMP

Un comando común de emplearse en el protocolo IP es el Mensaje de Eco, de Respuesta de Eco o, informalmente, PING, definido dentro del RFC 0792 para el protocolo de mensajes de control de Internet (ICMP). Al ser un protocolo de control está contenido en todos los SO: así se podrá ejecutar PING en cualquier PC.

El comando PING es visto como en la figura 2.14.

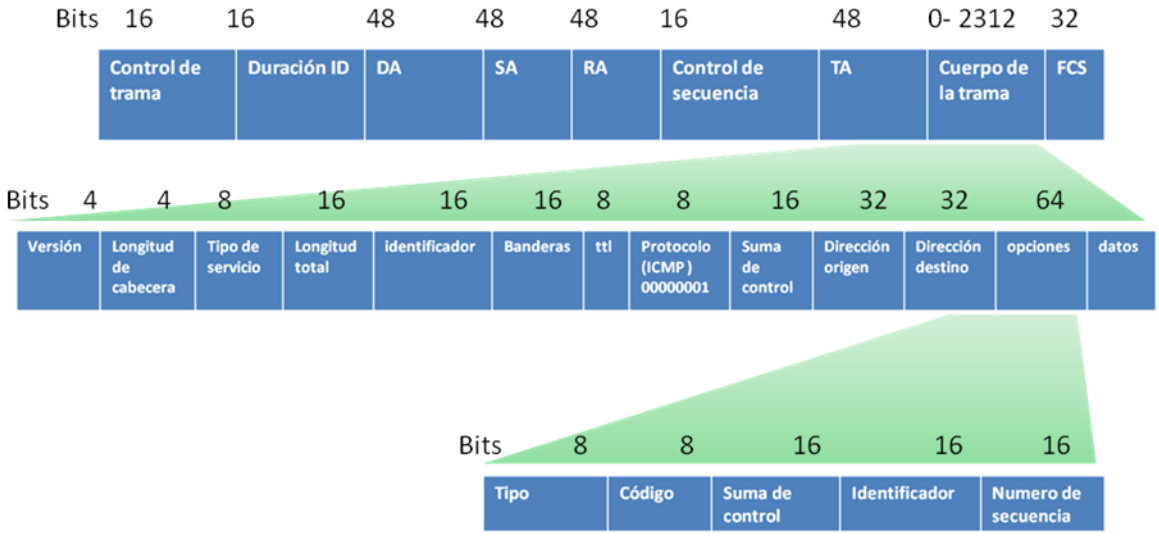


Figura 2.14 Estructura de Respuesta de Eco.

En el comando PING existe el campo de bits OPCIONES, que es explicado a continuación:

Tipo. Toma el valor 1000b para mensaje de eco y 0 para mensaje de respuesta de eco.

Código. Siempre fijo en 0.

Suma de Control. El complemento a uno de 16 bits de la suma de los complementos a uno del mensaje ICMP, comenzando por el tipo ICMP.

Identificador. Un identificador se usa como referencia para emparejar ecos y respuestas, que puede ser cero.

Número de Secuencia. Un número de secuencia se usa como referencia para emparejar ecos y respuestas, que puede ser cero.

El comando PING reconoce diversos parámetros para su uso, los cuales sirven para enviar datos específicos a la aplicación. Cada SO define los parámetros válidos para el comando. Como se puede observar en la tabla 2.6, es muy reducido el número de parámetros que coexisten en los sistemas operativos más comunes.

Tabla 2.6 Parámetros del comando PING.

Parámetro Windows	Parámetro GNU/Linux	Descripción
-n	-c	Número de peticiones eco para enviar.
-l	-s	Tamaño del búfer.
-i	-t	TTL (tiempo de vida)
-w	-w	Tiempo de espera en milisegundos para esperar cada respuesta.

Uno de los parámetros del comando PING que puede manipular el usuario es el tamaño del búfer.

Teniendo en cuenta la figura 2.14 se observa que el primer byte tiene el valor 0x88, lo cual indica que es de datos. Los bytes 33 y 34 son la longitud total del encapsulado IP en formato hexadecimal. El usuario puede mandar bytes (un búfer) adicionales, sin embargo, el valor 0x1C es el tamaño de la cabecera IP inalterable. Esto puede ayudar como indicador.

$$Tamaño\ del\ búfer = Longitud\ total - longitud\ de\ la\ cabecera\ (0x1C)$$

Así con una longitud total de 0x1F, la relación queda:

$$Tamaño\ del\ búfer = 0x1F - 0x1C$$

$$Tamaño\ del\ búfer = 0x03$$

2.3 Microcontroladores y sus aplicaciones

Un microcontrolador, abreviado como MCU o µC, es un circuito electrónico integrado programable. Una serie de instrucciones grabadas dentro de su memoria interna permiten a un microcontrolador ejecutar acciones secuenciales con el fin de realizar una o varias tareas personalizables.

Los elementos o módulos de un MCU, además de su memoria interna, son la unidad central de procesamiento (microprocesador) y los periféricos de entrada y/o salida.

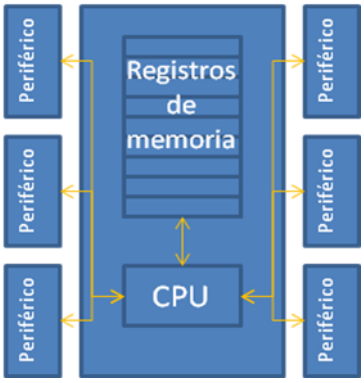


Figura 2.15 Esquema de los componentes de un microcontrolador.

La memoria de un microcontrolador básicamente son dos: la memoria RAM (memoria de acceso aleatorio) y la memoria ROM (memoria de sólo lectura). En la actualidad son usadas la EEPROM (Memoria de Sólo-Lectura Eléctricamente Borrable) y la denominada Flash (derivada de la EEPROM, con mayor rapidez de acceso).

La unidad central de procesamiento (núcleo, microprocesador o CPU), se encarga de ejecutar las instrucciones establecidas en la memoria del microcontrolador por medio de operaciones lógicas secuenciales. Estos cambios son realizados por pulsos de voltaje de un reloj u oscilador.

Los MCU incluyen periféricos (circuitos electrónicos que permiten la interacción del CPU con el exterior) y puertos. Físicamente suelen incluir pines de entrada/salida de propósito general (GPIO), cuyo comportamiento es determinado precisamente por el programa de la memoria interna.

A medida que han evolucionado a los MCU se les han agregado nuevos circuitos y puertos que facilitan el trabajo con otros tipos de dispositivos, por ejemplo, convertidores analógicos a digitales, temporizadores, y múltiples puertos físicos para protocolos o estándares de comunicación electrónica como RS-232, I^2C, SPI, Ethernet, Bluetooth y el extendido puerto de la especificación USB.

Gracias a ello, los microcontroladores tienen la posibilidad de emplearse como pequeñas computadoras embebidas para interactuar con dispositivos muy diversos que solamente requieren que trabajen bajo las mismas reglas de comunicación.

2.3.1 Microchip, los microcontroladores PIC y MPLAB

Entre las diversas compañías que se han especializado en la elaboración de microcontroladores se encuentra Microchip Technologies Inc., cuyos productos tienen un extendido soporte y documentación descriptiva de su funcionamiento.

Los microcontroladores PIC, fabricados por Microchip, son de fácil empleo y programación. Han creado también un software, entorno de desarrollo, para programar sus MCU tanto en lenguaje ensamblador como en lenguaje C, llamado MPLAB. Puede ser descargado desde su página web oficial gratuitamente, es compatible para PC con cualquiera de los siguientes sistemas operativos: Windows 7, XP, Vista, y GNU/Linux (versión MPLAB X).

Con MPLAB pueden también ser programados los MCU PIC, mediante la tecnología conocida como ICSP (In-Circuit Serial Programming). A grandes rasgos, con ICSP el MCU puede reconocer un nuevo código a programarse en su memoria interna, con algún puerto estandarizado de una PC.

Los PIC más recientes en su desarrollo por Microchip son los de la familia PIC32. Dichos PIC usan registros en su memoria de 32 bits, a semejanza de los computadores personales más empleados.

Los PIC32 tienen varias grandes diferencias de sus antecesores, debido en parte a que emplean un núcleo adquirido de MIPS Technologies. Una de sus características más notorias es el manejo que hacen de sus funciones internas en las llamadas *interrupciones*. Las interrupciones pueden ser manejadas por múltiples vectores, puede especificarse la prioridad, una sub prioridad, y de esta manera las interrupciones pueden quedar en una c*ola* en espera de ser manejadas. Sin embargo, los registros de control que manejan a las interrupciones existen tanto en el núcleo como en el módulo de Control de Interrupciones, elevando el nivel de complejidad de configuración.

Para familiarizar al usuario con el manejo de estos microcontroladores, Microchip ha fabricado tres circuitos "entrenadores", cuyo componente principal es uno de dichos MCU. Estos circuitos son: PIC32 *Starter Kit I*, PIC32 *USB Starter Kit II* y PIC32 *Ethernet Starter Kit III*.

Al ser adquiridos con el fabricante, el paquete incluye un CD que contiene una variante de MPLAB: *MPLAB IDE PIC32 Starter Kits*, que permite reprogramar y obtener información de cualquiera de los tres kits que esté conectado a la PC mediante interfaz USB. Emplea, primordialmente, el lenguaje C para programar a los PIC32.

Microchip ha desarrollado plantillas y archivos con códigos fuente para estos dispositivos, que pueden ser empleados para otros proyectos. Una colección de dichos archivos son llamados pilas (Stack). Entre ellos puede mencionarse, como ejemplo, al USB Stack de Microchip.

2.3.2 Aplicaciones con microcontroladores

Infinidad de proyectos son realizados a partir de microcontroladores. Se expondrán en este apartado las que se necesita comprender para los capítulos siguientes.

- 2.3.2.1 Robots móviles

Un robot es en esencia un sistema mecánico-electrónico-informático que ejecuta diversas tareas interactuando con el medio que le rodea. Posee la gran ventaja de ser flexible ante nuevas instrucciones que le sean dadas. Son *móviles* aquellos que son capaces de abandonar por sí mismos la posición espacial en la que se encuentran, es decir, son capaces de trasladarse de un punto de su entorno a otro sin necesidad de un elemento ajeno a ellos.

Para lograrlo, se plantean dos tipos de robots móviles: *con ruedas* y *con extremidades* (miembros o patas).

Los segundos son de mayor complejidad tanto teórica como práctica. Requieren de cálculos de mayor grado de dificultad, y muchos más elementos por incluir en el sistema de traslación.

Los primeros suelen no requerir demasiados cálculos ni componentes físicos, ya que el principal elemento de actuación son motores, que de alguna manera ven reflejado su efecto en el movimiento de las ruedas.

Tienen en contra que no son aptos para terrenos irregulares o deslizantes, o donde el terreno puede sufrir daños por el roce de las ruedas.

Existen diferentes tipos, se mencionan las más comunes:

Triciclo dásico: La rueda delantera sirve para hacer la tracción y el direccionamiento, por lo que las dos ruedas traseras están acopladas a un mismo eje y giran por el arrastre de la delantera. Su desventaja es su ocasional inestabilidad en terrenos abruptos. Figura 2.17 (b).

Ackerman: De cuatro ruedas, las dos delanteras definen la dirección, son independientes entre ellas. Las ruedas traseras proporcionan la tracción. En este sistema una de las llantas delanteras, al realizar una maniobra de cambio de dirección, realiza un ángulo ligeramente superior a la otra para evitar deslizamientos. Figuras 2.16 y 2.17 (a).

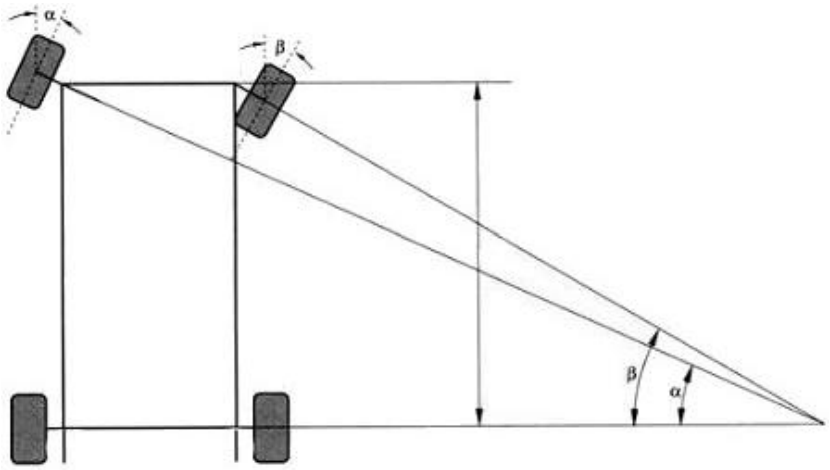
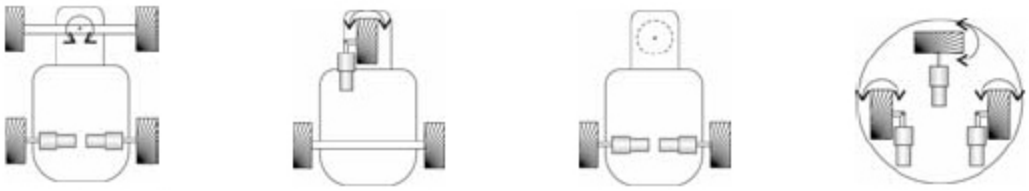


Figura 2.16 Sistema Ackerman.

De pistas de deslizamiento: Las ruedas de un mismo lado están envueltas por pistas, y actúan independientemente. Son útiles para la traslación en terrenos irregulares. Se logra una mejor resistencia al desgaste, y el impulso se limita menos por el deslizamiento.

Direccionamiento diferencial: Cada lado del vehículo cuenta con sistemas diferentes de tracción, y los cambios de dirección del vehículo se logran cuando las ruedas de un extremo rotan a una menor velocidad que las otras. Figura 2.17 (c).

De ruedas omnidireccionales: Las ruedas pueden obtener ubicaciones en diferentes ejes. Figura 2.17 (d).



(a) **(b)** **(c)** **(d)**

Figura 2.17 Diagrama de diferentes configuraciones de las ruedas en robots móviles.

- **2.3.2.2 Control remoto**

La interfaz del usuario puede estar físicamente integrada con la misma aplicación. En otras ocasiones la interfaz se encuentra a cierta distancia de ésta. Esto es un *control remoto*.

El control remoto requiere de dispositivos intermediarios (sistema de comunicación) que interpreten las señales eléctricas recibidas de la aplicación principal. Es común que el creador emplee estándares de transferencia de comunicación, pues los circuitos y el resto del hardware para dichos estándares pueden ser adquiridos en el mercado, listos para ser utilizados. El diseñador de la aplicación ya no tiene que encargarse de crear normas y el hardware del sistema de comunicación en sí, sino de preocuparse de saber la información necesaria para ser enviada o recibida.

También es frecuente que el control remoto se realice mediante una computadora común, y el creador tiene entonces que usar alguno de los puertos físicos de la computadora para enviar/recibir información, a no ser que modifique la computadora internamente, lo cual es más complejo y en muchos casos innecesario.

Los puertos de la computadora siempre son estándares: el creador de la aplicación necesita emplear alguno de estos estándares para que ambos, computadora y aplicación, usen las mismas normas de comunicación.

Figura 2.18 Robot MOBILE ARMATRON controlado por el puerto LPT de una PC.

Tanto la computadora como la aplicación deben reconocer la información, pero esto nuevamente dependerá del creador. Por ejemplo, una simple frase "Buenos días" (empleando un estándar) podría equivaler una orden a la computadora de que muestre el mapa de un país en el monitor, o que la aplicación debe calibrar el sensor de humedad del espacio de trabajo.

Algunos microcontroladores de las aplicaciones emplean los mismos estándares de los puertos físicos de una computadora, lo que implica que el diseñador de la aplicación no tiene que preocuparse en buscar tantos dispositivos intermediarios. Un ejemplo notable es el del estándar USB.

2.4 Estándar USB

USB es un estándar industrial cuyas especificaciones son creadas y reguladas por el USB Implementers Forum (USB IF), que es un conglomerado de diversas empresas de electrónica e informática.

USB evolucionó, por lo que ha tenido diversas versiones. La más reciente es la versión 3.0, pero debido a su relativamente reciente aparición en el mercado global, la versión más empleada sigue siendo la versión 2.0.

A diferencia de otros estándares de comunicación, en USB se necesitan muchas más reglas para la transferencia y recepción de datos. Dichas reglas pueden encontrarse en las especificaciones generales de la USB IF.

Empero, dado que agrupa muchas ventajas, se ha convertido en el primer estándar de comunicación para un sinfín de periféricos informáticos.

En el presente proyecto, debido a que el MCU no soporta ciertas características de la versión USB 2.0, sólo se muestran algunas pocas de ésta, pero se analiza con mayor detalle la versión 1.1.

2.4.1 Características físicas y electrónicas del estándar USB

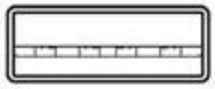
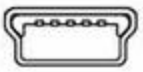
La especificación USB define que un solo dispositivo podrá controlar la comunicación (host) y los demás serán denominados simplemente dispositivos o esclavos.

El host debe permitir la alimentación de voltaje de los dispositivos conectados a él, con 5 V. Cada puerto sólo permite una salida máxima de 500 mA.

El host debe tener un puerto físico de tipo hembra, de tipo A, aunque en la versión 2.0 se incluyeron nuevos conectores para que un dispositivo USB pueda actuar en modo host de ser necesario. Esta flexibilidad es llamada *Sobre-La-Marcha* (OTG).

El dispositivo deberá tener también su propio conector, de tipo B o mini-B. Para unirlos, se requiere un cable USB de con conectores de tipo macho.

Tabla 2.7 Tabla de conectores USB más comunes.

Tipo	Esquema	Figura
Tipo A	4.5mm x 12.0mm 	
Tipo B	7.3mm x 8.5mm 	
Mini-B	3.0mm x 6.8mm 	

El cable USB se compone de 4 hilos o líneas, dos de los cuales, llamados VBUS y GND proporcionarán el voltaje requerido y la señal eléctrica de referencia. Los otros dos cables, denominados DATA+ y DATA– (o simplemente D+ y D-), se encargarán de transferir la información.

D+ y D- son los hilos que transmitirán la información por medio de señales diferenciales entre ellos, como muestra la gráfica de voltaje en los hilos D+ y D- de la figura 2.19.

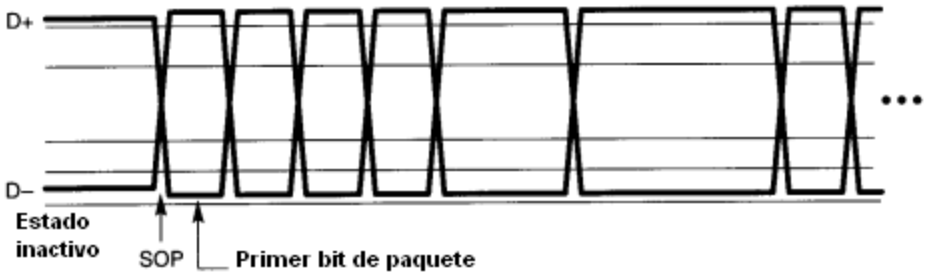


Figura 2.19 Campo de bits USB representando un *Start Of Packet* (SOP) o de sincronización (SYNC).

El host tiene la capacidad de enviar información a básicamente todos los dispositivos conectados al bus por medio de redireccionadores llamados hub (que convierte a USB en una topología estrella clásica). Teóricamente, el bus permitiría la conexión de 127 dispositivos.

2.4.2 Transferencias y paquetes de datos en USB

USB manda secuencialmente bits de información a través de los hilos D+ y D-. Los bits conformarán bytes que a su vez, son parte de grandes conjuntos que se denominan *paquetes*.

La codificación de los bytes de los paquetes incluye la técnica *NZRI* y de *bit-stuffing*.

En USB los bytes transmitidos primero son los menos significativos.

La versión USB 1.1 maneja dos velocidades de transferencia de datos de los dispositivos, llamadas *Low Speed* y *Full Speed*. Low Speed solo maneja velocidades de hasta 1.5 Mbps, Full Speed de hasta 12 Mbps. USB 2.0 incluye la velocidad *High Speed*, de hasta 480 Mbps.

Los dispositivos de USB 1.1 pueden interactuar con host creados con la especificación 2.0. El host puede reconocer la velocidad de los dispositivos detectando un cambio de potencial derivado de dos resistores en serie, conectados a la línea D+ o D-. La figura 2.20 muestra un dispositivo Full-Speed, el resistor estaría conectado en D- en caso de uno Low-Speed.

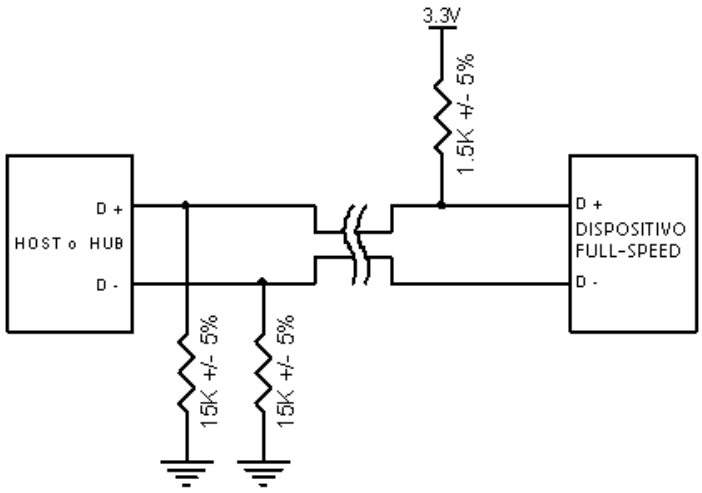


Figura 2.20 Resistores de pull-up que identifican la velocidad máxima del dispositivo.

En base a estas velocidades, pueden segmentarse *tiempos* en el bus USB. Cada uno de estos tiempos es denominado *frame* (trama). Un *frame* en USB 1.1 es de 1 ms, en High Speed son llamados *microframes* y duran 125 µs.

Dentro de estos *frames*, puede haber un número cualesquiera de intercambio de información entre el host y el dispositivo, denominados *transferencias*.

Cada transferencia siempre contiene paquetes. En USB una transferencia típica consiste de tres paquetes, *Token*, *Data* y *Handshake*. La tabla 2.8 los describe con detalle.

Todos los paquetes tienen campos de bytes que indican ciertas condiciones. Los campos tienen estructuras igualmente estandarizadas. Se ilustran en la figura 2.21.

Los paquetes *Token* indican una nueva transferencia, e inician siempre con el campo de bytes SYNC (sincronización). A continuación, el campo ID indica qué tipo de transferencia es (IN, OUT, SETUP). Para saber a *cuál de todos los dispositivos conectados* será dirigida esta transferencia, se requiere el siguiente campo, el de dirección (ADDRESS). Cada dispositivo tiene una serie de *endpoints,* y para saber a cuál endpoint se dirige esta transferencia, se especifica en el siguiente campo (ENDPOINT). El campo CRC (Cyclic Redundancy Check) es una primera forma de saber que los datos anteriores del paquete fueron correctos, consta de 5 bytes. Finalmente, un paquete finaliza siempre con el campo End-Of-Packet (EOP).

Los paquetes *Data* (DATA1, DATA0) difieren a los *Token* porque no incluyen ya los campos ADDRESS ni ENDPOINT. El paquete Data incluirá la verdadera información que manda, ya sea el host o el dispositivo. Su campo CRC tiene 16 bytes.

Los paquetes *Handshake* solamente se componen por los campos SYNC, ID y EOP.

Los paquetes Start-Of-Frame se componen de los campos SYNC, ID, *Frame Number* (indica el número de *frame* que inicializa el host), CRC y EOP.

Tabla 2.8 Tipos de paquetes típicos en una transferencia USB.

ID de paquete	Tipo	Función
TOKEN	IN	Indica que el host está solicitando datos del dispositivo.
	OUT	Indica que el host está mandando o está por mandar datos al dispositivo.
	SETUP	Indica que esta transferencia tendrá internamente una o más transferencias de tipo IN u OUT.
DATA	DATA0	En cualquiera de estos paquetes se encuentra información contenida. Es información que el fabricante emplea para que el dispositivo ejecute sus diversas funciones para las que fue creado.
	DATA1	
	DATA2 *	
	MDATA*	
HAND-SHAKE	ACK	Los datos y todos los paquetes cabecera USB (URB) han sido leídas o recibidas exitosamente por el host o por el dispositivo.
	NAK	La información fue recibida con errores, por lo que pide volver a hacer la transferencia. En transferencias de tipo *Interrupt*, indica que no hay datos actualmente por enviarse.
	STALL	El dispositivo está "confundido" y se ve imposibilitado de continuar la transferencia. El dispositivo necesita intervención del host para salir de este estado.
START-OF-FRAME (SOF)		No contiene una dirección específica, pues se manda a todos los dispositivos. Indica que se ha iniciado un nuevo frame. Evitan también que los dispositivos entren en estado de suspensión (SUSPEND).

* Sólo soportados a partir de la versión 2.0

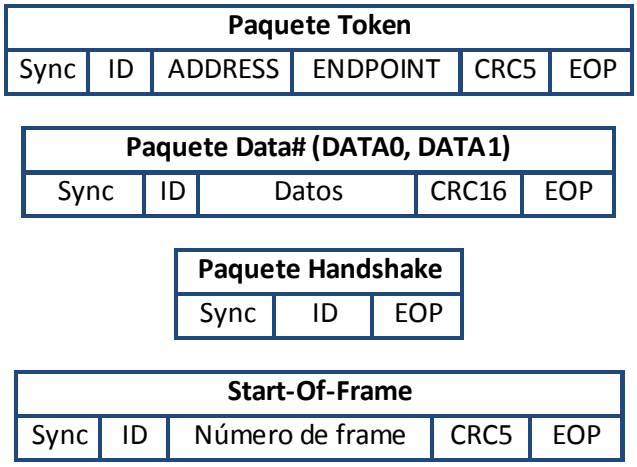


Figura 2.21 Composición de los bytes de los diferentes paquetes USB.

2.4.3 Características lógicas de los dispositivos USB

Un dispositivo USB contiene internamente campos lógicos denominados *descriptores*: descriptor de dispositivo, descriptores de *configuración*, descriptores de *interface*, descriptores de *endpoint* y descriptores de cadena.

El descriptor del dispositivo indicará, precisamente, el funcionamiento general del dispositivo (mouse, teclado, impresora). Contiene además números especificados por el

estándar USB, que son números reservados de clase, subclase del dispositivo y tipo de protocolo. Estos números permiten en ocasiones ejecutar tareas comunes del dispositivo.

Unos ejemplos claros son los teclados, con clase USB 0x03. Aun sin instalarse un *controlador*, muchas computadoras pueden interactuar con un nuevo teclado puesto que su SO incluye un controlador general que actúa sobre dispositivos que posean estos parámetros en su descriptor de dispositivo.

Dos campos aún más importantes contenidos en este descriptor son el PID (Product IDentifier) y VID (Vendor IDentifier), vitales para muchas funciones de la computadora.

El descriptor de dispositivo debe incluir además el número de *configuraciones* que se pueden emplear con él. Todos los dispositivos USB tienen *configuraciones* e *interfaces*. Una *configuración* es un panorama general para hacer funcionar al dispositivo, normalmente para seleccionar la forma de alimentación que requerirá: o tiene su propia fuente de alimentación de voltaje o se energiza a través del bus. Aunque el estándar permite al host cambiar la configuración mientras trabaja con los dispositivos, son muy pocos los fabricantes que emplean esta característica.

Una *interface* son las formas en las que el dispositivo puede funcionar. Por ejemplo, existen ratones de PC (mouse) que incluyen además la posibilidad de almacenar información dentro de ellas, a semejanza de las memorias flash portátiles. La posibilidad de funcionar tanto como memoria o mouse es un ejemplo de interfaces diferentes.

Una interface contiene uno o varios *endpoints*, que son descritos como búferes dentro del dispositivo, y se identifican con un número hexadecimal. Una interfaz solamente puede contener 15 endpoints, además del Endpoint 0, del Endpoint 1 al Endpoint 15.

Un endpoint puede ser "de entrada", para transferencias IN. De acuerdo a los números de identificación en USB, comienzan siempre con 8. Por ejemplo, el 0x81 sería el Endpoint 1 y es de tipo IN.

Los endpoints pueden ser también "de salida", para transferencias OUT. Comienzan con el número 0. El endpoint 0x08 es el Endpoint 8 y es de tipo OUT.

Los endpoints también se clasifican por las cuatro transferencias permitidas en USB:

- *Bulk* (masivos): La información que se envíe o reciba por ellos no requiere ser atendida prontamente. A cambio, se puede mandar una larga cantidad de información en un solo paquete Data. Ocupan el resto del *frame* que no fue requerido por las otras transferencias.
- *Interrupt* (interrupción): Manejan información tan importante, que cada *frame* tiene un espacio reservado para ellas. No puede emplearse con ellos mucha información en una sola transferencia.
- *Isochronous* (isócronos): La información requiere mandarse en tiempo real, que incluso no se emplean los paquetes de Handshake.

- Control: El host requiere mandar o recibir información, vital para hacer funcionar al dispositivo, por lo que son más importantes que las anteriores. En cada *frame* tendrán un tiempo reservado. Este tipo de transferencia tiene un paquete inicial SETUP (un paquete *token* con ID SETUP) y hace referencia a otras transferencias IN u OUT siguientes.

Los paquetes SETUP serán dirigidos al Endpoint 0, que por norma del estándar USB deben tenerlo implementado todos los dispositivos.

Los últimos descriptores, los de cadena, son mensajes que permitirían al usuario conocer acerca del dispositivo conectado, como su procedencia, fabricante, modelo, etc. En ambientes Windows es común observar estos mensajes cada vez que se conecta un dispositivo USB por primera vez en esa computadora.

- 2.4.3.1 Transferencias de control típicas

Luego del paquete Token con ID SETUP, continúa el paquete Data, que definirá el tipo de solicitud (REQUEST) que se está haciendo al dispositivo. Este paquete Data contiene 8 bytes que definen el uso que se le dará a las transferencias IN u OUT posteriores. Estos 8 bytes tienen campos de bits cuyo significado se muestra en la tabla 2.9.

Tabla 2.9 Valores que incluyen exclusivamente las transferencias SETUP.

Offset	Campo	Tamaño (bytes)	Valor	Descripción
0	**bmRequestType**	1	Combinación de bits	D7 *Dirección de transferencia de la fase Data* 0 = Host a dispositivo (SET) 1 = Dispositivo a host (GET)
				D6… D5 *Tipo* 0 = Estándar 1 = Clase 2 = Vendedor 3 = Reservado
				D4…. D0 *Recipiente* 0 = Dispositivo 1 = Interface 2 = Endpoint 3 = Otro 4 … 31 = Reservado
1	**bRequest**	1	Valor	REQUEST
2	**wValue**	2	Valor	Valor
4	**wIndex**	2	Índice u offset	Índice
6	**wLength**	2	Conteo	Número de bytes para transferir si existe una fase Data

Existen varios REQUEST típicos en USB, mostrados en la tabla 2.10.

Tabla 2.10 REQUEST (solicitudes) establecidas más comunes del host al dispositivo.

	bmRequest Type	bRequest	wValue	wIndex	wLength	Data
DISPOSITIVO	1000 0000b	GET_STATUS (0x00)	Cero	Cero	Dos	Estado del dispositivo
	0000 0000b	CLEAR_FEATURE (0x01)	Selector de característica	Cero	Cero	Ninguno
	0000 0000b	SET_FEATURE (0x03)	Selector de característica	Cero	Cero	Ninguno
	0000 0000b	SET_ADDRESS (0x05)	Dirección de dispositivo	Cero	Cero	Ninguno
	1000 0000b	GET_DESCRIPTOR (0x06)	Tipo de descriptor e índice (index)	Cero o ID de idioma	Tamaño del descriptor	Descriptor
	0000 0000b	SET_DESCRIPTOR (0x07)	Tipo de descriptor e índice (index)	Cero o ID de idioma	Tamaño del descriptor	Descriptor
	1000 0000b	GET_CONFIGURATION (0x08)	Cero	Cero	1	Valor de configuración
	0000 0000b	SET_CONFIGURATION (0x09)	Valor de configuración	Cero	Cero	Ninguno
INTERFACE	1000 0001b	GET_INTERFACE (0x0A)	Cero	Inter-face	Cero	Configuración alterna
	1000 0001b	SET_INTERFACE (0x11)	Configuración alterna	Inter-face	Cero	Ninguno

Un ejemplo de la combinación de los bytes mostrados en las tablas 2.10 y 2.9 puede ser un paquete SETUP que tiene un paquete Data como el siguiente:

00 05 01 00 00 00 00 00

Se procede a dividirlo, como se muestra a continuación:

bmRequestType	**bRequest**	**wValue**	**wIndex**	**wLength**
0x00	0x05	0x01 0x00	0x00 0x00	0x00 0x00

El campo bmRequestType se representa en binario:

0x00 = 0000 0000b

Dividiendo esta cantidad binaria, se obtiene la naturaleza del paquete SETUP:

Dirección	**Tipo**	**Recipiente**
0	0 0	0 0 0 0 0
SET	STANDARD	DEVICE

El bRequest es 0x05, que según la tabla 2.10 es SET_ADDRESS. El campo wValue es 0x0001 (USB transmite primero los bytes menos significativos). Los campos wIndex y wLength no son necesarios, son puestos en 0. La oración queda completa entonces:

00 05 01 00 00 00 00 = SET ADDRESS 0x01 TO DEVICE (Fijar dirección 0x01 al dispositivo)

SET_ADDRESS es un proceso estándar, de ahí que sea especificado en el campo de bits TIPO del byte bmRequestType.

En particular, este ejemplo ayuda a comprender la información expuesta a continuación.

2.4.4 El proceso de enumeración

El proceso de *enumerar* a un dispositivo consiste en otorgar un número identificador (ADDRESS) a un nuevo dispositivo conectado al bus, de tal forma que en lo sucesivo, el host pueda mandar transferencias dedicadas exclusivamente a dicho dispositivo, empleando el número en el campo ADDRESS de los paquetes Token.

Windows enumera al dispositivo luego de identificar al dispositivo. GNU/Linux lo hace en forma inversa.

El host identifica al dispositivo solicitándole sus descriptores.

Una enumeración típica de un dispositivo en GNU/Linux lo muestra la tabla 2.11.

Lo que sigue a continuación dependerá de las múltiples tareas que estén definidas dentro del controlador del dispositivo. En ocasiones, el controlador sólo indica al dispositivo que espere determinados eventos por parte del usuario para trabajar, o bien, manda una cantidad muy grande de información para proceder con el correcto funcionamiento de él.

Tabla 2.11 El proceso de enumeración de un dispositivo.

PASO	REQUEST & RESPONSE	FUNCIÓN
1	SET ADDRESS	El host detecta el cambio de voltaje en D+ o D-, por lo que intenta asignarle un ADDRESS al posible dispositivo. Todo dispositivo no enumerado debe responder ante cualquier paquete con ADDRESS igual a cero.
2	GET DEVICE DESCRIPTOR	El host pide al dispositivo su descriptor, donde se induye el PID, VID y el número serial del dispositivo. En este descriptor se incluye también el número de configuraciones que maneja el dispositivo.
3	GET CONFIGURATION DESCRIPTOR	Se pide al dispositivo los descriptores de todas las configuraciones que posea. Por cada una de ellas pedirá además el número de interfaces.
4	GET INTERFACE DESCRIPTOR	Por cada interface el host pedirá un descriptor. Dentro de cada interface habrá una lista de endpoints que se empleen. Cada descriptor de endpoint definirá su número y dirección, su tipo de transferencia que emplea (Bulk, Interrupt, etc.) y el intervalo en el cual se manejan los de tipo Interrupt e Isochronous.
5	GET STRING DESCRIPTOR	El host pide información que pueda presentar al usuario. Comúnmente los dispositivos muestran su nombre, su fabricante y su función.
6	SET CONFIGURATION	El host, de tener un controlador instalado en el sistema operativo para la clase y subclase USB o al PID y VID del dispositivo, selecciona la configuración más apropiada y se lo indica al dispositivo. En caso de tener solamente una configuración, aun así debe especificársela.

2.4.5 Transferencias típicas en USB

El tipo de transferencia se determina de acuerdo al endpoint objetivo, sea Bulk, Interrupt, Isochronous o de control.

El host siempre iniciará las transferencias. Primero, debe indicar al dispositivo qué tipo de transferencia será la que inicie (IN, OUT, SETUP) determinada en el paquete Token. El mismo paquete indicará la cantidad de información (bytes) que el host espera recibir o mandar.

Los paquetes Data serán realizados por el dispositivo, en caso de que el paquete Token haya sido un IN. En caso de un OUT, el host será quien los elabore.

El paquete Handshake será hecho por el host, en caso de un previo paquete IN. En caso de un OUT, será hecho por el dispositivo. El paquete Handshake no se realiza cuando se trabaje con un endpoint de tipo Isochronous.

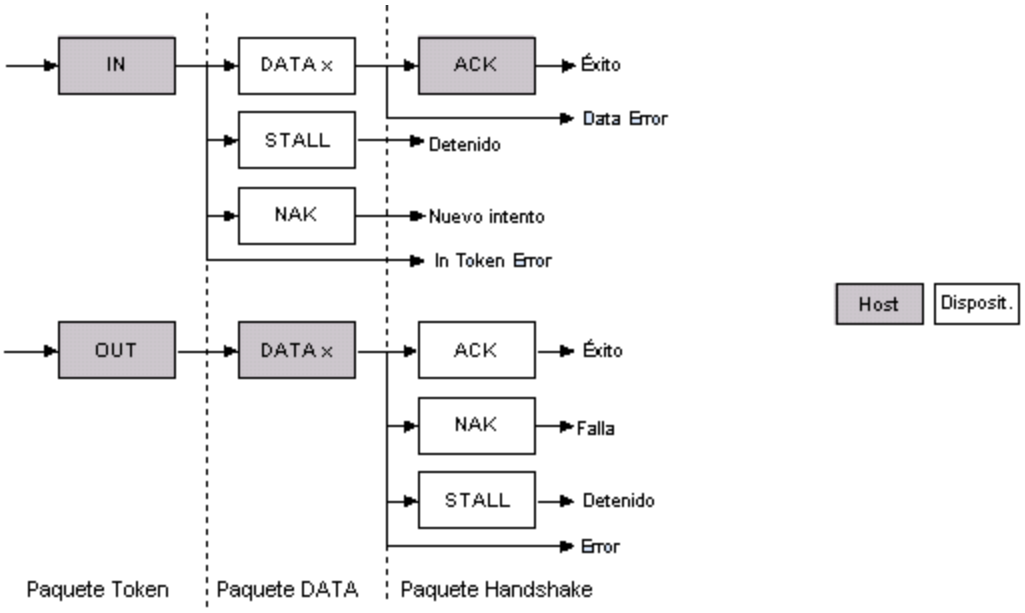


Figura 2.22 Transferencia típica USB.

Cuando la información excede el máximo de bytes (en los paquetes Data) permitidos por transferencia, se requiere de otras más hasta mandar la información deseada.

El máximo de bytes que se pueden incluir en los paquetes Data depende de la versión USB y del tipo de transferencia.

Tabla 2.12 Cantidad máxima de bytes por paquete Data.

Transferencia	Bytes máximos en paquetes Data		
	Low Speed	Full Speed	High Speed
Bulk	(no soportado)	64	512
Control	8	64	64
Interrupt	8	64	1024
Isochronous	(no soportado)	1023	1024

Por consiguiente una transferencia IN exitosa será cuando el host ha recibido todos los bytes del paquete Data creado por el dispositivo y responde con un ACK. Una transferencia OUT exitosa implicaría que el dispositivo ha contestado con ACK ante los paquetes Data creados por el host (salvo en las transacciones Isochronous).

2.5 USB Host Stack de Microchip

El módulo USB *OTG* de los microcontroladores PIC32 les permiten actuar como host o esclavo, o ambas (OTG). No soporta el modo High Speed.

El USB Stack de Microchip ha sido diseñado para automatizar en gran parte el uso de las funciones USB, de tal forma que sólo es necesario ocuparse de conocer la información que

se enviará al dispositivo (escritura) o bien, se recibirá de él (lectura). La forma de hacerlo es reservar cierto espacio en memoria RAM y determinar el tamaño de esta reservación, tanto para recibir o enviar datos (bytes) al dispositivo.

Empleando al MCU como host, Microchip recomienda visualizar el uso del USB Stack como una primera *capa* o *nivel*, de cualquier proyecto a desarrollar, empleando MPLAB.

Para garantizar el uso correcto de esta capa, se debe agregar una segunda, semejante a los controladores de un dispositivo en un sistema operativo de una PC. A esta capa se le llama *controlador cliente*.

Finalmente, las diversas acciones que suceden dentro del controlador cliente servirán para la ejecución de las tareas propiamente deseadas con el microcontrolador. A esta tercera capa se le denomina *capa de aplicación*.

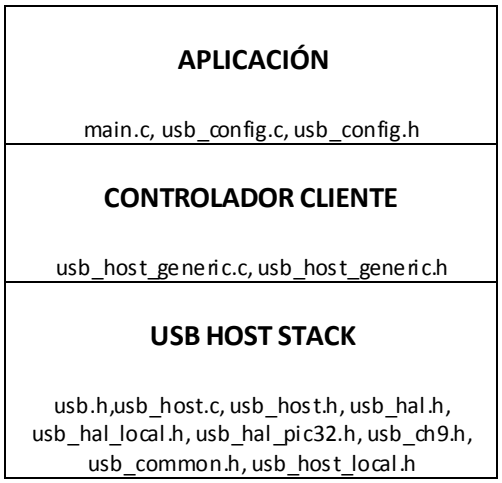


Figura 2.23. Forma de representar el uso del USB Stack y el controlador genérico de Microchip con sus archivos representativos.

2.5.1 Controladores cliente de Microchip

Microchip clasifica a los controladores de dos formas: por monitorización y por eventos. A grandes rasgos, un controlador cliente por monitorización siempre requerirá obtener información de la primera capa (propiamente del bus) una vez que termine de hacer determinada tarea. En los controladores por-eventos, la aplicación es notificada por la segunda capa automáticamente de algunos eventos de relevancia. Este tipo de controlador, sin embargo, requerirá de una ligera carga adicional en la memoria flash del MCU.

Para facilitar aún más el empleo del USB Stack, Microchip ha desarrollado diversos controladores optimizados para diversas clases de dispositivos USB, como los HID, almacenamiento masivo, impresora, y un controlador genérico. Este último se creó especialmente para dispositivos que pertenezcan a la clase 0xFF, Específica de Vendedor.

El controlador genérico está inmerso en dos archivos, *usb_host_generic*.h y *usb_host_generic*.c. Pueden emplearse tanto en forma de monitorización o por-eventos.

Para continuar, cabe resaltar que a las transferencias OUT las denomina *de escritura* (TX) y a las de tipo IN, *de lectura* (RX).

2.5.2 Empleo de diversos controladores con PIC32

Puesto que prácticamente cualquier dispositivo puede conectarse al módulo USB OTG de los PIC32, Microchip creó un método para saber si el MCU puede funcionar con dicho dispositivo. Este método emplea la llamada TPL (Targeted Peripheral List), que es en esencia una lista de VID y PID. En cuanto se conecta físicamente un dispositivo USB, el MCU verificará que existan su VID y PID en el TPL. En caso de existir, la lista indica *con cuál controlador funcionará*. El PIC inicializará una función inmersa en dicho controlador, cuyo nombre sigue una sintaxis como la siguiente:

<Controlador_de_dispositivo_X>*Init*

La TPL es versátil porque también puede esperar asignar controladores a dispositivos que pertenezcan a alguna misma clase USB. Dos teclados de marcas diferentes pueden emplear un mismo controlador, siempre y cuando su descriptor de dispositivo contenga el mismo número de clase que la TPL tiene registrado.

La TPL permite por tanto, que el MCU tenga grabados en su memoria diferentes controladores y permitir que más de un dispositivo USB pueda funcionar con éste.

2.5.3 Aplicación empleando controlador genérico

La aplicación (*main*.c en la mayor parte de los proyectos creados por Microchip) debe incluir una sola llamada a la función `USBHostInit`, inmersa en el archivo *usb_host*.c.

A continuación, la aplicación necesariamente tiene que incluir un bucle que continuamente llame a la función `USBHostTasks`.

La función es muy importante, porque de acuerdo al IRQ recibido del módulo USB del MCU, hará que avance la máquina de estados del bus, para que el software emplee al bus correctamente.

La máquina de estados consiste básicamente en los estados `DETACHED`, `ATTACHED`, `ADDRESSING`, `CONFIGURING`, `RUNNING` y `HOLDING`.

Tabla 2.13. Representación con una máquina de estados por Microchip simulando los estados del bus.

ESTADO	DESCRIPCIÓN	SUB ESTADOS
DETACHED	Es el estado inicial del bus. Se encarga de esperar la conexión de un dispositivo. Si un dispositivo es desconectado, el módulo es apagado y se libera toda la memoria ocupada.	INITIALIZE WAIT_FOR_POWER TURN_ON_POWER WAIT_FOR_DEVICE
ATTACHED	El bus notifica que un dispositivo es conectado. Se resetea al dispositivo, se espera a que concluyan tiempos de seguridad (settling) y se manda el primer paquete esencial de la enumeración USB, GET-DEVICE-DESCRIPTOR. El dispositivo debe responder, y de acuerdo a la dase USB que indique o de su PID y VID, se le asigna un controlador en caso de existir en la memoria del MCU.	SETTLE WAIT_FOR_SETTLING SETTLING_DONE RESET_DEVICE GET_DEVICE_DESCRIPTOR_SIZE WAIT_FOR_GET_DEVICE_DESCRIPTOR_SIZE GET_DEVICE_DESCRIPTOR_SIZE_COMPLETE GET_DEVICE_DESCRIPTOR WAIT_FOR_GET_DEVICE_DESCRIPTOR GET_DEVICE_DESCRIPTOR_COMPLETE VALIDATE_VID_PID
ADDRESSING	Consiste en mandar la transacción de control con el paquete de SET-ADDRESS, y esperar la respuesta.	SET_DEVICE_ADDRESS WAIT_FOR_SET_DEVICE_ADDRESS SET_DEVICE_ADDRESS_COMPLETE
CONFIGURING	El MCU debe mandar los paquetes de GET-CONFIGURATION- DESCRIPTOR-SIZE y GET-CONFIGURATION-DESCRIPTOR. De acuerdo a las respuestas selecciona la configuración USB del dispositivo adecuada y manda el paquete SET-CONFIGURATION. En cuanto recibe la respuesta del dispositivo intenta llamar a la función Init del controlador. Este estado guarda los parámetros principales del dispositivo, como la lista de las interfaces y de los endpoint con sus atributos propios y otros más que permitirán el manejo adecuado de ellos.	INIT_CONFIGURATION GET_CONFIG_DESCRIPTOR_SIZE WAIT_FOR_GET_CONFIG_DESCRIPTOR_SIZE GET_CONFIG_DESCRIPTOR_SIZECOMPLETE GET_CONFIG_DESCRIPTOR GET_CONFIG_DESCRIPTOR_COMPLETE SELECT_CONFIGURATION * SEND_SET_OTG * WAIT_FOR_SET_OTG_DONE * SET_OTG_COMPLETE SET_CONFIGURATION WAIT_FOR_SET_CONFIGURATION SET_CONFIGURATION_COMPLETE INIT_CLIENT_DRIVERS
RUNNING	Si los estados anteriores se realizaron correctamente, este es el estado final: el dispositivo ya ha sido enumerado, se le asignó un controlador y está completamente listo para funcionar. Este estado no hace más que esperar un IRQ para avanzar hacia uno nuevo. Sin embargo también controla los estados de suspensión y reanudación de un dispositivo en caso de ser necesario. Permite que todas las funciones que requieran interactuar con el dispositivo tengan validez a nivel aplicación. Mientras, el módulo USB del MCU puede ejecutar todas las funciones de bajo nivel (mandar paquetes SOF, energizar el puerto, etc).	NORMAL_RUN SUSPEND_AND_RESUME SUSPEND RESUME RESUME_WAIT RESUME_RECOVERY RESUME_RECOVERY_WAIT RESUME_COMPLETE
HOLDING	Si algún estado anterior no pudo ser llevado a cabo por alguna circunstancia, el bus permanece en este estado y no se llevan a cabo las tareas de bajo nivel del módulo del µC. El único estado que acepta para cambiar es el DETACHED (desconectar físicamente al dispositivo).	HOLD_INIT HOLD

El archivo *usb_host_generic.c* tiene incluida la función USBHostGenericInit. Esta se encarga de recuperar la dirección asignada al dispositivo, el número de controlador asignado para dicho dispositivo y su descriptor.

Este controlador tiene algunas características importantes:

- Virtualiza al dispositivo como una estructura de datos definida por Microchip (GENERIC_DEVICE), con atributos como el PID, VID y unas banderas de tipo bit, denominadas rxBusy y txBusy.
- Asume que sólo se emplearán dos endpoints, además del Endpoint 0: el Endpoint 0x81 y el 0x01.

Como solamente se pueden emplear 2 endpoints, uno de salida y otro de entrada, el controlador puede simplemente manejar la recepción y transmisión de datos cerciorándose que las banderas rxBusy y txBusy no estén igualadas a 1.

Cuando el host recibe un ACK ante paquetes de tipo OUT y corrobora que no tenga más datos que mandar, o que ante un IN haya completado el número de bytes que esperaba recibir, puede limpiar las banderas y hacer posible invocar nuevamente las funciones de mandar o recibir datos por USB.

Tanto para leer o escribir al dispositivo, se requiere un búfer. El búfer puede ser una matriz, vector, estructura. Se permite la facilidad que sea un apuntador al primer elemento de dicho búfer.

Para mandar paquetes OUT y el búfer con información al Endpoint 0x01 se realiza por medio de la función USBHostGenericWrite. Para mandar paquetes IN al Endpoint 0x01, y asignar un búfer al cual recibir los datos leídos, es por medio de la función USBHostGenericRead.

Dichas funciones solamente verifican que el MCU no esté esperando algún paquete de Handshake tras haber iniciado una transferencia OUT (o un paquete Data en una transferencia IN) y que no se esté tratando de manipular información con los endpoints mientras exista algún estado inválido (como el estado del bus HOLDING). Al superarse estas verificaciones, puede invocarse a la función de la primera capa, en *usb_host.c*, llamadas USBHostWrite o USBHostRead, de acuerdo al endpoint del dispositivo que se desee controlar.

Dentro de estas funciones se invoca a la función, dentro del mismo archivo, llamada _USB_FindEndpoint. Lo que esta función devuelve es toda la información del endpoint (en una estructura definida por Microchip como USB_ENDPOINT_INFO) que se obtuvo durante los estados CONFIGURING y ATTACHED. Es quizá lo más importante de esta información el tipo de transferencia USB de este endpoint.

_USB_FindEndpoint es en esencia, una función que facilita que el usuario maneje la información de los paquetes Data, sin tener que recordar los atributos USB del endpoint.

Con la información obtenida se sabe si el endpoint está listo o no para iniciar una transferencia (es decir, que actualmente no esté esperando información o paquetes Handshake del dispositivo, que el endpoint exista, entre otras cosas). En caso de superarse estas verificaciones, USBHostRead o USBHostWrite invocan respectivamente a las funciones _USB_InitRead o _USB_InitWrite.

Estas funciones configurarán la información del endpoint (la estructura del endpoint USB_ENDPOINT_INFO), para prepararlo ante una nueva transferencia. A manera de ilustración se ejemplifica con la función _USB_InitWrite, sin embargo sucede de manera similar para _USB_InitRead.

Lo importante de estas funciones es el valor que se asigna al atributo transferState de la estructura USB_ENDPOINT_INFO. En el caso de _USB_InitWrite, el módulo USB OTG simplemente verificará la lista de las estructuras de todos los endpoints (siempre que la máquina de estados del bus se encuentre en estado RUNNING). Si el módulo comprueba que algún endpoint tiene igualado este atributo a TSTATE_BULK_WRITE, TSTATE_ISOCHRONOUS_WRITE o TSTATE_INTERRUPT_WRITE, configurará al bus, creará los paquetes *Token* correctos y mandará adecuadamente la información por el bus con la información del búfer asignado. A su vez, esperará el correspondiente paquete de Handshake del dispositivo para determinar si la transferencia fue exitosa o no. En caso de recibir un paquete NAK, automáticamente el módulo reintentará hacer la transferencia.

De recibirse el paquete ACK ante una transferencia OUT, un IRQ notificará a la primera capa que un evento USB (EVENT_TRANSFER) ha sucedido. A su vez (y si es un controlador por-eventos el que se emplea), la primera capa notificará a la segunda (específicamente a la función USBHostGenericEventHandler). También le notificará en cuál endpoint fue hecha la transferencia.

Esta función determina si el endpoint en el que sucedió la transferencia fue el 0x01 (predeterminado en el controlador genérico). De ser así limpia la bandera txBusy de la estructura del dispositivo, y con ello permitir una nueva escritura.

Como su nombre lo indica, USBHostGenericEventHandler es un manejador de eventos, e incluye también la posibilidad de recibir de otros tipos, como el USB_DETACH, el evento que indica que el dispositivo ha sido físicamente desconectado del bus. Básicamente, este manejador de eventos *filtra* aquellos eventos USB que son de utilidad de los que no lo son, por ejemplo, no notifica a la aplicación que una transferencia OUT hacia un endpoint, que no sea el 0x01, ha sido completada.

Los eventos que son aceptados por el manejador del controlador son redirigidos hacia el manejador de eventos de la aplicación, cuyo nombre es por definición

USB_HOST_APP_EVENT_HANDLER, aunque puede redefinirse su nombre en el archivo *usb_config*.h. Es realmente éste quien da resultados finales de importancia como: el endpoint del dispositivo ha aceptado la totalidad de una transferencia OUT, el dispositivo esperado ha sido conectado físicamente, entre otros. Se muestra de manera gráfica en la figura 2.24, en donde el color amarillo son las funciones de la capa de aplicación, en anaranjado las del controlador y en verde las del USB Host Stack.

En el caso de una transferencia IN, el módulo USB busca en el atributo transferState de los endpoints igualado a TSTATE_BULK_READ, TSTATE_INTERRUPT_READ o TSTATE_ISOCHRONOUS_READ. Entonces configura al bus y crea el paquete *Token* IN, y se dedica a esperar la llegada de los paquetes Data necesarios. Cuando se recibe el número de bytes especificados con el parámetro size de la función USBHostRead, envía un ACK al bus, y un IRQ a la primera capa (EVENT_TRANSFER).

La primera capa su vez, informa a la segunda. El controlador, con su función USBHostGenericEventHandler, determinará si el endpoint que se está reportando es el 0x81. Así puede saber que el Endpoint 0x81 ha mandado al host la cantidad de información deseada. Por consiguiente, limpiará la bandera rxBusy y notificará a USB_HOST_APP_EVENT_HANDLER.

De esta manera, la aplicación podría manipular la información obtenida en el búfer especificado en USBHostGenericRead.

Para un controlador por monitorización, a grandes rasgos se requiere continuamente revisar al USB Stack y ejecutar igualmente tareas del controlador (USBHostGenericTasks). De acuerdo a la información que se tenga, el controlador la guarda y a su vez, permite que la aplicación tenga al alcance dicha información.

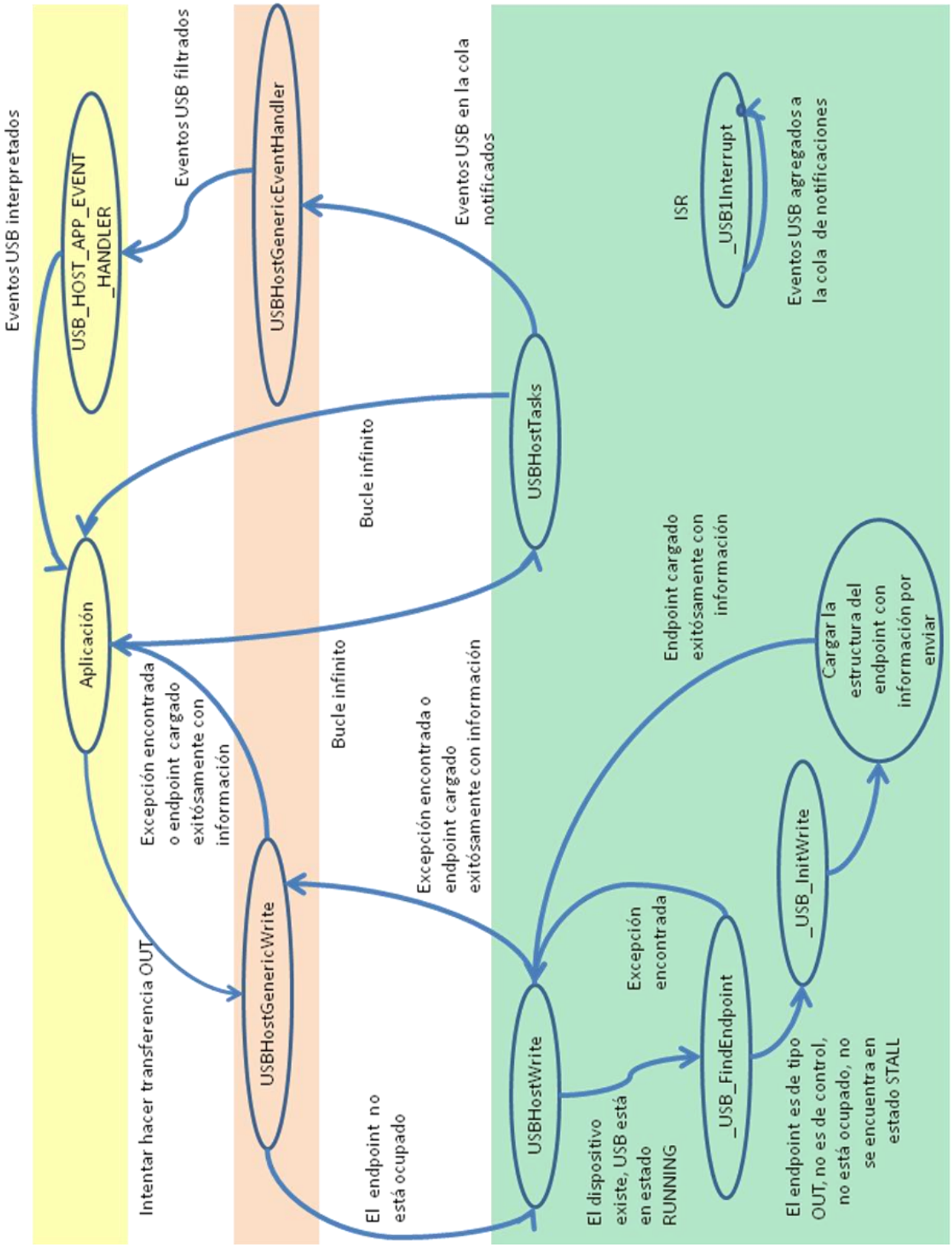


Figura 2.24 Máquina de estados simplificada de un proceso típico de una transferencia OUT con un controlador genérico por-eventos.

Capítulo 3

Desarrollo de la comunicación inalámbrica

En el presente capítulo se expone la forma en la que se intentó realizar la comunicación inalámbrica por medio de la hipótesis propuesta. Se retoman algunos conceptos explicados en el capítulo anterior, y la manera en que se emplearon para resolver las problemáticas y diversas maneras de emplear herramientas que se requirieron durante el desarrollo.

La hipótesis del capítulo 1, en resumen, indica que se replicarán los paquetes USB que una computadora manda a la tarjeta de red.

Debido a las nulas especificaciones técnicas de los fabricantes de las dos tarjetas de red expuestas en la problemática del capítulo 1 del presente trabajo, no se puede obtener información oficial del funcionamiento del tráfico USB entre ellas y una PC.

Esto hace que surjan múltiples interrogantes. Puede ser que algunos paquetes no requieran ser replicados, probablemente los paquetes USB tengan alguna clase de codificación, o los paquetes de asociación en el bus sean suprimidos para permitir llanamente la información más esencial. Esto sólo por mencionar algunas.

La única manera de verificar esto es observar el tráfico USB entre una computadora y la tarjeta de red.

3.1 Selección del analizador de protocolo

Un analizador de protocolo descifra, filtra y muestra información de tráfico en los diversos periféricos de una computadora. Existen para interfaz de red, audio, video, etc. En el presente proyecto se utiliza un analizador de protocolo USB.

Existen diversos tipos de analizadores de tráfico USB y cada uno de ellos muestra a diferentes grados de detalle los paquetes USB. Algunos podrían inclusive generar información sobre el bus.

Existen dos tipos:

Sólo software: Se instalan en la PC y muestran el tráfico USB enviado y recibido por dicha PC. Los analizadores de sólo software muestran información del mismo controlador de host pero no pueden mostrar información a bajo nivel del bus. Existen de licencia temporal (*shareware*), *demo* o gratuita (*freeware*).

Software-hardware: Además de un software adicional usa dispositivos físicos para monitorear el tráfico en cualquier momento de un puerto USB. Puede mostrar incluso errores de CRC, los paquetes NAK, etc. (información a bajo nivel). Todos ellos tienen un precio monetario.

Debido a que se desea que los costos del proyecto sean bajos, se elegirá uno de los primeros, haciéndose una comparativa en la tabla 3.1.

Tabla 3.1. Comparación entre diversos analizadores de protocolo USB de sólo software

Analizador	Características	Desventajas
USBlyzer	Es de prueba por 30 días. Funciona en Windows. Documentación suficiente.	No diferencia entre transferencias Interrupt de las de Bulk. La interfaz puede llegar a ser complicada de manejar.
Device Monitor	Es gratuito. Funciona en Windows. Interfaz muy amigable.	No muestra el tráfico de un puerto sin ningún dispositivo, lo que significa que no se puede ser lo suficientemente rápido para observar el proceso de enumeración.
Snoopy Pro	Gratuito. Se tiene el control de cuándo iniciar el tráfico USB en un puerto para hacer funcionar un dispositivo. Funciona en Windows.	Requiere instalar libusb para Windows. Interfaz muy confusa. Pocas herramientas, especialmente las de filtrado.
Wireshark	Gratuito. Para analizar tráfico USB solamente funciona en GNU/Linux (modo superusuario). Interfaz amigable. Muchas herramientas. El puerto puede ser escaneado sin tener un dispositivo enchufado.	Escasa documentación para capturar tráfico USB. Requiere la instalación de librerías adicionales (libcap). Muestra solamente eventos registrados por el kernel, lo que puede llegar a confundir cuando no se registran respuestas de parte de un dispositivo.

De lo anterior, se elige a Wireshark como el analizador para el proyecto.

Por otra parte, es más recomendable escanear el tráfico generado por un controlador de código abierto, para evitar complicaciones con el fabricante original. Entonces se selecciona emplear GNU/Linux como SO con módulos de código abierto.

Empleando Wireshark en la captura

En una PC con SO GNU/Linux, distribución Ubuntu 10.10 *Maverick Meerkat,* se obtuvieron diversas capturas de dos adaptadores USB de red, una con modelo TPLINK 722n, y una ENUWI G2.

En las capturas, Wireshark automáticamente cataloga los REQUEST principales durante la enumeración USB.

3.2 Captura de las configuraciones USB iniciales de las tarjetas de red

El primer paso para el correcto funcionamiento de cualquier dispositivo USB es el proceso de enumeración. Se espera entonces observar este proceso en la fase inicial del tráfico USB. Esto se puede comprobar en la figura 3.1

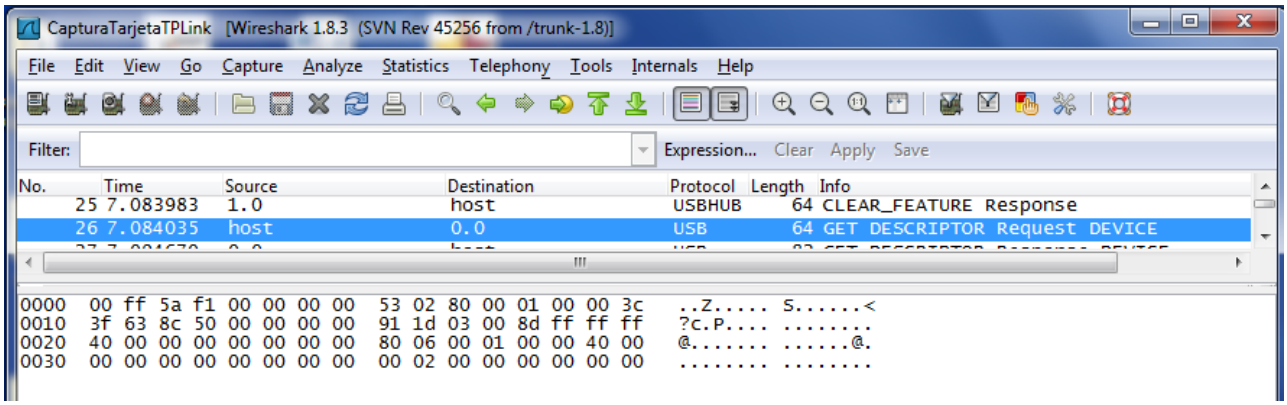


Figura 3.1 Captura del paquete USB GET DESCRIPTOR REQUEST.

La respuesta GET DESCRIPTOR RESPONSE es enviada desde el dispositivo al host. Como se mencionó en el capítulo 1, el grupo de trabajo "Valtroniks" cuenta con dos modelos de tarjetas de red inalámbrica USB, así que se analiza el tráfico USB generado por ambas.

3.2.1 Tarjeta TPLINK 722n

Como se muestra en la figura 3.2 el dispositivo en particular tiene definido al PID con el número 0x9271 y el VID con el 0x0CF3.

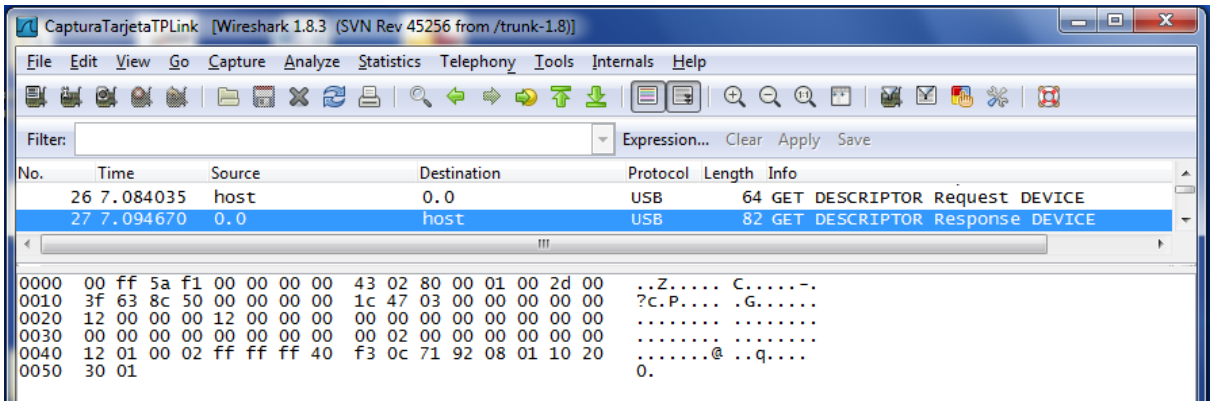


Figura 3.2 Captura del paquete GET DESCRIPTOR RESPONSE de tarjeta TPLINK

Una vez reconocido se le asigna un ADDRESS (dirección) al dispositivo para poder establecer la comunicación. Con esto se acaba el proceso de enumeración, y actúan ahora las instrucciones propias de su controlador. Se puede observar en la figura 3.3 que se envían en 12 ocasiones 4096 bytes al Endpoint 0, después 2128 y al final una transferencia de 0 bytes. Se observan 4160, debido a los 64 bytes cabecera de los paquetes USB.

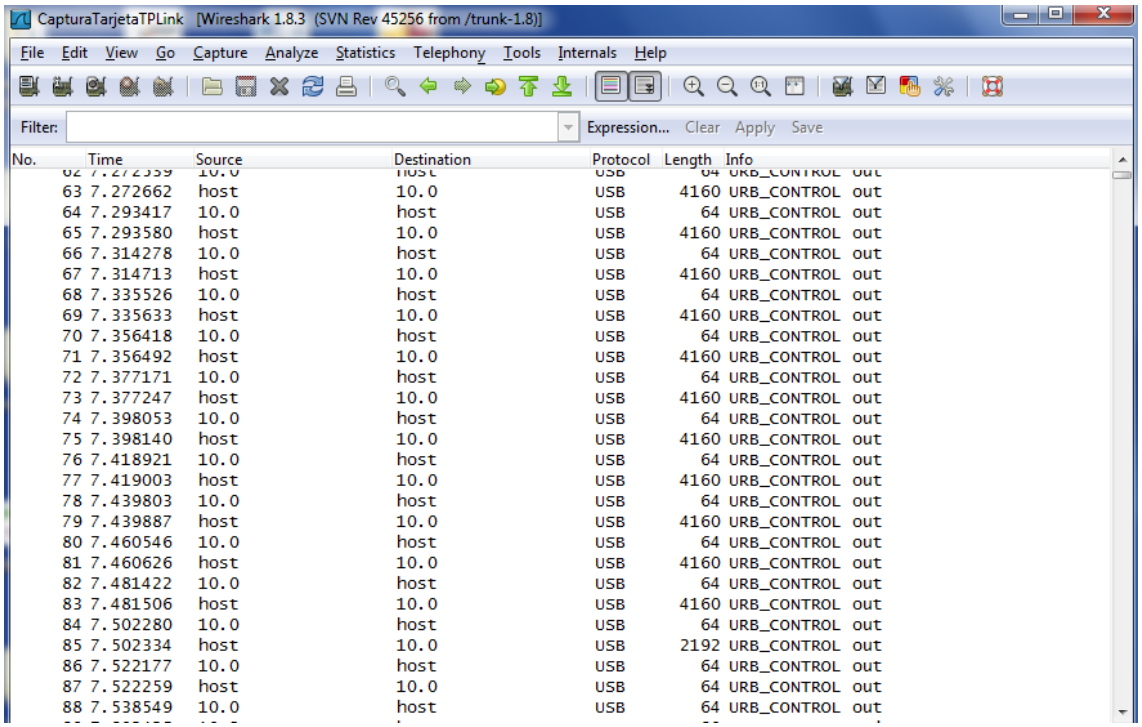


Figura 3.3 Captura de paquetes de longitud 4096.

En la figura 3.4 se observan transacciones Interrupt-IN Bulk-OUT, que siguen un patrón específico, explicado más adelante.

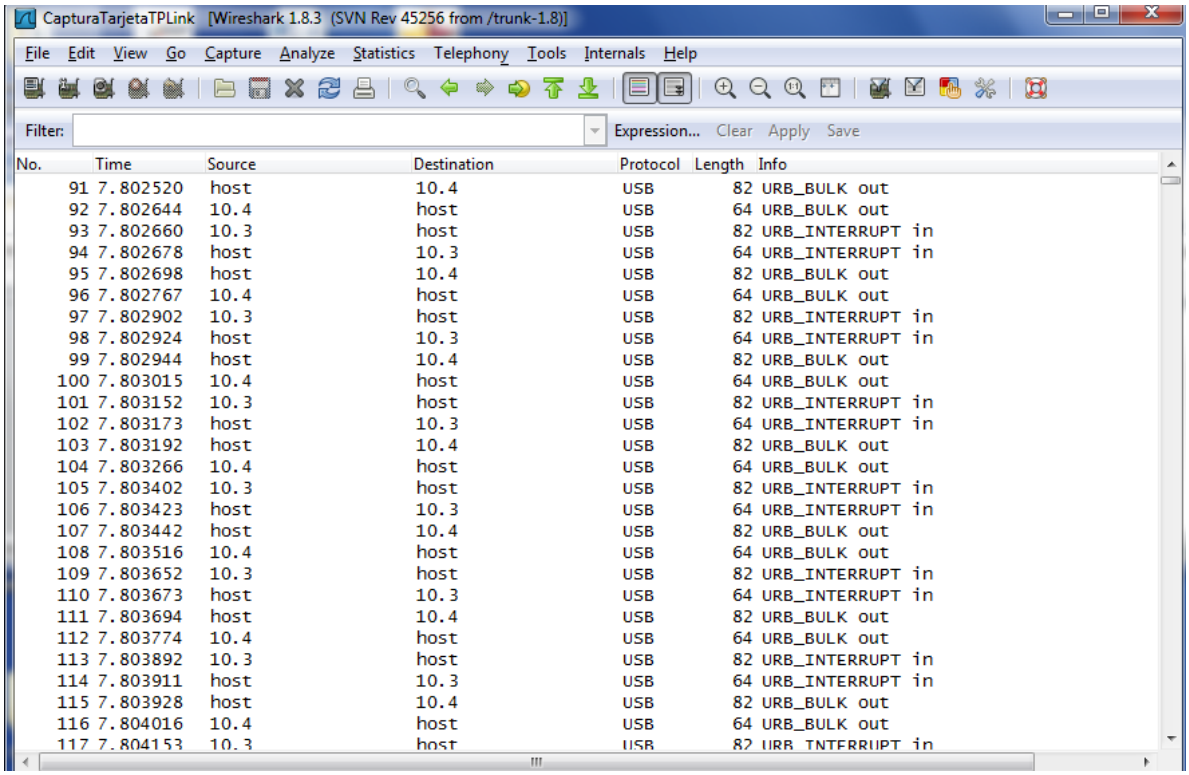


Figura 3.4 Transferencias Interrupt y Bulk.

3.2.2 Tarjeta ENUWI G2

La tarjeta ENUWI G2 con la que se cuenta con PID= 0x8189 y VID= 0x0BDA.

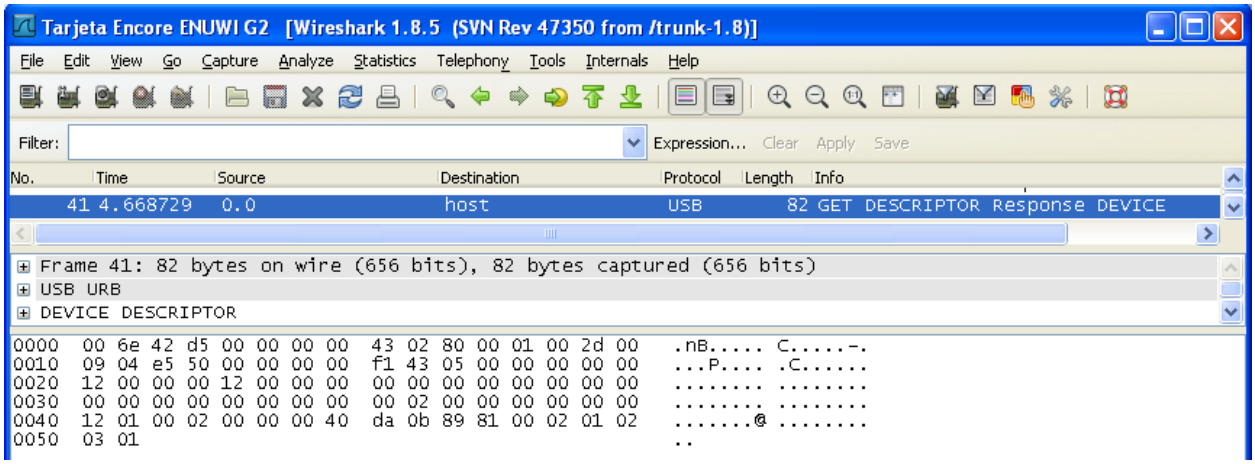


Figura 3.5 Captura del paquete GET DESCRIPTOR RESPONSE de la tarjeta ENUWI G2.

Se puede observar en la figura 3.6 que se envían múltiples paquetes tipo Control al Endpoint 0 de cero o 2 bytes.

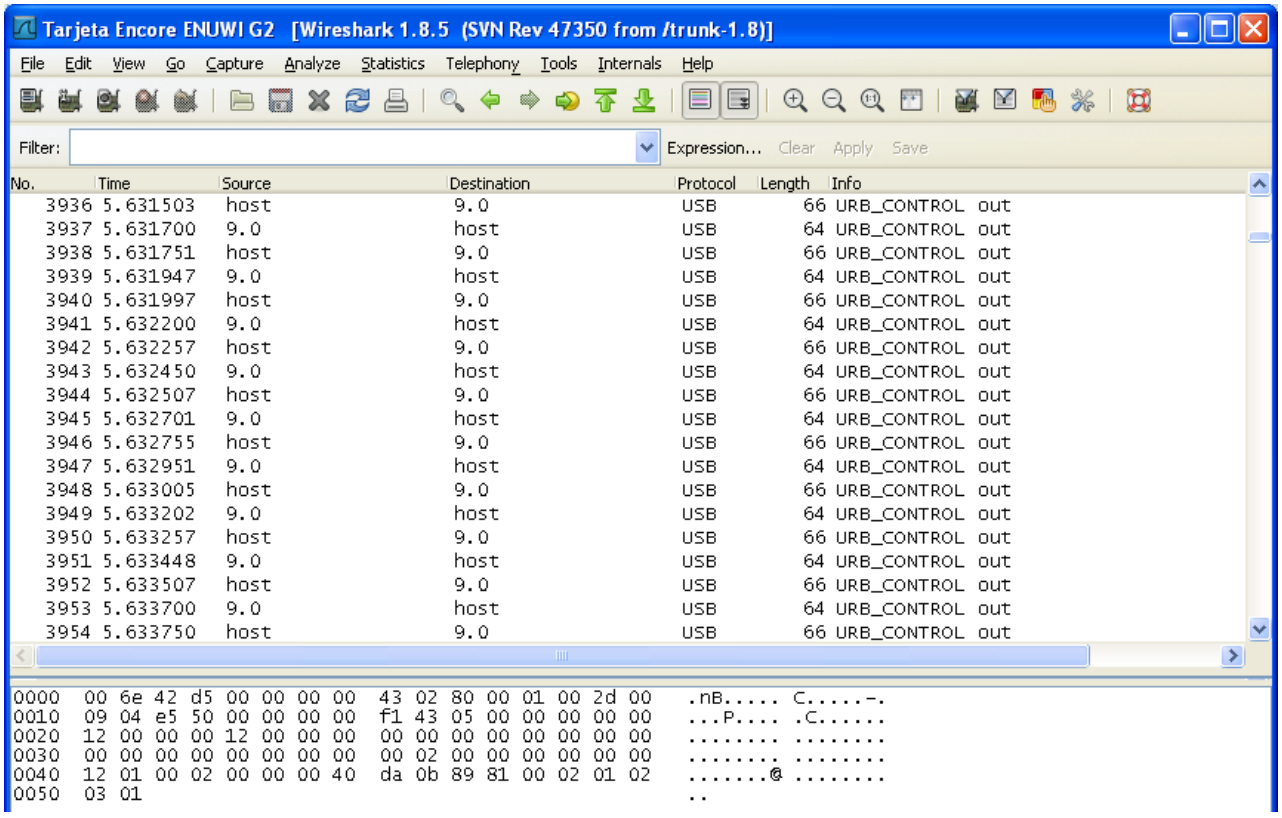


Figura 3.6 Captura de paquetes USB Control.

3.3 Proceso de asociación inalámbrica

Para que un host se asocie a un DS se siguen los siguientes pasos.

Paso 0. La figura 3.7 es la captura de una trama Beacon, donde es posible observar el valor del campo control de trama 0x80 00 00 00.

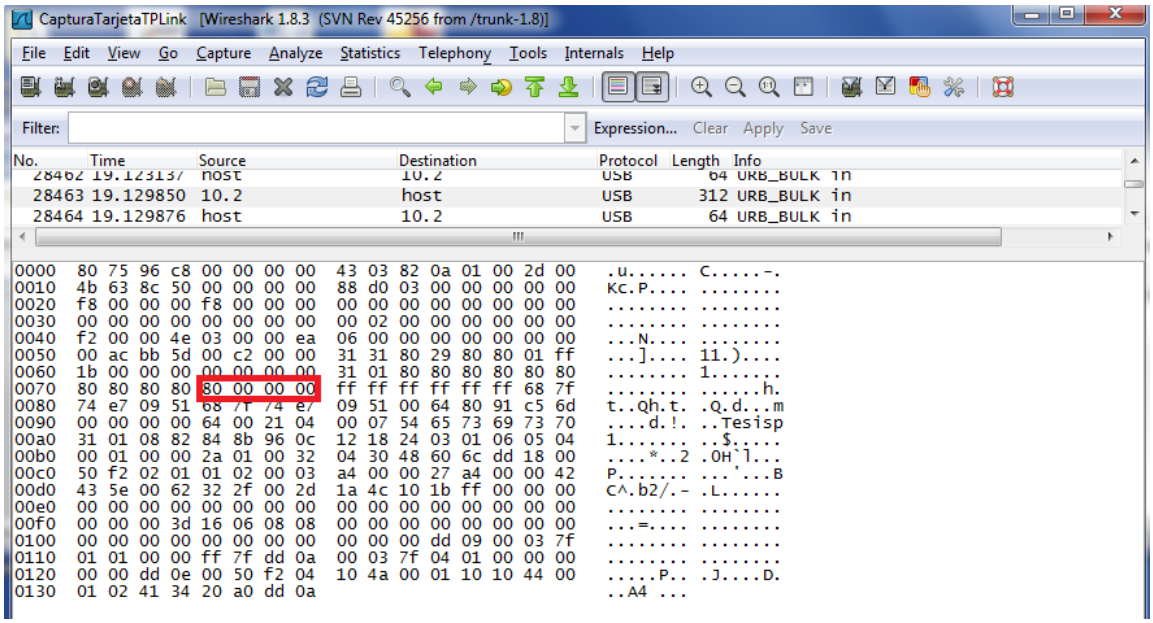


Figura 3.7 Captura de una trama Beacon

Paso 1. El host envía una trama denominada Probe Request. Como se muestra en la figura 3.8 el control de trama es: 0x40 00 00 00.

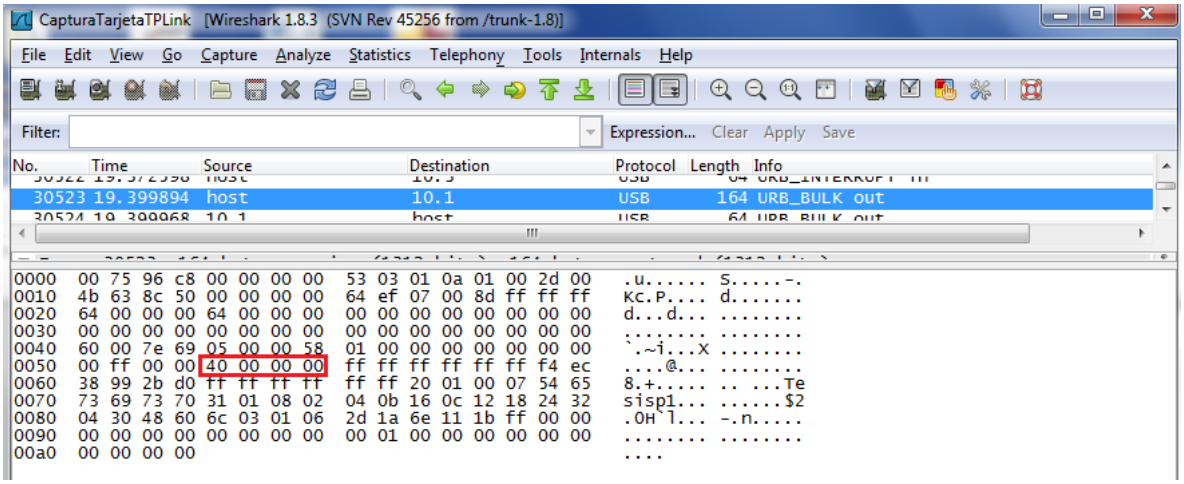


Figura 3.8 Captura de una trama Probe Request

Paso 2. El DS regresa una trama del tipo Probe Response. La figura 3.9 corresponde a la captura de un paquete Probe Response para la cual el control de trama es: 0x5000.

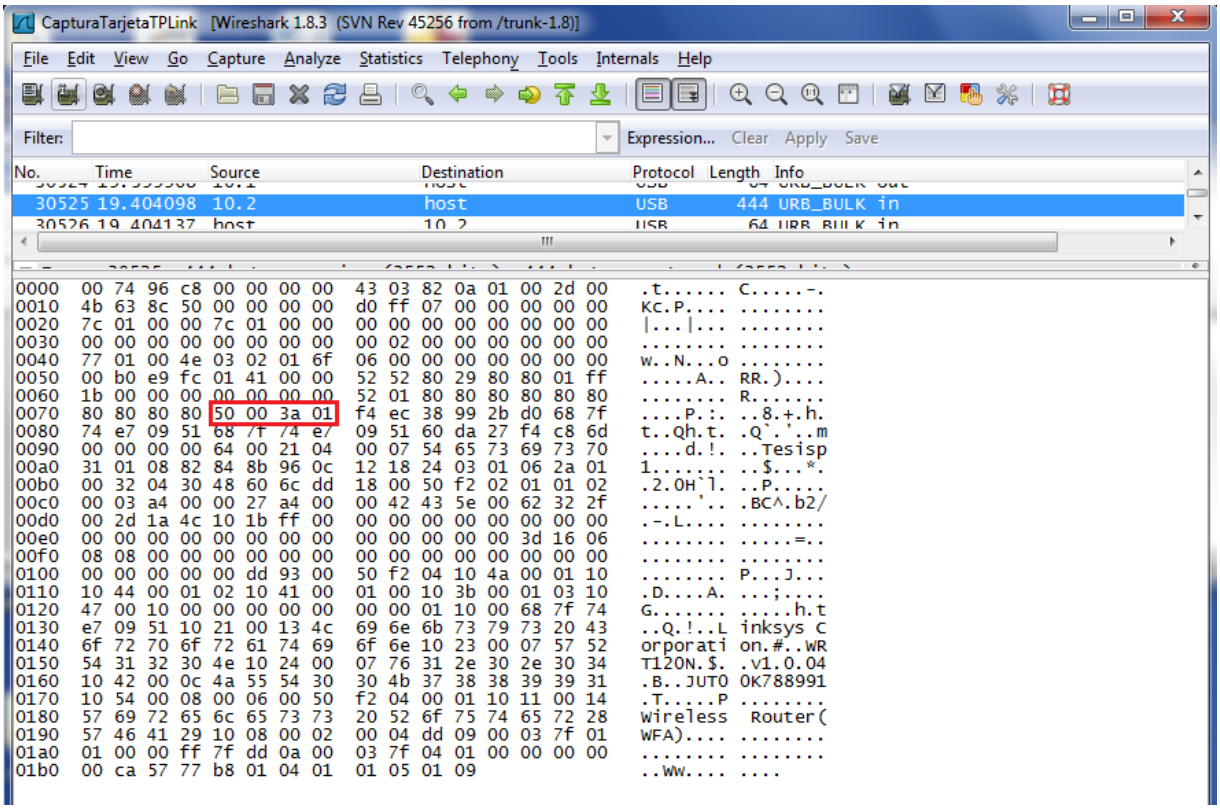


Figura 3.9 Captura de una trama Probe Response.

Paso 3 y 4. El host envía una trama denominada Autenticación (Authentication) para solicitar al DS que lo reconozca. La figura 3.10 corresponde a la captura de una trama de Autenticación, donde el control de trama es 0xB000. El DS regresa el mismo paquete al host.

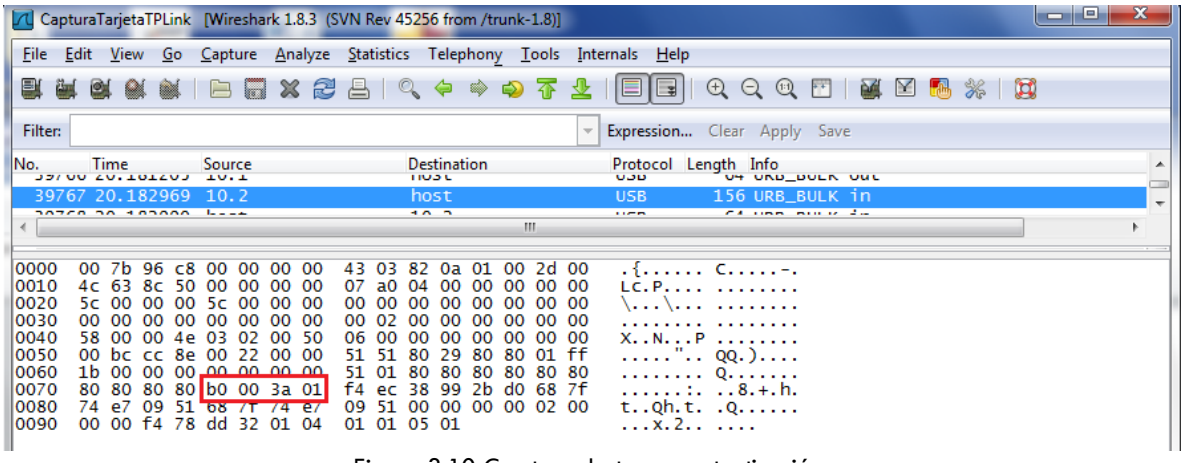


Figura 3.10 Captura de trama autenticación.

Paso 5. Después del descubrimiento y la autenticación, el host envía una trama denominada Association Request, cuyo control de trama es: 0x0000 (Figura 3.11).

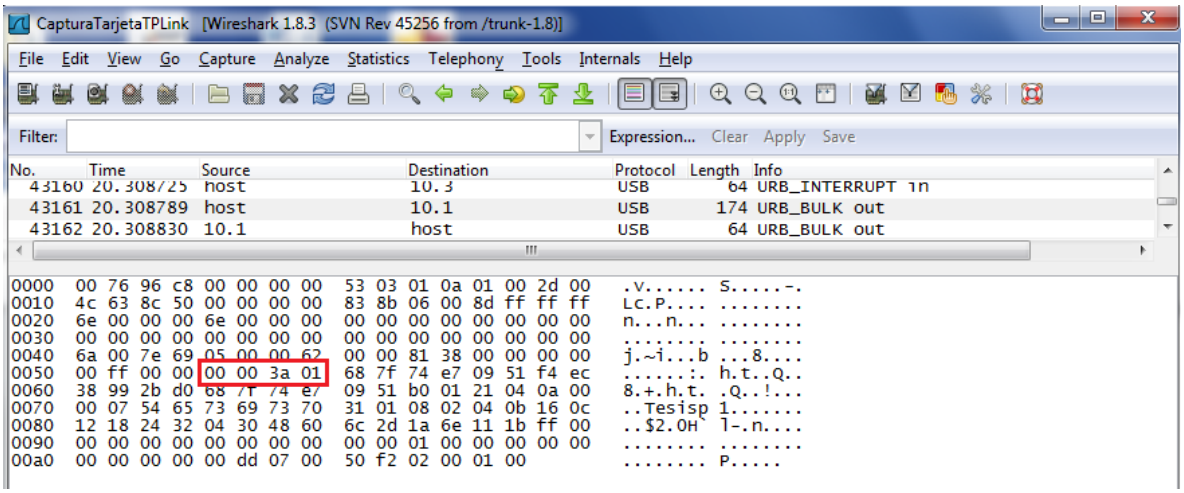


Figura 3.11 Captura de trama Association Request.

Paso 6. Finalmente el DS responde con una trama Asociación Response, cuyo control de trama es 0x1000 (Figura 3.12).

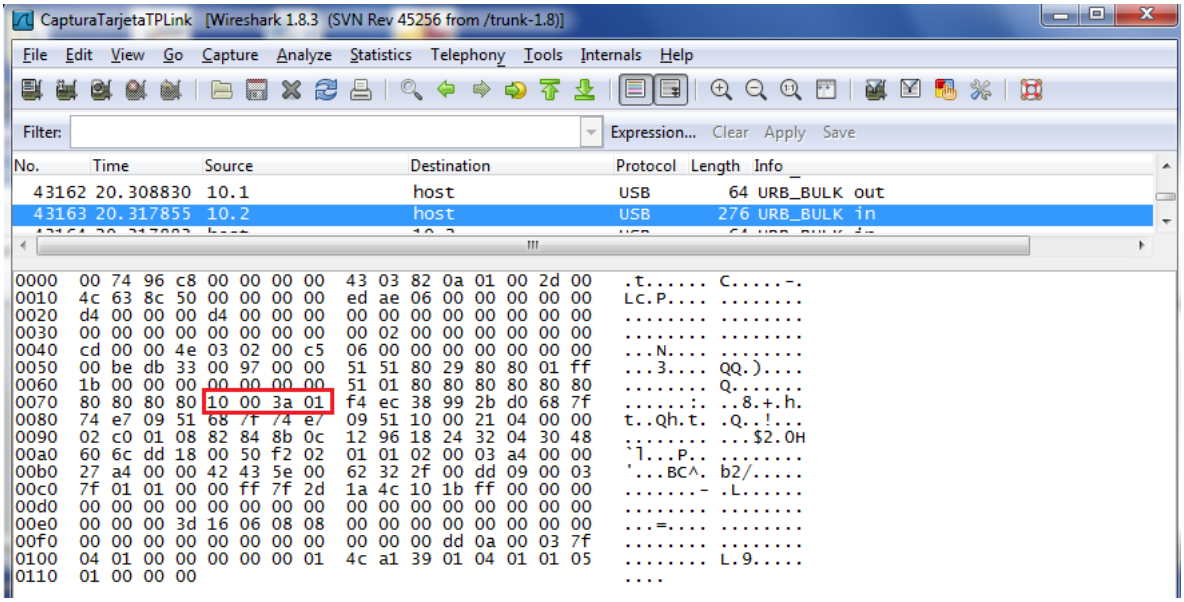


Figura 3.12 Captura de trama Association Response.

A partir de este momento, el host ya forma parte de la infraestructura de la WLAN.

Sólo es necesario asignarle una dirección IP dentro del segmento de red que en este caso es 192.168.1.0/24 siguiendo el diagrama de red de la figura 3.13.

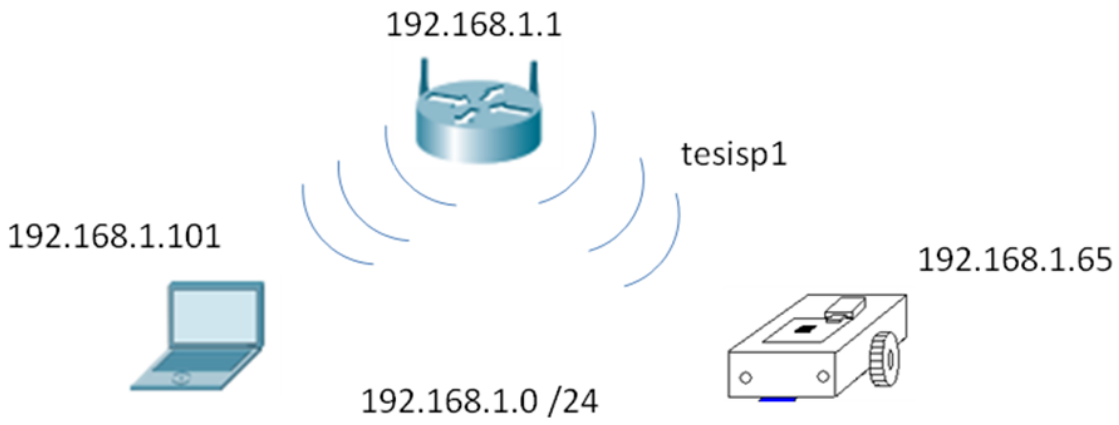


Figura 3.13 Diagrama de red propuesto.

Para hacer uso de la aplicación se emplea el comando PING. La figura 3.14 muestra la captura de una trama de datos (control de trama=0x8801) correspondiente a un paquete PING. El contenido del paquete depende del SO: en algunas ocasiones es una secuencia de bytes pseudoaleatoria y en otros, como en la figura, es el abecedario. También se señala el tamaño del encapsulado IP como 0x1F, y como ya se mencionó en el capítulo 2, se utiliza para conocer el tamaño del búfer del PING.

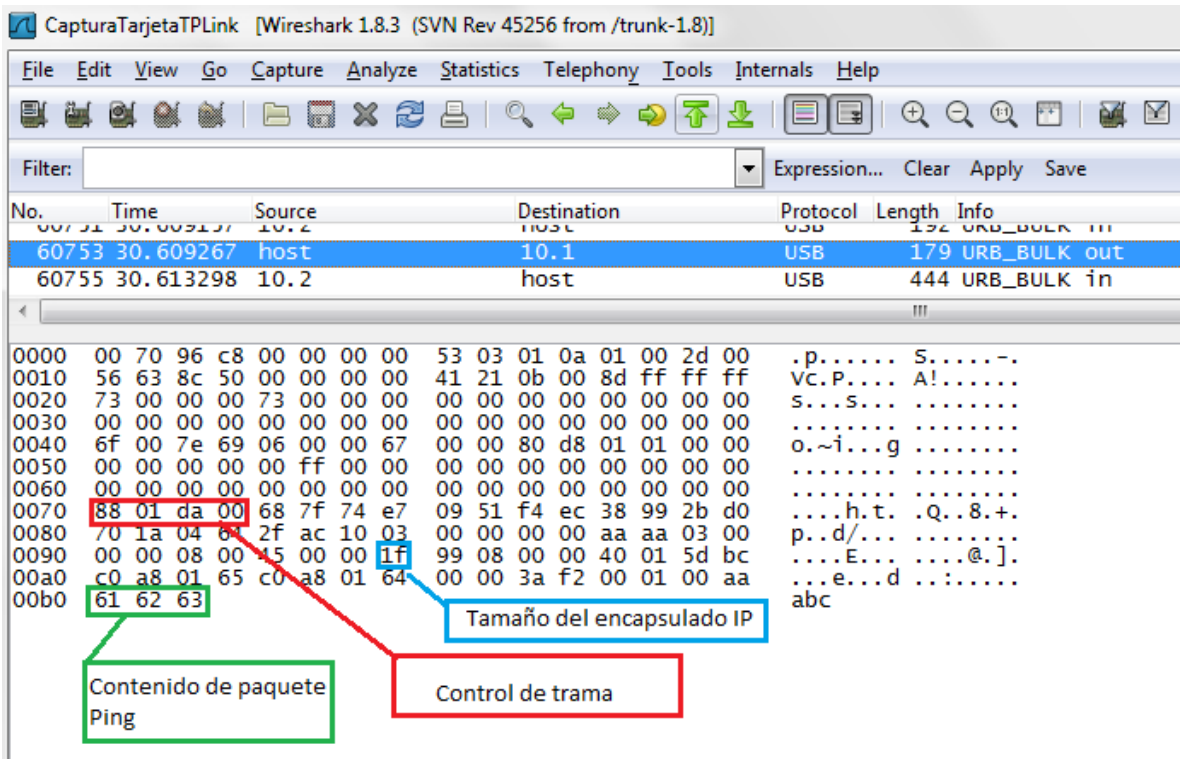


Figura 3.14 Captura de comando PING.

3.4 Programación de las capturas en el PIC32 USB Starter Kit II

Se ha mencionado que Microchip cuenta con el USB Stack para el desarrollo de aplicaciones que operan bajo el protocolo USB. Siguiendo con la información obtenida de las capturas del tráfico USB de las dos tarjetas que se poseen, se puede replicar la comunicación entre ellas y la PC.

El principal problema radica en que se requiere un controlador (segunda capa) adecuado para el dispositivo. Dentro de los controladores hasta ahora creados por la empresa, no existe alguno que sea específico para tarjetas de red. Ante tal situación, se emplea el controlador genérico, expuesto a grandes rasgos en el capítulo anterior.

De este modo se replicará la información obtenida de las capturas del analizador Wireshark.

El primer paso para emplear al PIC32 USB Starter Kit II es configurar la TPL (identificar la clase o el par VID-PID de los dispositivos que se manejarán y cuál controlador se les asignará), la corriente del bus inicial, las transferencias que usa el dispositivo, los números de NAK permitidos para cada tipo de transferencia y una configuración de los registros de la información por mandar en USB, denominado *modo ping-pong*. La manera más rápida de configurar todas estas cosas es utilizar una aplicación que se aloja dentro de uno de los directorios de instalación (USB Tools) del MPLAB IDE PIC32 Starter Kits, denominada USBConfig.exe (figura 3.15).

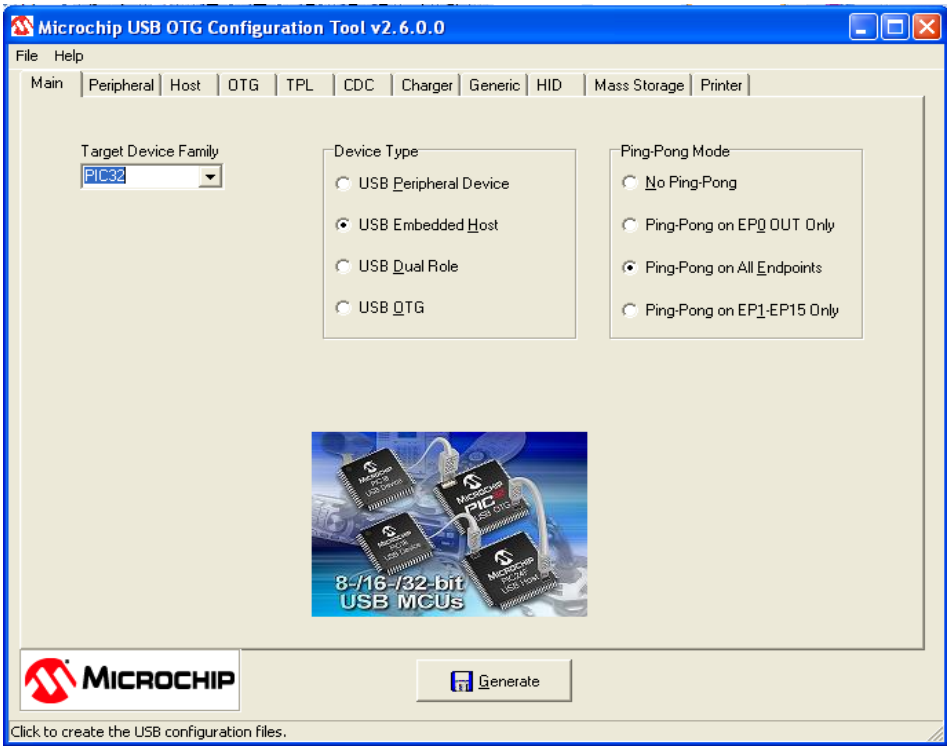


Figura 3.15 Pantalla inicial de USBConfig.exe de MPLAB.

Esta aplicación crea dos archivos: *usb_config*.h y *usb_config*.c. Dentro de estos archivos, además de configurar lo mencionado anteriormente, incluye la posibilidad de redefinir el nombre del manejador de eventos de la aplicación del MCU, de seleccionar el modo de operación de los controladores asignados (por eventos o por monitorización), entre otras cosas. También redefine ciertas funciones de importancia, por ejemplo, la función `USBHostTasks` es redefinida por `USBTasks` y `USBHostInitialize` por `USBInitialize`, en caso de que el usuario pretenda manejar al PIC en modo host.

Estos dos archivos se necesitan incluir con todos los necesarios en un nuevo proyecto, creado en MPLAB IDE PIC32 Starter Kits.

El segundo paso es la identificación de los bits de configuración del MCU. El código mostrado en el apéndice 1.1 muestra cómo se programaron para el presente proyecto, que es uno de los formatos más empleado en los proyectos-ejemplo de *MAL* USB.

Con el tercer paso se inicia propiamente la elaboración de la aplicación deseada por el usuario. Para el caso del proyecto se definen algunas variables globales, especialmente aquellas que se también emplee el manejador de eventos de la aplicación.

El proceso de enumeración es realizado automáticamente por el USB Stack en cuanto sea conectado físicamente un dispositivo. En caso de que su VID y PID sean los mismos entre los especificados en el TPL, se busca a la función de inicialización del controlador que le corresponda, en este caso el controlador genérico.

Para ambas tarjetas se necesita un proceso general similar. El MCU requiere obtener la información de cuando sea conectada la tarjeta. Sucedido esto, se debe asociarla a una WLAN presente, siguiendo la secuencia de envío de los paquetes del estándar 802.11 que se mencionaron en el capítulo 2 (Probe Request, Authentication, etc). Si la tarjeta devuelve información que sea igual a las respuestas que manda normalmente a una PC, se puede concluir que se ha asociado a la WLAN. En este punto el PIC está listo para recibir información desde una PC, por lo que se limitaría a esperarla para ejecutar cualesquiera tareas, sin embargo, debe continuamente mandar información al DS para hacerle comprender que sigue empleando la infraestructura (que para el proyecto se denominarán paquetes *keep-alive*).

Todo el proceso se muestra en el diagrama de la figura 3.16.

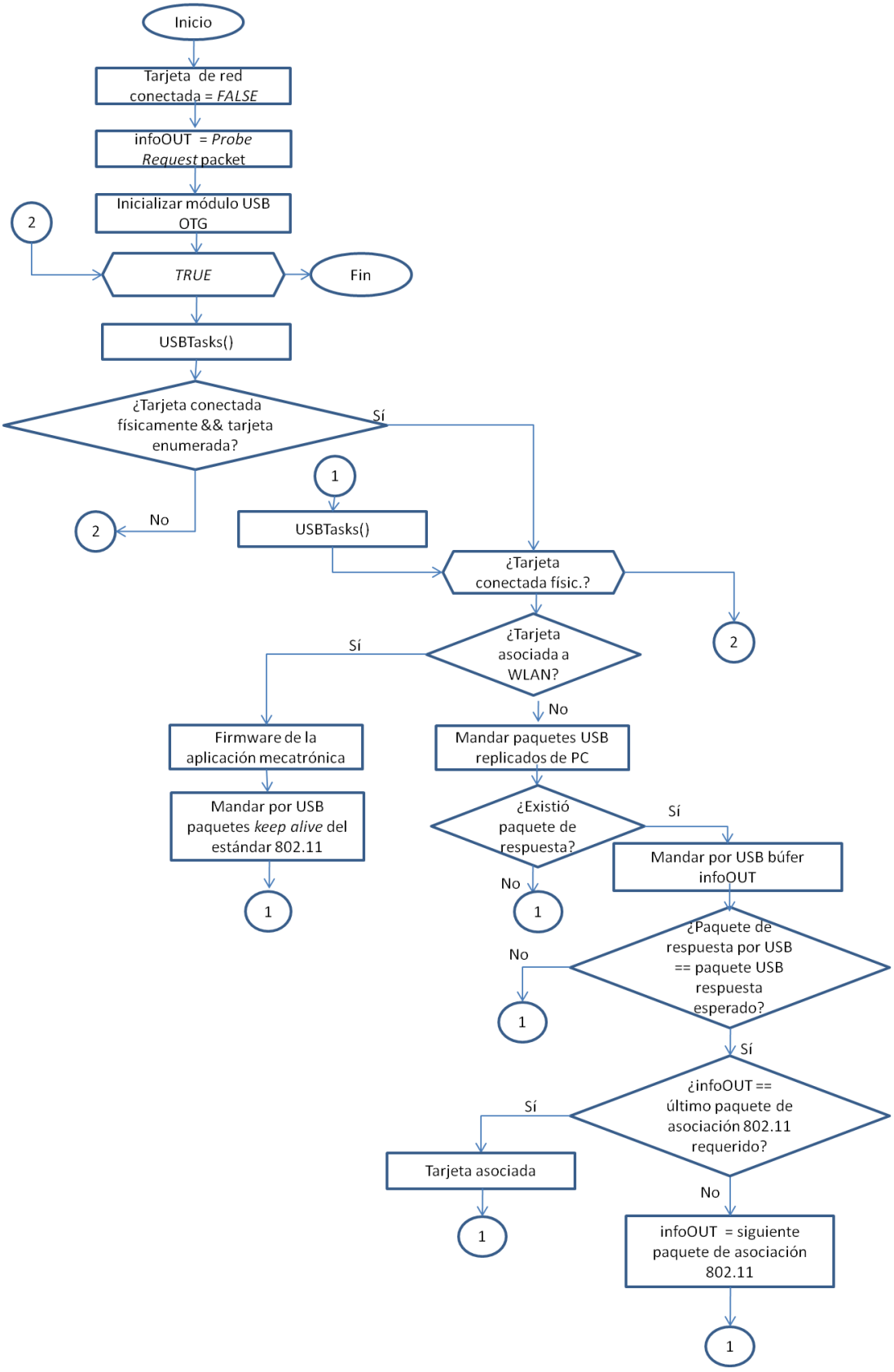


Figura 3.16 Diagrama de flujo general para el empleo de ambas tarjetas de red.

3.4.1 Tarjeta TPLINK 722N

El primer inconveniente encontrado es que la tarjeta TPLINK no maneja solamente los endpoints 0x01 ni 0x81, de acuerdo al controlador genérico de Microchip.

De acuerdo a los descriptores USB, la tarjeta TPLINK cuenta con las características lógicas mostradas en la tabla 3.2.

Tabla 3.2.Características lógicas USB de la tarjeta TPLINK 722n.

Característica	Valor
Clase USB	0xFF (Específica de vendedor)
Número de configuraciones	1
Número de interfaces en configuración 1	1
Endpoints	0x01 Bulk 0x82 Bulk 0x83 Interrupt 0x04 Interrupt 0x05 Bulk 0x06 Bulk
Corriente solicitada al host	500 mA

En las capturas, se observa que todos los endpoints son empleados, a excepción del 0x05 y 0x06.

- 3.4.1.1 Modificando al controlador genérico

Las funciones `USBHostGenericWrite` y `USBHostGenericRead` invocan respectivamente a la función `USBHostWrite` y `USBHostRead` con un parámetro adicional: el número del endpoint con el que se desea trabajar.

El controlador genérico tiene definido este parámetro con los números 0x01 y 0x81. Si se desea controlar otro endpoint, sería menester modificar este número.

Se puede crear otra función semejante con otro número fijo de endpoint. Para el proyecto se renombraron algunas funciones y se crearon otras dos, réplicas de las originales, pero con el número de endpoint cambiado:

`USBHostGenericWrite1:` para Endpoint 0x01

`USBHostGenericWrite4:` para Endpoint 0x04

`USBHostGenericRead2:` para Endpoint 0x82

`USBHostGenericRead3:` para Endpoint 0x83

Con estas cuatro funciones, es posible manipular los cuatro endpoints que están activos según lo observado con Wireshark.

- **3.4.1.2 Replicando las transferencias USB**

Un segundo inconveniente es encontrado tras el proceso de enumeración observado con Wireshark: la tarjeta de red manda enormes cantidades de bytes dedicados al Endpoint 0. No existe ninguna función, en el controlador genérico, que permita ejecutar directamente transferencias hacia dicho endpoint. El USB Stack clasifica como inválido emplear como parámetro al Endpoint 0 en las funciones `USBHostWrite` y `USBHostRead`, invocadas a su vez por las funciones `USBHostGenericWrite` y `USBHostGenericRead`. La única función que permite esto es la que se encuentra en el archivo *`usb_host`*`.c`, llamada `USBHostIssueDeviceRequest`. Cabe destacar que esta función emplea los mismos parámetros (analizados previamente) de los paquetes SETUP (bmRequestType, bmRequest, wValue, etc).

Para mejorar el empleo de la memoria, se ha programado que la aplicación invoque a una función llamada `Mandar4096`. Esta función establece los parámetros de un paquete SETUP, y posteriormente, invoca a otra. Esta otra función llamada `TarjetaConfig` (en un nuevo archivo incluido al proyecto llamado *`configuracion`*`.h)` es realmente quien define cada uno de los bytes observados con Wireshark, analizando cuál es el caso que se presenta: si es la primera ronda de 4096 bytes, si es la segunda, etc.

Como se vio anteriormente, Wireshark mostró 12 veces un envío de 4096 bytes al Endpoint 0, uno más de 2128 y uno de 0. La manera en que se mandan estos bytes es mostrada en el apéndice 1.6.

Tras estas primeras, se observa que, cíclicamente, la PC inicia transferencias IN al Endpoint 0x83, luego, transferencias OUT con información al Endpoint 0x04. La tarjeta contesta con ACK desde el Endpoint 0x04, y finalmente responde con información desde el 0x83.

Como se había visto durante las capturas, las primeras 45 veces esta "rutina" entre la información enviada y recibida, aunque guarda cierta semejanza, es muy irregular. Tras estas 45 veces las transferencias guardan cierta relación. El algoritmo que sigue esta relación se muestra en el apéndice 1.4.

Una vez superado esto, se intenta iniciar una transferencia IN al Endpoint 0x82, que es el endpoint en el que se observa que se obtienen las tramas Beacon. Ya que el USB Host Stack automáticamente maneja los NAK y los errores en el bus, se puede esperar a que dicho endpoint responda con los datos de la WLAN de prueba.

De recibir Beacon, es posible entonces intentar mandar las tramas de asociación (con el formato observado en las capturas USB). Si la respuesta a dichas transferencias resultan en las esperadas (las tramas Response o respuestas del AP), se puede enviar por USB la siguiente trama de asociación, hasta finalizar con todas ellas.

Una manera de saber que se ha obtenido la respuesta a todas ellas es verificando las posiciones 52 y 53 del búfer donde se recibió la información. La tabla 3.3 muestra sus valores para determinar las respuestas de las tramas.

Tabla 3.3. Reconocimiento de la trama por medio de valores de bytes en el búfer de respuesta.

Respuesta del AP	Posición 52 y 53 del búfer (paquete USB recibido)
Probe Response	0x 5000
Authentication (respuesta)	0x B000
Association Response	0x 1000

3.4.2 Tarjeta Encore ENUWI G2

Las características lógicas de esta tarjeta se muestran en la tabla 3.4.

Tabla 3.4.Características lógicas USB de la tarjeta ENUWI G2.

Característica	Valor
Clase USB	0xFF (Específica de vendedor)
Número de configuraciones	1
Número de interfaces en configuración 1	1
Endpoints	0x83 Bulk 0x04 Bulk 0x05 Bulk 0x06 Bulk 0x07 Bulk 0x89 Bulk 0x0A Bulk 0x0B Bulk 0x0C Bulk
Corriente solicitada al host	500 mA

De las capturas, se observa que sólo funcionan el 0x83, el 0x05, 0x89 y 0x0C.

- 3.4.1.1 Modificando al controlador genérico

De la misma forma que con la tarjeta anterior, se crean las siguientes funciones para manejar a los endpoints requeridos:

`USBHostGenericWrite5:` Endpoint 0x05

`USBHostGenericWriteC:` Endpoint 0x0C

`USBHostGenericRead3:` Endpoint 0x83

`USBHostGenericRead9:` Endpoint 0x89

Un problema particular de esta tarjeta es que durante el resto de su funcionamiento, siempre emplea al Endpoint 0. La PC parece mandar aleatoriamente paquetes pequeños. Puesto que su uso es continuo, lo mejor es implementar una función adicional en el controlador, que se denomina `PostConfiguracion`.

Esta función graba el número de la transferencia en la que se encuentra, y con ello determina algunos parámetros para invocar a otra, dentro del archivo *`configuracion`*`.h`, denominada `TarjetaConfig`. A diferencia de la otra tarjeta, esta función replica las transferencias hechas al Endpoint 0, configurando además, los parámetros que se requieren en la inevitable función `USBHostIssueDeviceRequest`. Por lo que no se limita a realizar un número pequeño (como las 14 de la otra tarjeta) de transferencias de control.

- 3.4.1.2 Replicando las transferencias USB

Las transferencias, hasta aproximadamente la número 1056, fueron replicadas en la función `TarjetaConfig`.

Mientras se hacen las transferencias de control, se intenta hacer una transferencia IN al Endpoint 0x83, con el fin de obtenerse las tramas Beacon.

De recibirse, se sigue la secuencia, similar a la tarjeta anterior, de enviar las tramas Probe, Authentication y Association por USB. La única diferencia radica en la posición de los bytes esperados en el búfer de respuesta (ver tabla 3.5), puesto que la ENUWI G2 no agrega bytes en los paquetes Data que no sean los exactos del formato estándar de las tramas 802.11.

Tabla 3.5. Reconocimiento de la trama por medio de valores de bytes en el búfer de respuesta.

Respuesta del AP	Posición 00 y 01 del búfer (Data USB recibida)
Probe Response	0x 50 00
Authentication (respuesta)	0x B0 00
Association Response	0x 10 00

3.5 Pruebas

3.5.1 Configurando al router de prueba

El primer paso es la configuración del DS. Para el proyecto se emplea un router CISCO Linksys WRT300N con el SSID “tesisp1” como se muestra en la figura 3.17.

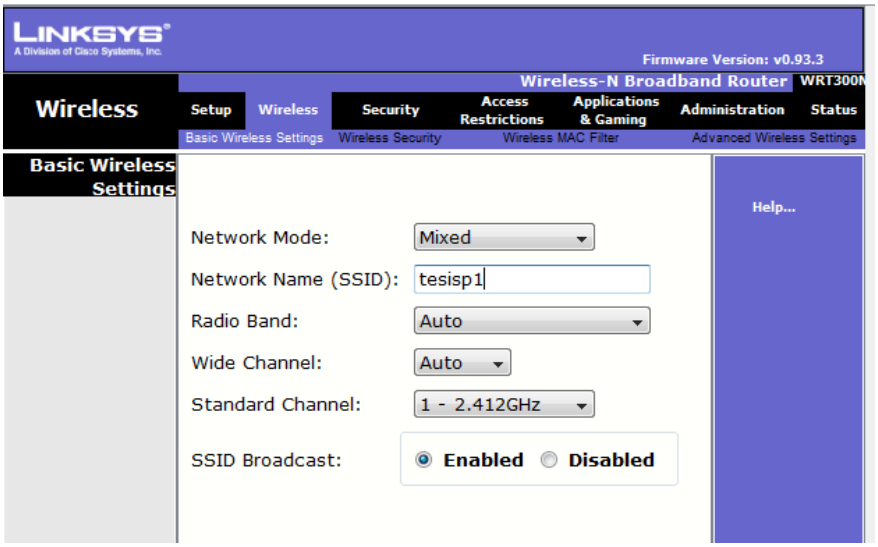


Figura 3.17 Parametros básicos de la configuración inalámbrica.

Después se deshabilita la seguridad como se muestra en la figura 3.18, ya que en la implementación no se contempla algún tipo de cifrado.

Figura 3.18 Seguridad inalámbrica.

Como no se utiliza el servicio DHCP, se deshabilita y se establecen la dirección IP y la máscara de subred del DS, como se muestra en la figura 3.19.

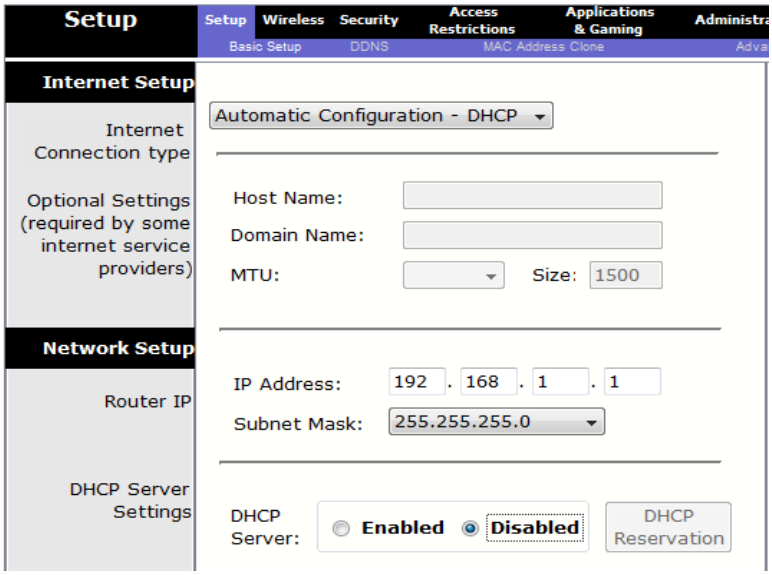


Figura 3.19 Configuración básica del router inalambrico.

3.5.2 Grabando el proyecto en el PIC32 USB Starter Kit II

Para programar al PIC32MX795F512L contenido en el PIC32 USB Starter Kit II por ICSP se selecciona como herramienta de debug. Esto se hace en el menú Debugger>Select Tool> 9 PIC32 Starter Kit(Figura 3.20). Previamente se conecta el USB Starter Kit II a la PC que cuenta con el MPLAB IDE PIC32 Starter Kits, por medio de un cable USB que incluya conectores macho de tipos A y mini B.

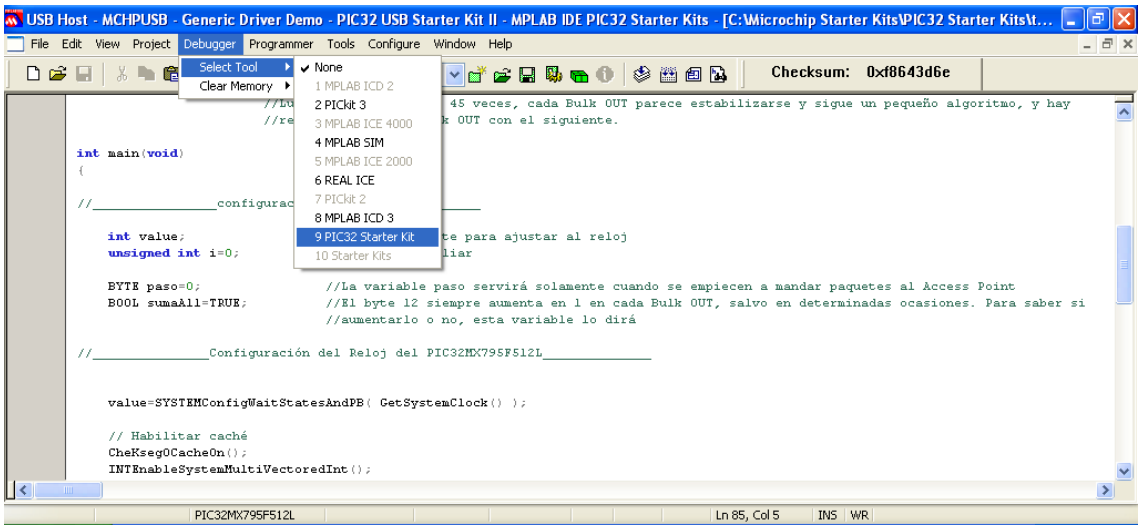


Figura 3.20 Selección del microcontrolador en MPLAB IDE PIC32 Starter Kits.

El siguiente paso es compilar el programa haciendo clic en el botón Make (Figura 3.21), que básicamente creará los códigos a bajo nivel para ingresarlos a la memoria del PIC. El diálogo de compilación correcta, la leyenda BUILD SUCCEEDED se muestra en la figura 3.22.

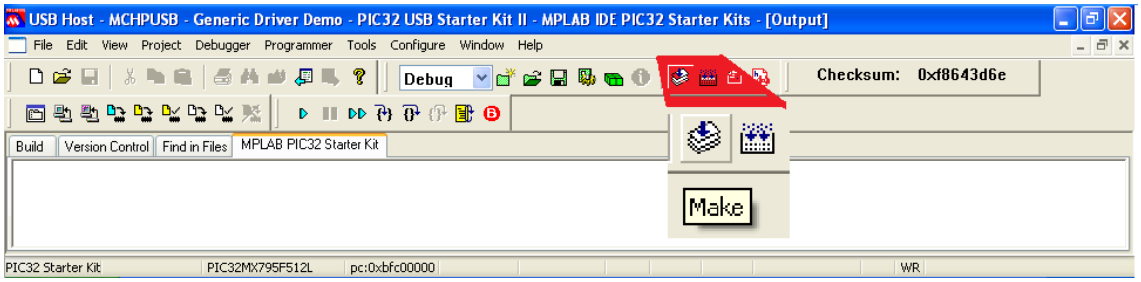


Figura 3.21. Botón Make para compilar el programa del PIC32.

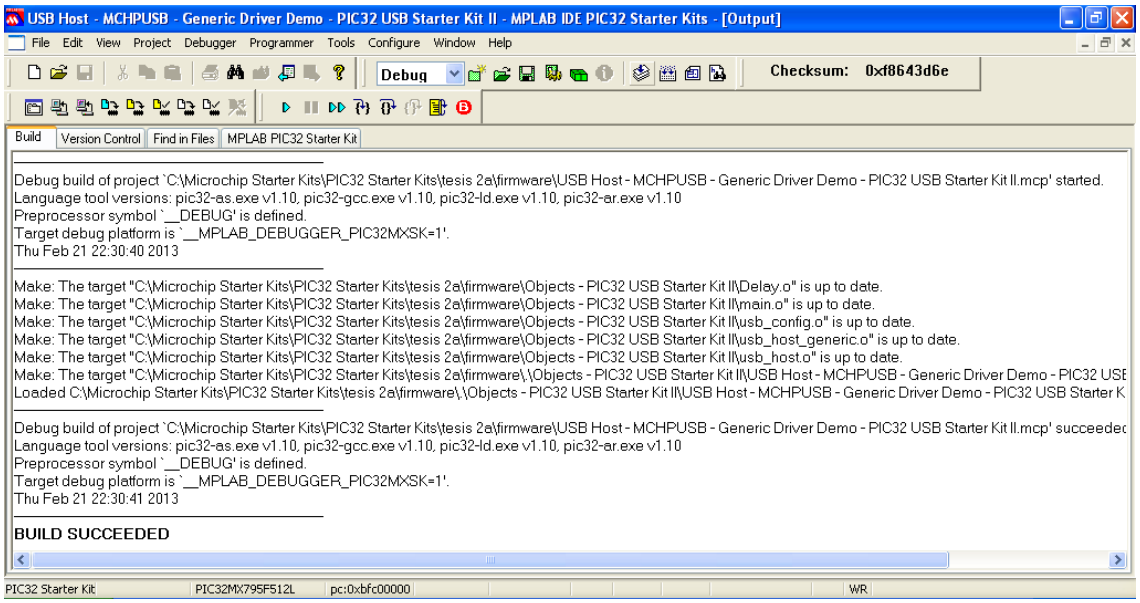


Figura 3.22 Ventana Output informando la compilacion exitosa.

El programa se graba en la memoria del PIC haciendo clic en el botón Program All Memories (Figura 3.23). La ventana Output indica el estado de la grabación (Figura 3.24).

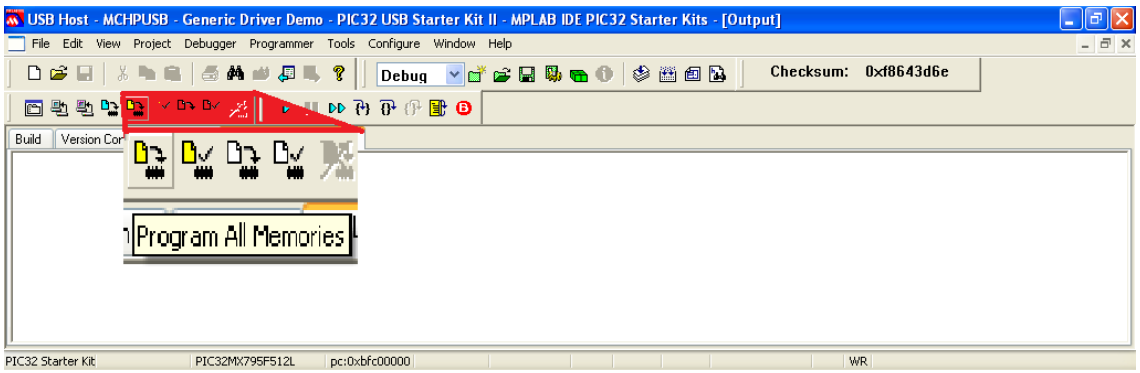


Figura 3.23 Botón Program All Memories.

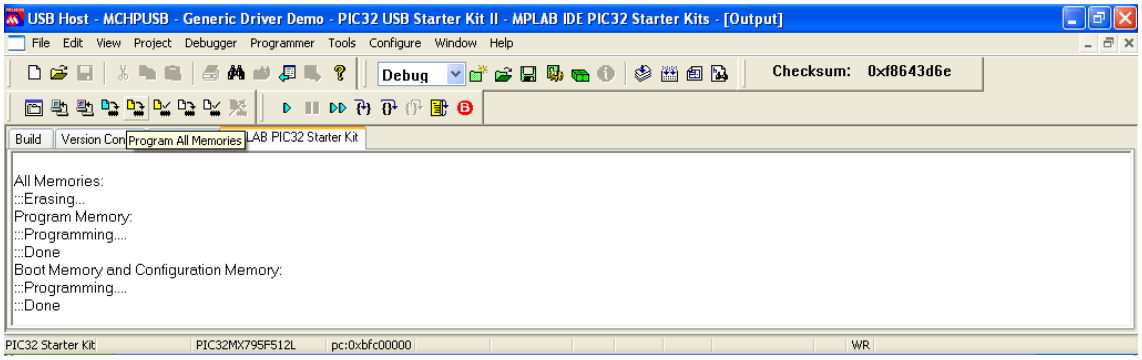


Figura 3.24 Resultado de la programacion del microcontrolador.

El PIC ya se encuentra programado para probar la aplicación. Puede desconectarse y trabajar autónomamente, pero puesto que requiere de alimentación, se mantiene conectado. Como MPLAB está controlando al USB Starter Kit II, se requiere cerrar.

Este kit, sin embargo, no puede energizar a la tarjeta de red, puesto que ocupa 100 de los 500 mA que el puerto USB le entrega. La tarjeta requiere de 500 mA. Una solución momentánea es emplear el hilo VBus (5 V) de otro puerto USB de la PC, y unir los GND de ambos. Los hilos D+ y D- son controlados por el Starter Kit II. Para poder conectar a la tarjeta se emplea un conector hembra tipo A.

La fotografía de las pruebas de esta conexión son mostradas en la figura 3.25.

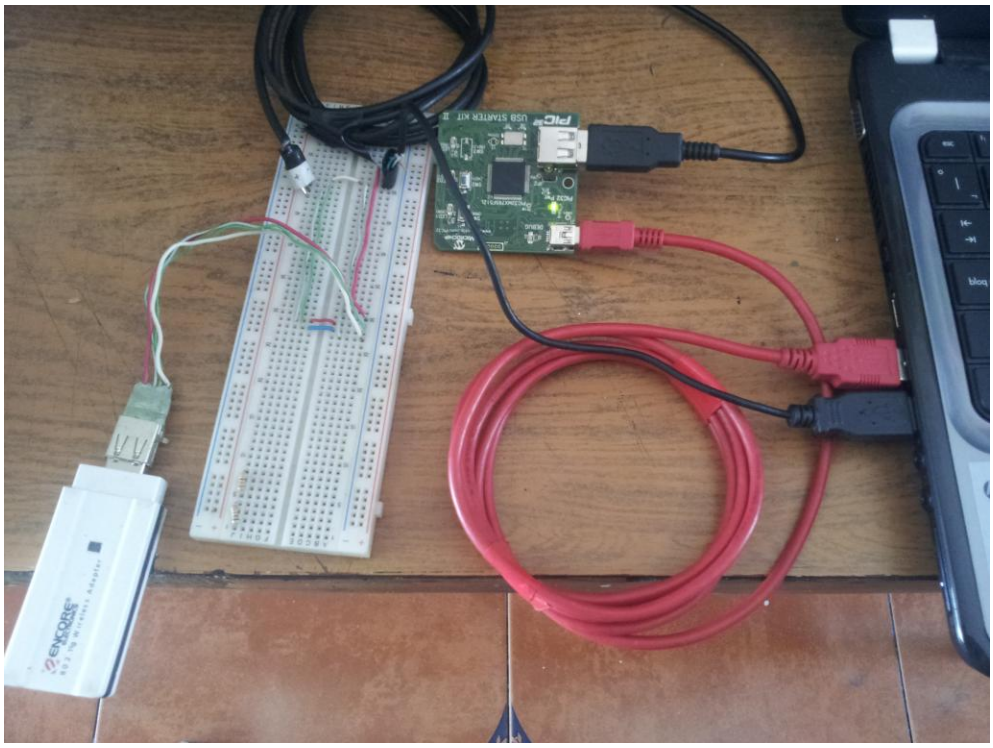


Figura 3.25. Conexiones realizadas para energizar correctamente a las tarjetas de red.

En este momento, el PIC está ejecutando el bucle infinito principal de la aplicación, ejecutando la función `USBTasks`, sin notificar los únicos eventos que la ISR registra: los SOF. La máquina de estados del bus no avanza: continúa en el estado DETACHED. El módulo USB OTG está a la espera de que un dispositivo USB sea conectado, para poder verificar su descriptor y si tiene un controlador para él. En caso de que así sea, el evento USB_ATTACHED llegará a la capa de la aplicación y seguirá con el programa principal. En caso de que no, el módulo USB OTG espera que sea removido.

Es el momento para conectar la tarjeta de red al conector hembra. De funcionar el programa, se espera que surja el tráfico con el DS.

3.5.3 Buscando las tramas 802.11

Se procede a hacer la prueba de asociación con el DS con la tarjeta TPLINK 722n. Se captura el tráfico de la red inalámbrica y se filtra buscando la trama Probe Request que es el primer paquete que debe enviar la tarjeta (Figura 3.26), el cual no es encontrado.

Se reprograma al PIC con el proyecto creado para ENUWI G2 por lo que se procede a hacer la misma prueba con esta (Figura 3.27).

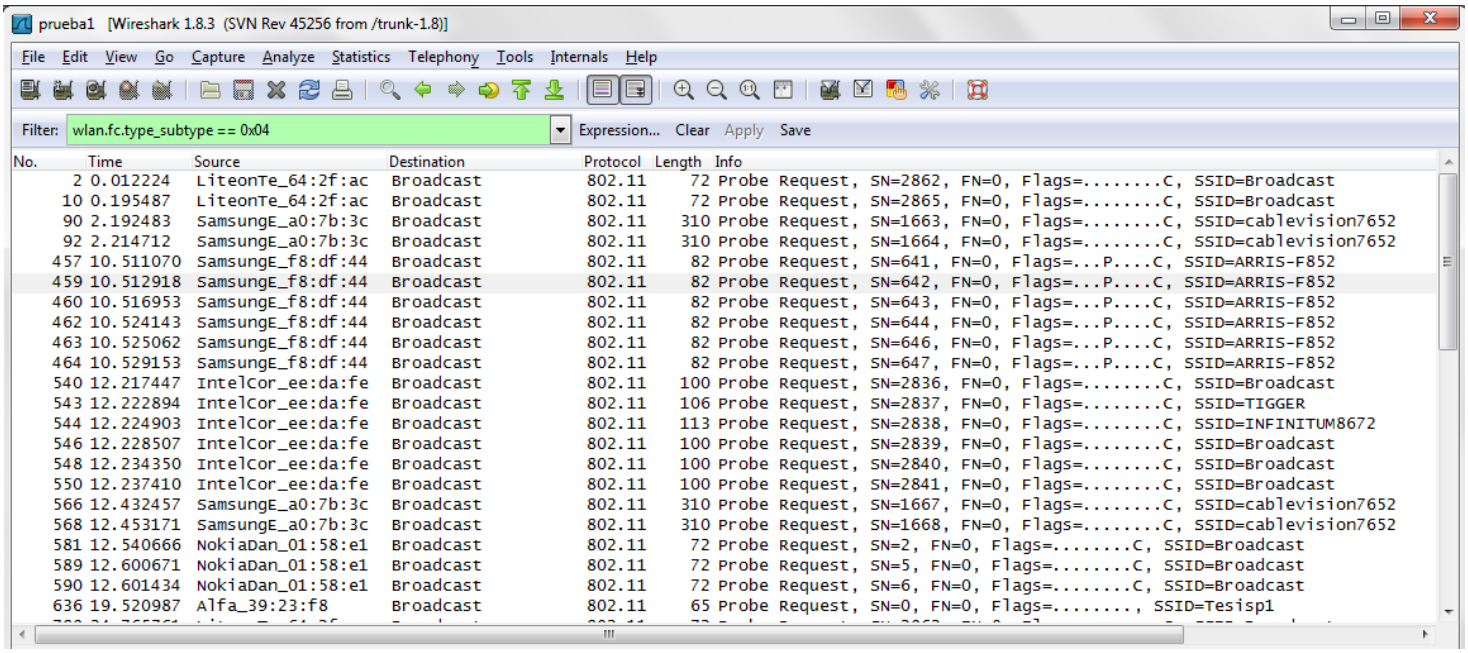


Figura 3.26 Captura del tráfico buscando tramas de la tarjeta TPLINK 722n.

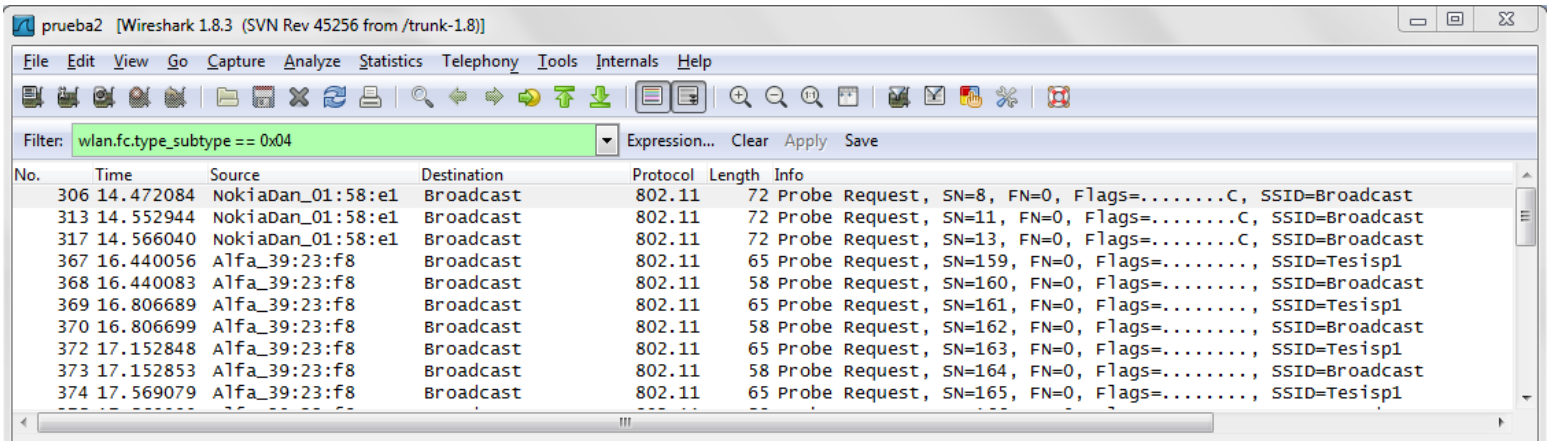


Figura 3.27 Captura del tráfico buscando tramas de la tarjeta tarjeta ENUWI G2.

Tampoco se logra observar que los paquetes sean enviados al DS.

Se necesita entonces verificar el correcto funcionamiento del programa grabado en el PIC y poder observar las transacciones USB.

3.5.4 Comprobando las transferencias USB y sus respuestas

Se vuelve a abrir la aplicación MPLAB PIC32 Starter Kits, con la finalidad de seguir la continuidad de la ejecución del programa.

Se configura nuevamente al USB Starter Kit II como debugger, se compila y se graba nuevamente el proyecto en él.

En lugar de cerrar el software, se emplearán las herramientas Debug. Puede hacerse clic en el botón Run (Figura 3.28) para ejecutar el código con el PIC32. Por otro lado, se empleará una función llamada `DBPRINTF` (desarrollada por Microchip), que imprime en la ventana Output cualquier información que se especifique en su parámetro.

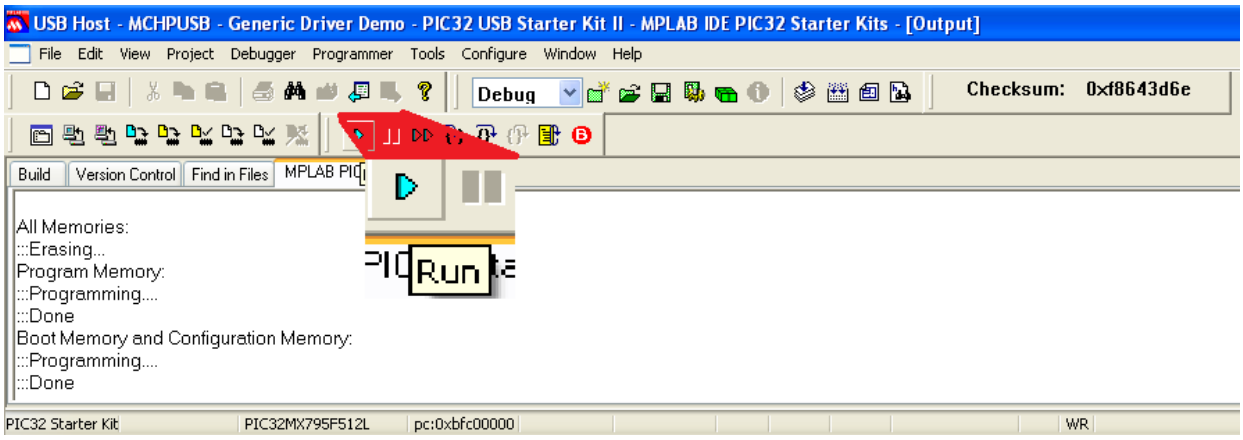


Figura 3.28 Botón Run.

- **3.5.4.1 TPLINK 722n**

Se comprueba el correcto reconocimiento de la tarjeta imprimiendo en la ventana Output: su VID y PID, la manera en que es reconocido en el TPL, su configuración, el ADDRESS que se le asigna y un mensaje que indique que efectivamente el controlador se ha inicializado.

La captura de pantalla de la figura 3.29 muestra precisamente la información deseada, por lo que el problema no existe en el proceso de enumeración ni en la inicialización del controlador.

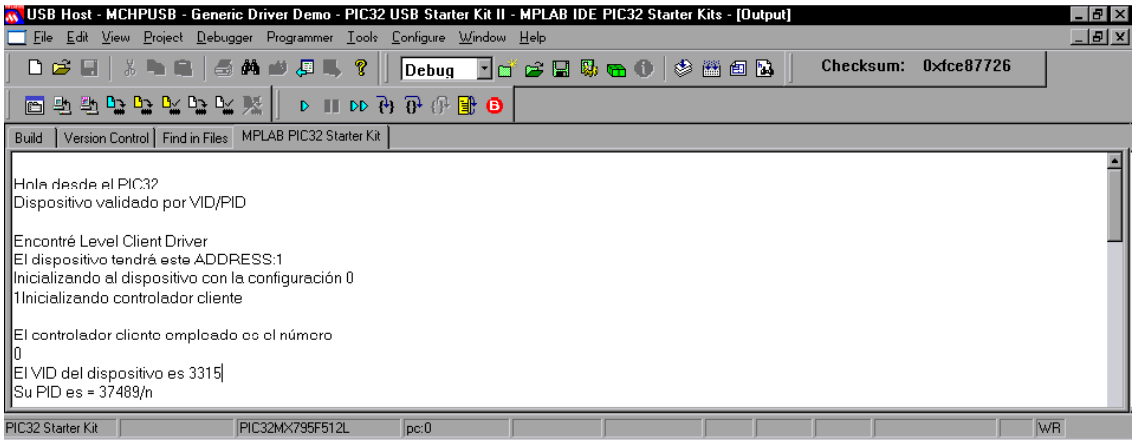


Figura 3.29 Reconocimiento del dispositivo.

La función `Mandar4096` consiste en mandar 14 búfer consecutivos de bytes. Lo que se plantea es que un mensaje aparezca cada vez sean mandados dichos búferes a la tarjeta (”caso no. x”). La modificación es mostrada en el apéndice 1.6. La captura de pantalla de la figura 3.30 muestra que efectivamente todos los bytes fueron enviados a la tarjeta.

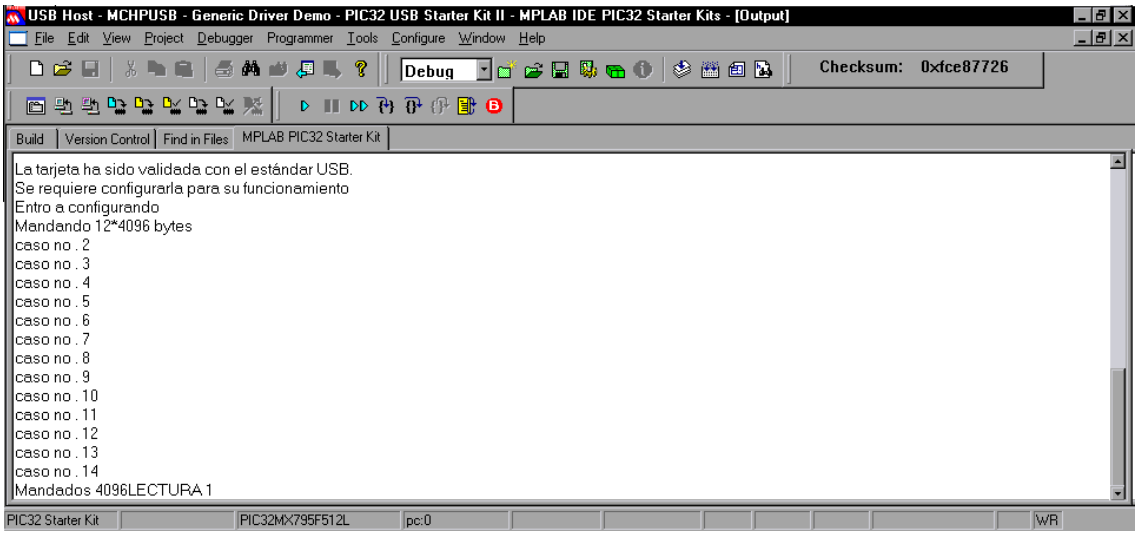


Figura 3.30 Envío de los paquetes de control iniciales.

Es probable que las transferencias a los otros endpoints hayan sido rechazadas. Se modifica a la función `_USB_FindEndpoint` para que muestre el nombre de los endpoints a medida que los encuentra de la información almacenada durante la enumeración.

La aplicación es modificada para notificar que las funciones `USBHostGenericRead3`, `USBHostGenericRead2`, `USBHostGenericWrite1` y `USBHostGenericWrite4` inicializaron las transferencias sin errores, o se toparon con alguna excepción que hizo que no pudieran cargar la información (por enviar o recibir) en la estructura del endpoint.

Se aprovecha la “rutina” cíclica de transferencias Interrupt IN-Bulk OUT, para determinar cuál de esta serie de transferencias se encuentra realizando el MCU. Se les denominará `Lectura` aunque en realidad sean tanto lectura y escritura.

Al manejador de eventos de la aplicación se le agrega la posibilidad de mostrar en la ventana Output los bytes recibidos desde el Endpoint 0x83 y el término exitoso de las transferencias OUT. En la figura 3.31 se aprecian los textos de la búsqueda de los endpoints para la transferencia, las leyendas “TXDONE” (OUT realizado) y “RXDONE” (IN realizado), los bytes recibidos de la tarjeta tras “RXDONE” y el número de secuencia (`Lectura`) que se encuentra realizando el MCU.

`Lectura` se estanca en el número 2, puesto que no vuelve a recibir los eventos de “TXDONE” ni de “RXDONE”. Por otra parte el vector de recepción de los paquetes Data USB muestra bytes con valores iguales a cero.

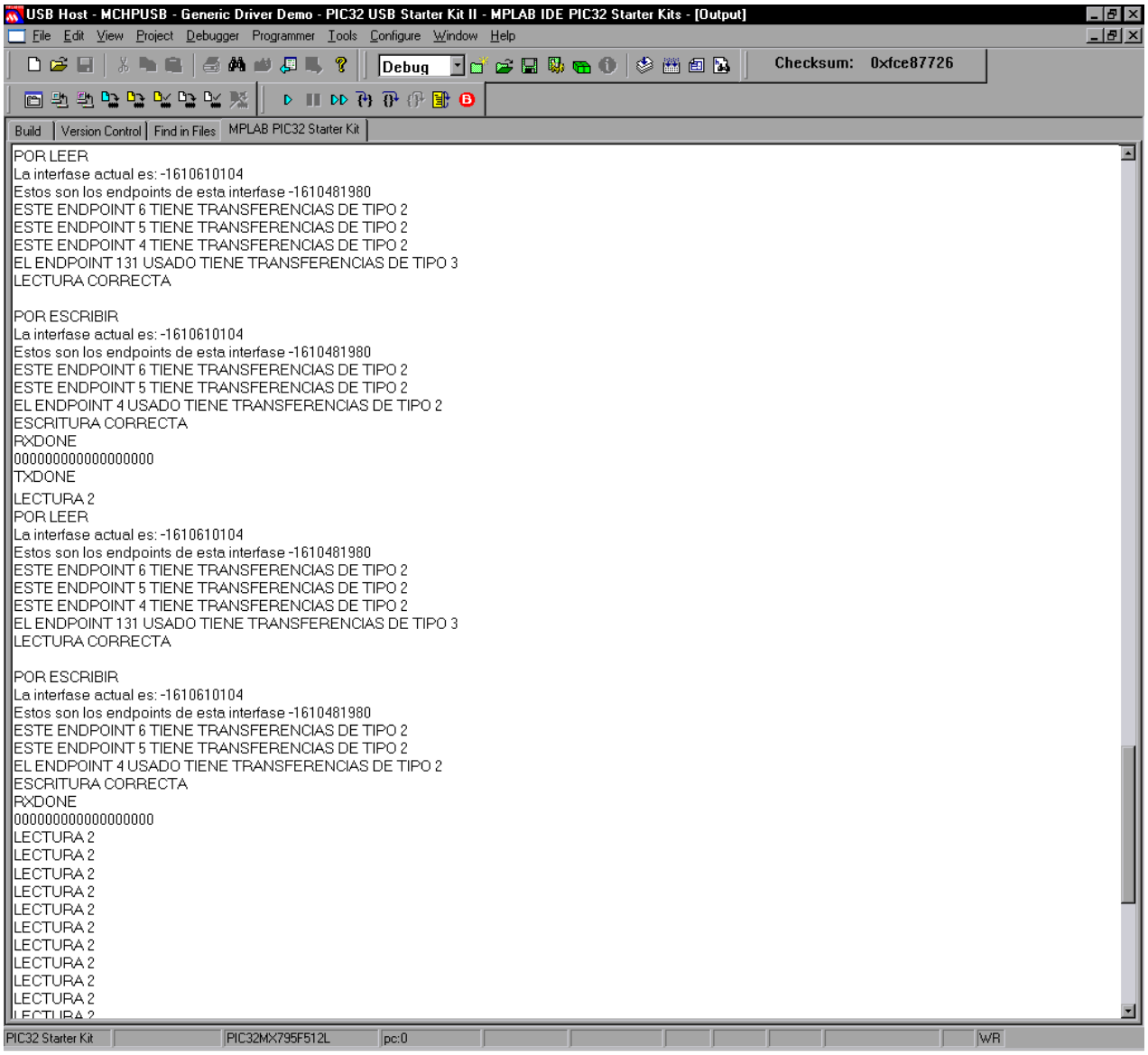


Figura 3.31 Comprobación del funcionamiento de las transferencias, de la búsqueda de los endpoints y observación de los bytes en paquetes Data recibidos de la tarjeta.

Se requiere entonces analizar con mayor profundidad el comportamiento de las transferencias. Algún proceso automatizado por Microchip dentro del USB Stack impide que se pueda apreciar la posible problemática.

Como se menciona en el capítulo 2, es el IRQ del módulo USB OTG quien notifica los principales eventos ocurridos en el bus. Una transferencia IN exitosa implica que se han recibido el número de bytes esperados, y una transferencia OUT exitosa cuando el dispositivo ha transferido un paquete ACK.

Con este conocimiento, se modifica entonces la ISR del USB Stack, dentro del archivo `usb_host.c`, llamada `_USB1Interrupt`, de tal manera que en lo sucesivo se pueda desplegar en la ventana Output cuándo se reciben paquetes NAK y ACK (Figura 3.32).

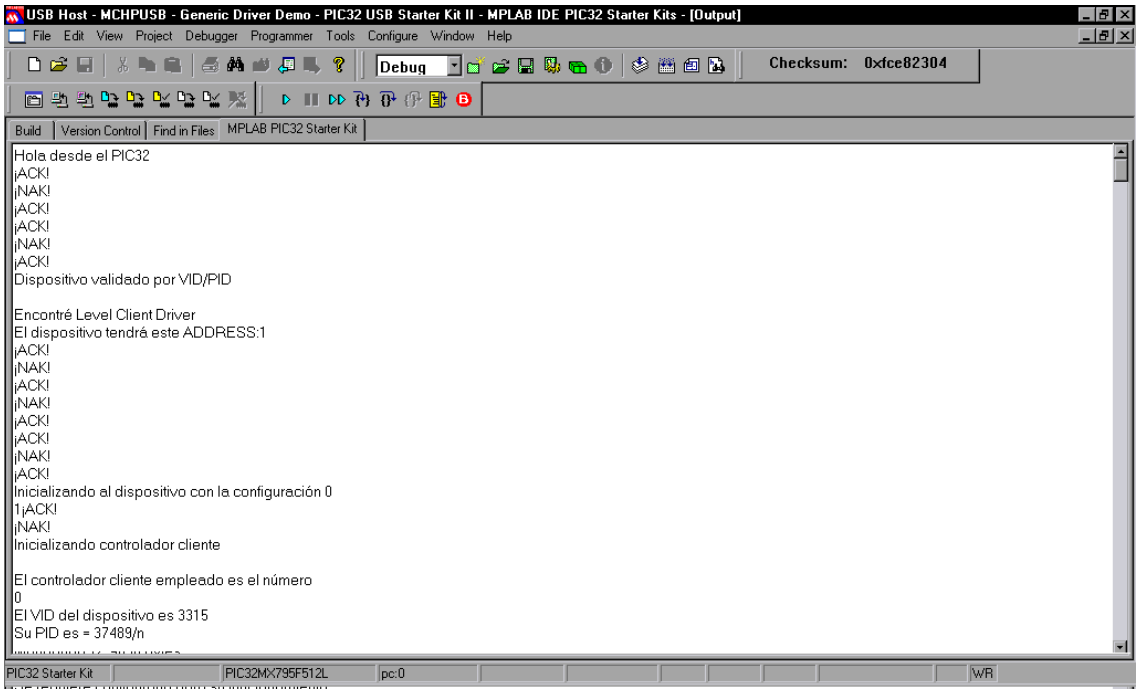


Figura 3.32 Muestra de los paquetes Handshake.

El máximo de bytes de los paquetes de control son 64. 4096 bytes de control en bloques de 64 bytes requerirían 64 transferencias. 65 ACK se aprecian entre las leyendas "caso no. x" en la ventana Output, cuando se invoca a la función `Mandar4096`.

64 corresponden entonces a cada transferencia de control con 64 bytes, y el primer ACK es la respuesta al paquete SETUP. Parte de estos 65 ACK son mostrados en la figura 3.33.

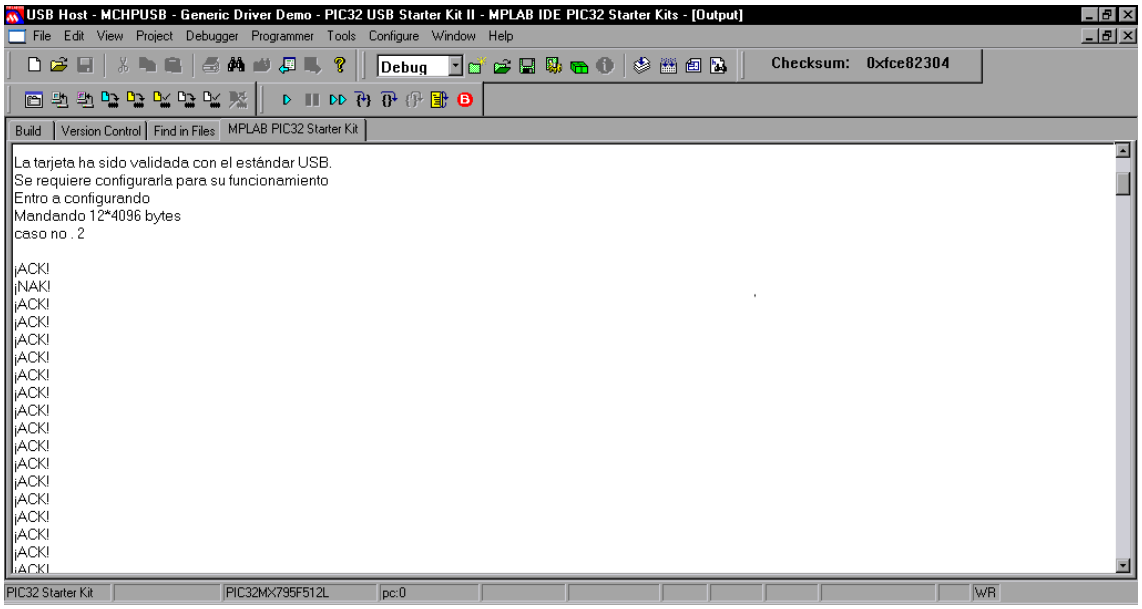


Figura 3.33 Vista de los paquetes Handshake tras mandar los primeros 4096 bytes.

Con esto se comprueba que la función DBPRINTF funciona para visualizar paquetes Handshake USB.

Tras finalizar la función `Mandar4096` y comenzar con la "rutina" cíclica de mandar y recibir transferencias hacia endpoints diferentes del 0, es cuando se observa la razón del porqué el MCU se estanca en `Lectura` igual a 2. En la figura 3.34 se observa como la "Lectura" 1 es realizada visualizándose los paquetes Handshake. Se muestra el búfer o vector donde se reciben los bytes de respuesta, y tienen todos valores 0.

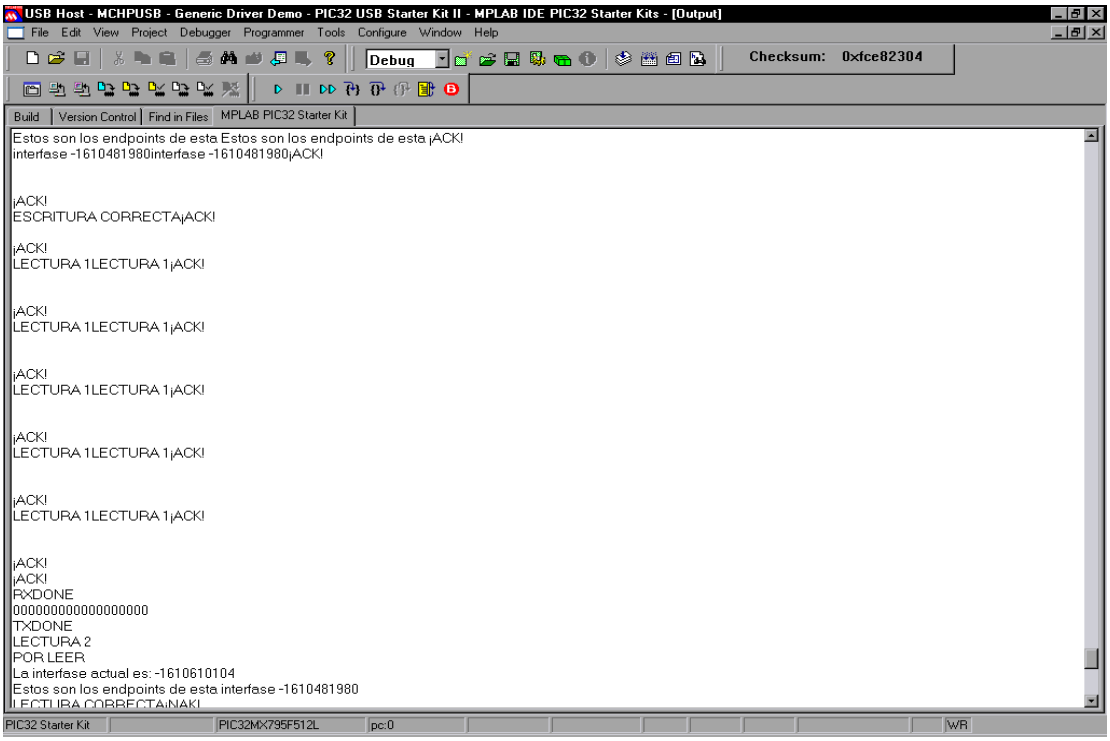


Figura 3.34 "Lectura" 1.

En la figura 3.35 se observa que cuando `Lectura` iguala su valor a 2, el dispositivo siempre responde con paquetes NAK, y lo sigue haciendo incluso después de dejar funcionando más de 15 minutos al MCU.

El empleo de esta tarjeta concluye entonces que no acepta ya las transferencias, y por otra parte, no contesta con los mismos bytes que envía a una computadora (el búfer de respuesta está completo de bytes igualados a cero, diferentes a los bytes apreciados en las capturas con Wireshark).

El código modificado de la aplicación, del controlador y del USB Stack es mostrado en la sección 1 del apéndice del presente documento.

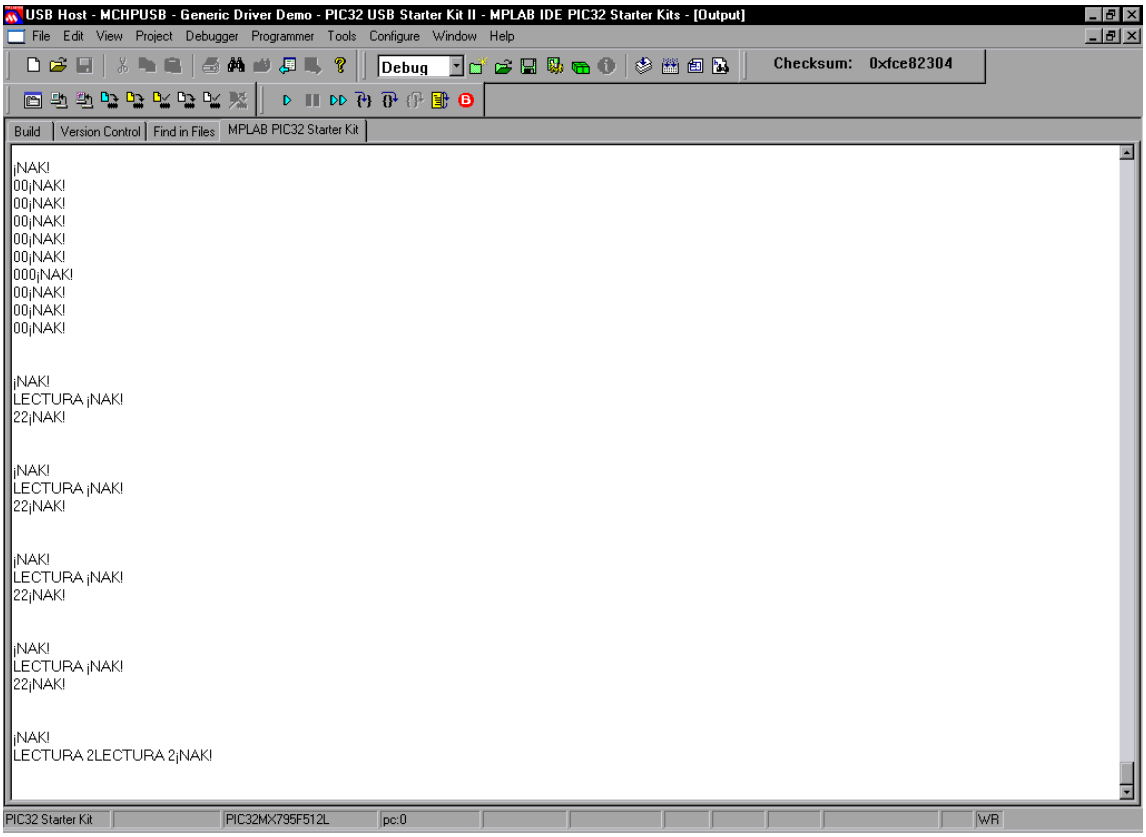


Figura 3.35 Paquetes NAK permanentes tras la segunda "lectura".

- **3.5.4.1 ENUWI G2**

Con la ENUWI G2 se tiene el mismo problema que con la anterior: no se observa que las tramas de asociación sean transferidas por 802.11x. Para agilizar las pruebas, se dejan intactas las modificaciones al USB Stack y así apreciar los paquetes Handshake USB. Con la figura 3.36 se observa que el PID y VID son encontrados, y que igualmente se inicializa el controlador correctamente.

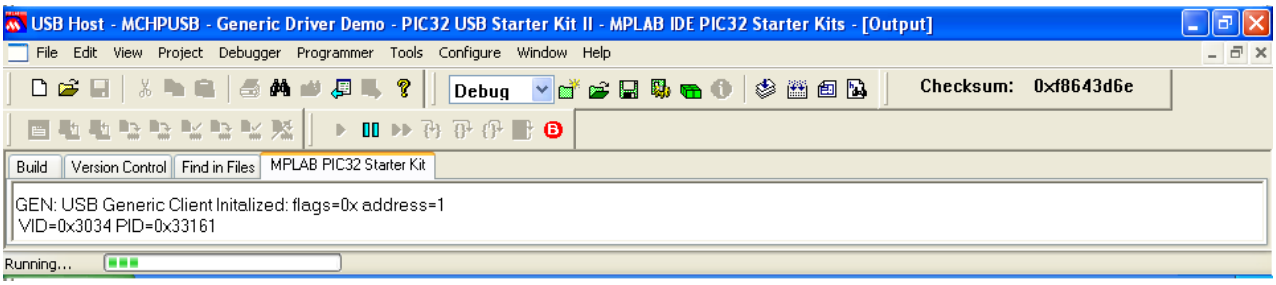


Figura 3.36 Reconocimiento de la tarjeta ENUWI G2.

A diferencia de la anterior tarjeta, cierto número de transferencias de control son realizadas antes de iniciar la transferencia IN al Endpoint 0x83, con el fin de obtener las tramas Beacon. De igual manera se intenta iniciar la transferencia OUT con el fin de mandar el primer paquete de asociación, el Probe Response. Si los endpoints siguen

ocupados al intentarse hacer nuevas transferencias, la leyenda "Sigo ocupado con la lectura/escritura" debe aparecer en la ventana Output.

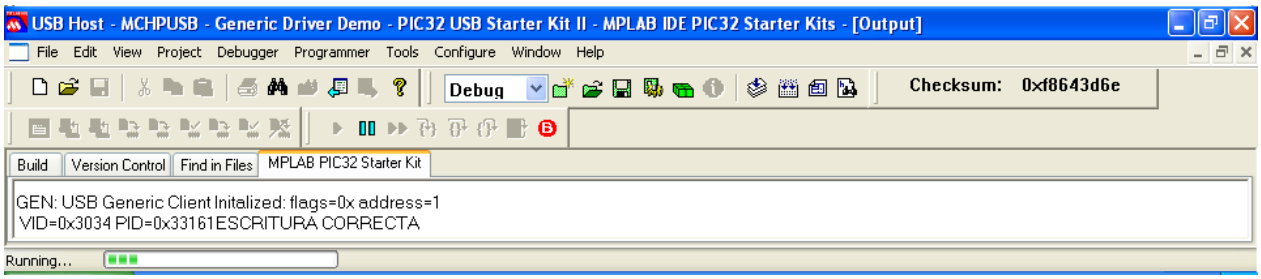


Figura 3.37 Respuesta a primera escritura.

En adelante los endpoints siguen siempre ocupados, según las leyendas programadas, y esto se debe nuevamente a una respuesta infinita de paquetes NAK por parte de la tarjeta, como se aprecia en la figura 3.38.

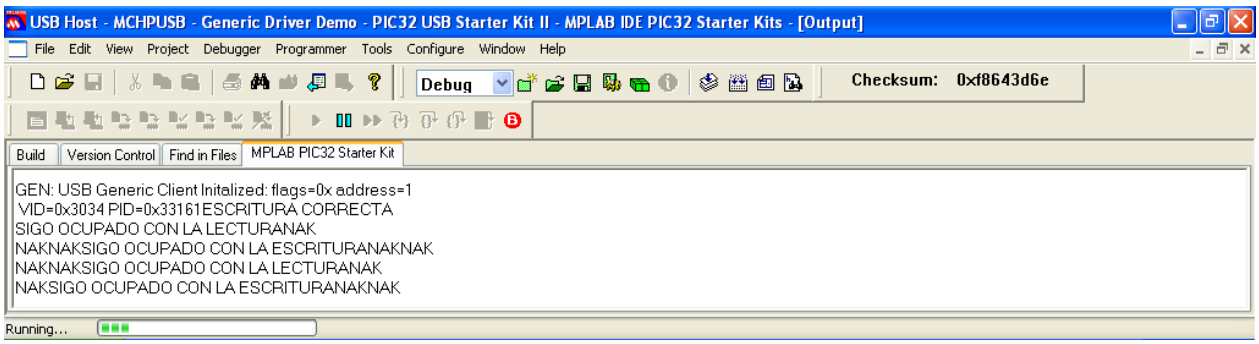


Figura 3.38 Paquetes NAK continuos luego de las primeras transferencias IN y OUT.

Ante una respuesta permanente de paquetes NAK, no es posible forzar a las tarjetas a abandonar este estado. Descartadas las posibilidades de una incorrecta programación del MCU o un incorrecto empleo del USB Stack y/o del controlador genérico, el análisis de estos resultados es expuesto en el capítulo 5 del presente trabajo.

Capítulo 4

Diseño de la aplicación mecatrónica

En este capítulo se muestran las diferentes etapas del diseño de la aplicación mecatrónica que se pretendió emplear con la comunicación inalámbrica propuesta, empleando la metodología explicada en el capítulo 2.

De los resultados obtenidos del capítulo anterior, el diseño no fue implementado físicamente. Sin embargo, podría emplearse de resolverse las problemáticas finales.

La metodología del diseño mecatrónico requiere delimitar a los objetivos finales, pero en base a una o varias necesidades que se esperan sean solucionadas con el producto final.

4.1 Identificación de las necesidades

El grupo de trabajo "Valtroniks" llegó a la conclusión de la necesidad de implementar sistemas mecatrónicos con el fin de llevar a cabo diversas tareas en el futuro.

Dichas tareas, sin embargo, requieren ser monitoreadas y manipuladas por los integrantes, de tal forma que, por el momento, las actividades de los sistemas serán directamente ejecutadas por operadores humanos desde cualquier PC dentro de una WLAN.

Para facilitar este monitoreo, se han descartado comunicaciones alámbricas: se emplearán las inalámbricas. El intercambio de información entre las aplicaciones y los operadores es, por el momento, no mayores a un millar de bytes (kB).

Debido a que no se cuenta con la capacidad de obtenerse los componentes necesarios para su empleo, no se pueden utilizar las comunicaciones industriales más comunes como ZigBee o Bluetooth.

El grupo de trabajo igualmente ha desechado la continua intervención inalámbrica implementada dentro del sistema mecatrónico pues solamente es requerida en cortos periodos de tiempo. El grupo desea que la inversión monetaria de la adquisición de los circuitos que hacen posible el intercambio de información pueda ser empleada para otros fines, es decir, que los circuitos no funcionen meramente con aplicaciones mecatrónicas. De aquí que se hayan interesado por la utilización de las tarjetas de red mencionadas en capítulos anteriores, que pueden ser desmontadas y ser empleadas en computadoras.

El grupo ha considerado microcontroladores para emplear en sus aplicaciones, y se ha interesado particularmente en los PIC32 de Microchip. Para el desarrollo del proyecto están conformes en facilitar para su empleo el PIC32 USB Starter Kit II.

La hipótesis del presente proyecto, expuesto durante el capítulo 1, interesó al grupo de trabajo, y para comprobar el correcto funcionamiento de la comunicación propuesta se requiere elaborar una aplicación mecatrónica (preferiblemente alguna que pueda desplazarse por un área establecida) y de esta forma estar seguros de la efectividad del control inalámbrico, o bien, para los trabajos a futuros con otras posibles hipótesis de solución.

El producto no será comercializado: simplemente consistirá en una demostración para los integrantes del mismo grupo de trabajo, quienes están aún más interesados en adquirir el conocimiento necesario de la manera de operar las herramientas de Microchip que se utilicen durante el desarrollo del presente proyecto.

Como se había expuesto, la aplicación no tendrá más que algunas funciones, que cumpla con las características de *mecatrónica*, por lo tanto se elige un vehículo móvil operado remotamente desde una computadora empleando la tarjeta USB de red inalámbrica, llamados AVG. Dicho vehículo móvil será de ruedas con direccionamiento diferencial.

Justificación de la elección de la aplicación

La idea de ser una aplicación móvil es reconocer que funcione en cualquier lugar dentro de un área de trabajo con cobertura de señal emitida desde un AP que emplee el estándar 802.11.

4.1.1 Especificaciones de diseño

Función principal

El propósito del vehículo es trasladarse en una zona con cobertura de una WLAN y que pueda ser controlado remotamente desde una PC con la comunicación desarrollada. Esto servirá para el fin demostrativo.

Los movimientos que debe realizar este modelo son los mostrados en la figura 4.1.

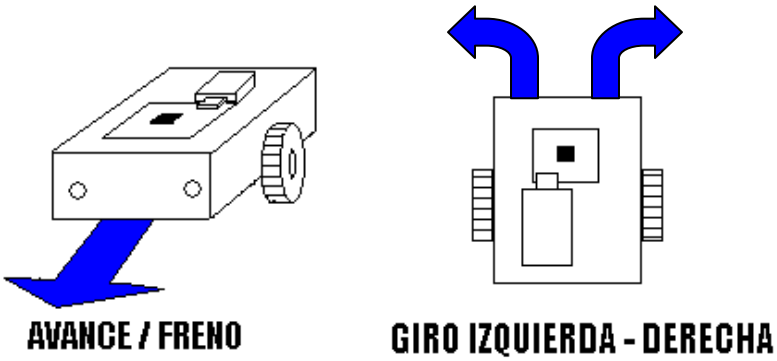


Figura 4.1 Movimientos y tareas propuestas para la aplicación.

Los giros son solamente de 90° por cada instrucción.

El vehículo debe conocer la posición espacial en la que se ubica, para no sobrepasar el área en la que se encuentra la red inalámbrica, área delimitada como un cuadrado de 9 m por lado cuyo centro es el Access Point. La total precisión es innecesaria. Esta área puede ser de concreto, tabique, hule plano o baldosas, sin desniveles ni fisuras.

Por otra parte, debe evitar colisiones frontales, en caso de haber objetos dentro del área. No existen objetos con alturas menores de 20 cm.

Materiales

Se deben emplear los materiales facilitados:

1. *PIC32 USB Starter Kit II*: Este kit consiste básicamente de una PCB cuyos principales componentes son un PIC32MX795F512L (que es el MCU que ejecutará el firmware de la aplicación), un PIC32MX440F512H (el MCU que programará y monitorea al PIC32 anterior desde MPLAB), un conector hembra diseñado por Microchip, y otros conectores físicos USB hembras de tipos diferentes: el tipo B conecta por un cable USB este kit con una PC, por el tipo A se conecta el dispositivo USB que se desea manipular. El tipo micro USB tipo B se emplea cuando el MCU se controlará en modo OTG.

2. *Adaptador USB de red inalámbrica* (TL 722n ó ENUWI G2).

3. *PIC32 I/O Expansion Board:* Esta es una tarjeta diseñada por Microchip como complemento de los PIC32 Starter Kits, debido a que éstos cuentan solamente con algunos botones y leds, insuficientes para proyectos como el actual. Tiene integrado el conector Microchip macho para los kits, y así permite interactuar con los pines de entrada y salida de sus MCU. Adicionalmente tiene reguladores que permiten salidas de 9, 5 y 3.3 V con un máximo de 1 A. La tarjeta tiene otros varios conectores para más productos de Microchip.

   Una ventaja de esta tarjeta es la posibilidad de ser energizada con una fuente de alimentación de hasta 12 V y, particularmente, permite que el PIC32 USB Starter Kit II logre suministrar hasta 500 mA al dispositivo USB por manejar.

El resto de los materiales a emplear se deja a cuestión del diseñador, considerando las otras especificaciones.

Operación

Se requiere de algunas características para el empleo de este vehículo:

1. Capacidad de apagar y prenderlo cuando se desee.
2. Capacidad de desmontar el módulo de comunicación inalámbrica, y poder emplear otros con interfaz USB.
3. Emplear el software de la comunicación inalámbrica desarrollada.

Costo del producto

El costo debe ser bajo, no mayor a la cantidad invertida en el USB Starter Kit II: $1’370.00.

Simplicidad de construcción

Deben diseñarse estructuras geométricas simples, como rectángulos, círculos, triángulos, en caso de requerirse manufacturar.

Los elementos electrónicos adicionales deben ser comunes en el mercado, o de disponibilidad en la República Mexicana, de consumo de energía no mayor a 1 A.

Tamaño

Cumpliendo los dos puntos anteriores, y adicionalmente para facilitar su transporte hacia diversas áreas se necesita un modelo de reducidas dimensiones, contempladas como menores de 50 cm.

Tiempo de operación

Se requiere que al menos, el modelo funcione por un tiempo mínimo de 30 minutos, empleando la comunicación inalámbrica.

4.2 Diseño conceptual

Para ser mecatrónico, el vehículo debe tener diferentes módulos o sistemas que permiten tanto la interdependencia y sinergia del control total de la traslación del vehículo, y al mismo tiempo poder ser seccionadas para facilitar todo el proceso ingenieril de diseño que conlleva, y cumplir adecuadamente con las especificaciones establecidas.

La toma de decisiones de manera autónoma por parte del vehículo hará que la aplicación cubra otro de los requerimientos de un producto mecatrónico: en este caso, un acercamiento a la inteligencia de un sistema artificial.

La figura 4.2 muestra el seccionamiento de las funciones que se planean para el vehículo, en el lado izquierdo se muestran las entradas del sistema, en el derecho, la salida.

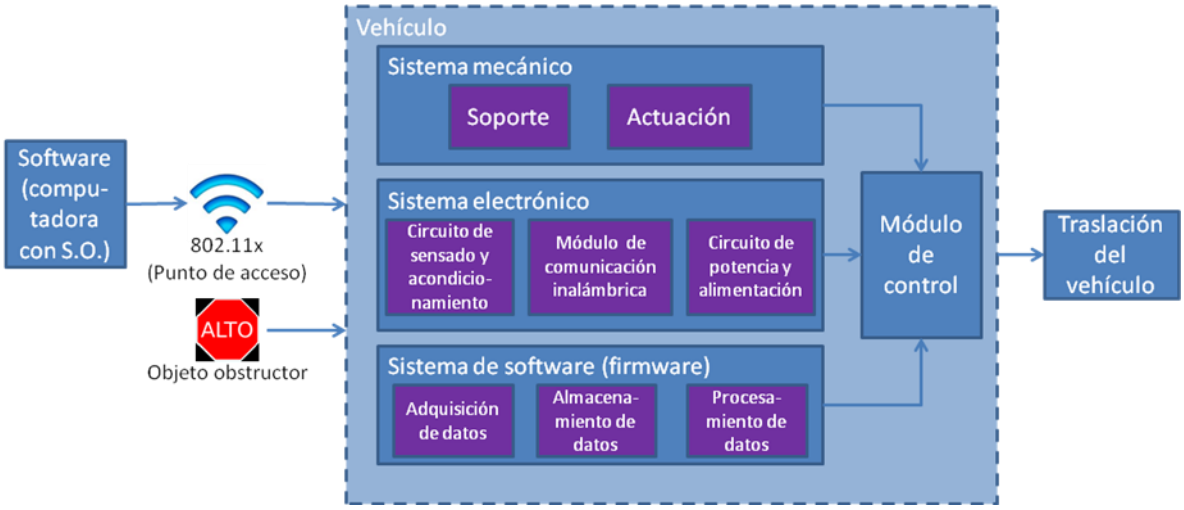


Figura 4.2 Diagrama de funciones del vehículo propuesto.

4.2.1 Sistema mecánico

El sistema mecánico consiste del soporte del vehículo y el sistema de actuación.

- **4.2.1.1 Soporte**

El soporte sirve para garantizar la protección de los circuitos, actuadores y módulos del vehículo. De las especificaciones se debe elegir un material, de fácil manufactura y de accesible en el mercado. El ambiente de trabajo no fue especificado, así que el material no requiere resistir rangos temperaturas, humedad o presiones fuera de las ambientales promedio de la República Mexicana.

Se presentan algunos materiales que destacan por sus precios accesibles de acuerdo a las especificaciones:

Aluminio: Es un metal no férreo. El aluminio no suele ser comercializado en su estado puro, sino en aleaciones con otros elementos. Uno de los más extendidos es el 6061 que combina magnesio, silicio y otros en menor proporción. Esta aleación es de manufactura de propósito general, y puede ser encontrado en latas y vehículos, entre muchos otros. Es un conductor de electricidad y de manufactura industrial, lo que implica que una adecuada manufactura no se puede realizar con herramientas de uso doméstico o de talleres artesanales. Puede conseguirse en diferentes formas, lo cual es ideal para infinidad de proyectos.

MDF (Medium Density Fibreboard): Este aglomerado de fibras de madera presenta una densidad mucho más ligera que la del aluminio, baja conductividad térmica y mala conducción de electricidad. No es resistente a la corrosión y a temperaturas altas o bajas (más de 100 o menores de 10 °C) y es vulnerable a la humedad del ambiente. Puede cortarse, perforarse y moldearse con mucha facilidad, y en realidad, con herramientas e instrumentos fáciles de conseguirse en microempresas como tlapalerías, ferreterías y madererías.

Polimetilmetacrilato: Mejor conocido como acrílico, tiene una adecuada resistencia a la tracción para proyectos básicos (en donde los pesos no superan la decena de kilogramos), es muy ligero (a volúmenes iguales, el acrílico pesa la mitad del vidrio residencial) y su manufactura adecuada requiere equipo especializado como máquinas CNC. Es ideal para diferentes condiciones ambientales. No conduce electricidad a voltajes típicos industriales (menores de 440 V). El acrílico es tan comercial que se consigue en placas en muchos lugares de venta de materiales plásticos dentro del territorio mexicano.

Tabla 4.1 Diferentes valores aproximados de propiedades físicas para los materiales propuestos.

Material	**Resistencia a la tracción máxima**	**Densidad**
Aluminio 6061	150 MPa	2700 kg/m^3
Acrílico (extruido)	48 MPa	1190 kg/m^3
MDF (4 a 6 mm)	3.3 MPa	780 kg/m^3

- **4.2.1.2 Actuación**

Para proporcionar la movilidad requerida, pueden adquirirse los siguientes actuadores:

Motores de corriente directa (C.D.): Tienen un momento de fuerza (torque o par motor) de salida bajo en comparación con otros tipos de motores (de corriente alterna, síncronos y de pasos), pero proporcional a su velocidad angular, cuyos rangos son de entre 1000 a 5000 RPM. Para lograr las revoluciones, basta con agregar una diferencia de potencial en sus devanados. El sentido en el que girará el rotor depende del sentido de la corriente eléctrica. Existen de múltiples precios, y de los tipos de motores analizados en el presente trabajo son los de menor.

Los motores de C.D. no proporcionan precisión en su posición angular a no ser que se le agregue alguna clase de sensor, como por ejemplo los *codificadores rotatorios* o *encoder*, que son discos con ranuras. Un haz de luz puede pasar por las ranuras y un sensor óptico lo detecta, como muestra la figura 4.3.

Una clase especial de motores de corriente directa son los *motorreductores*, que aumentan el torque de los motores y disminuyen la velocidad angular. Básicamente son motores de C.D. con engranajes adicionales, llamadas comúnmente *cajas reductoras*.

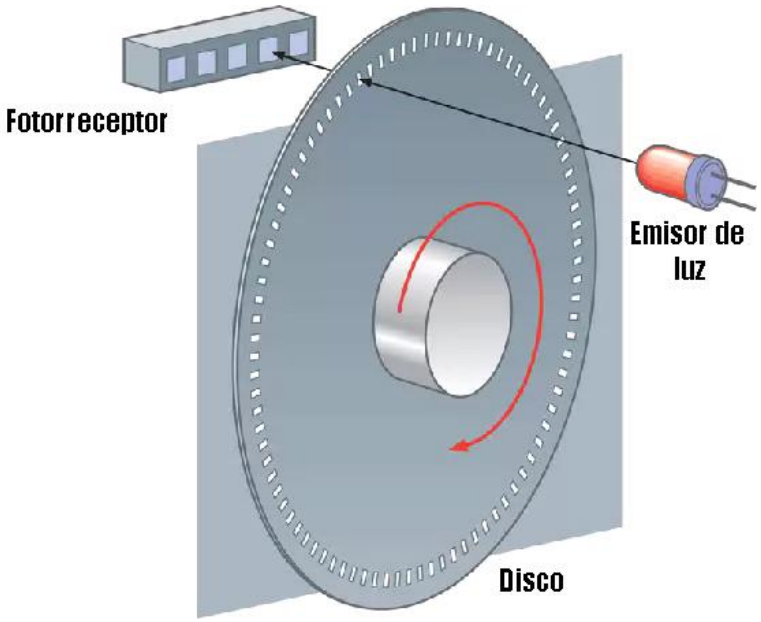


Figura 4.3 Codificador rotatorio o encoder.

Motores a pasos: Tienen una fuerza de torsión media comparada a la de otros motores expuestos en el presente trabajo. Los motores a pasos pueden existir en diferentes configuraciones eléctricas, siendo los más comunes comercialmente los bipolares y unipolares. La principal característica de estos motores es que funcionan por secuencias o trenes de pulsos de voltaje (en lugar de valores constantes de corriente) para realizar revoluciones. Gracias a esta secuencia puede determinarse automáticamente la posición angular en el que se encuentra su rotor, pero requieren un dispositivo adicional que permita realizar estas secuencias de pulsos. A diferencia de los motores de C.D., estos motores tienen un par más elevado mientras menor sea la velocidad angular.

Servomotores: Son esencialmente motores de corriente directa con reductores y un circuito de control. Con estas dos últimas características pueden moverse hacia cualquier posición angular deseada y mantenerse en ella. Para lograrlo requieren uso de la técnica conocida como *modulación de ancho de pulso*, lo que implica que requiere de algún dispositivo electrónico externo que pueda realizarla.

4.2.2 Sistema electrónico

El sistema electrónico consta de los circuitos de acondicionamiento, de potencia y alimentación y del módulo de comunicación inalámbrica.

- 4.2.2.1 Módulo de comunicación inalámbrica

El módulo de comunicación inalámbrica debe ser necesariamente uno de los dos adaptadores especificados. En la tabla 4.2 se muestran las características más relevantes de ambos.

Tabla 4.2 Características de los módulos de comunicación inalámbrica especificados.

Características	TL 722n	ENUWI G2
Corriente de alimentación	500 mA	500 mA
Dimensiones	93.5 mm x 26 mm x 11 mm	12.5 mm x 78 mm x 28 mm
Estándar soportado	IEEE 802.11n, IEEE 802.11g, IEEE 802.11b	IEEE 802.11g, IEEE 802.11b
Velocidad de señal	11n: Hasta 150Mbps (dinámico) 11g: hasta 54Mbps (dinámico) 11b: hasta 11Mbps (dinámico)	11b: hasta 11Mbps 11g: hasta 54 Mbps
Seguridad inalámbrica	64/128 bit WEP, WPA-PSK/WPA2-PSK	64/128 bit WEP, WPA-PSK/WPA2-PSK

Debido a las características vistas en el capítulo anterior, la información en el tráfico USB de la tarjeta ENUWI G2 es más reducida, razón por la cual se elige.

- **4.2.2.2 Circuito de sensado y acondicionamiento**

El circuito de acondicionamiento garantiza la adecuada transmisión de las señales eléctricas entre de los sensores.

Los circuitos integrados más esenciales que se emplean para el circuito de acondicionamiento de esta aplicación son amplificadores operacionales. Uno de los más comunes en el mercado son los denominados LM324, que incluye 4 OPAM en un solo encapsulado.

Se requiere de un sensor que sea capaz de detectar obstrucciones frente al vehículo. Existen de diferentes tipos según el principio físico que emplean, cada uno con diferentes características, como muestra la tabla 4.3.

Tabla 4.3 Características de los tipos de sensores de proximidad.

Tipo	Características
Capacitivos	Detectan objetos con alta constante dieléctrica (15 o más), imprecisos con los de baja (plásticos). Comunes en el mercado. Costos elevados (acorde a la especificación del precio). Detectan objetos a distancias no mayores de 50 cm.
Inductivos	Detectan objetos metálicos. Comunes en el mercado. Bajo costo (acorde a las especificaciones).
Ultrasónico	Sólo accesibles en algunos estados de la República Mexicana. Detectan objetos a distancias largas (1 m o más). Costo muy elevado (acorde a las especificaciones). Funcionan bajo algunas condiciones de suciedad, polvo o de baja luminosidad.
Fotoeléctricos	Bajo costo (acorde a las especificaciones). Comunes en el mercado. Distancia de detección de menos de 1 m. Detectan cualquier material opaco.

- **4.2.2.3 Circuito de alimentación y de potencia**

Este circuito depende especialmente del componente más esencial: la batería.

Otro componente electrónico dentro de este circuito que será requerido es de los denominados drivers, que son circuitos integrados que proporcionan la potencia requerida para movilizar a muchos motores que emplean menos de 24 V. En posteriores subcapítulos se retoma este concepto.

- **4.2.2.4 Módulo de control**

El circuito de control es representado por el PIC32 USB Starter Kit II, especificado. Como se mencionó, el MCU que incluye es el PIC32MX795F512L, que emplea un conector físico especialmente diseñado para el circuito de expansión, el PIC32 I/O Expansion Board. Esta tarjeta expansión permite el empleo de los pines de los puertos de entrada y salida del MCU.

4.2.3 Selección de conceptos

Se emplea la técnica de matrices de decisión, con el fin de seleccionar a los componentes más relevantes, con una escala con valores de 2, 4 y 6. Una valor de 2 define una característica poco viable, 4 define una posible selección y 6 define al que presenta una mejor característica.

Los criterios son explicados a continuación:

Actuadores: Se requiere un tipo de actuador que sea común en el mercado para su compra. Se debe procurar escoger uno que tenga el menor precio, y que al mismo tiempo proporcione un torque que pueda permitir trasladar el peso total del vehículo. Esto implica por tanto, que el peso de dicho tipo de motor debe tratar de ser el menor. Puesto que el vehículo debe reconocer la ubicación aproximada de su posición espacial, el tipo de actuador debe tener un control que permita reconocer las distancias lineales recorridas por el vehículo

Material del soporte: Para las partes mecánicas que conformarán al soporte, se requiere un material que pueda manufacturarse con herramientas de escaso o nulo soporte. Debe tratar de ser el material menos costoso. Para no requerir el torque máximo del motor por elegir, se requiere al material que no tenga la más elevada densidad, y en menor medida, que resista algunos esfuerzos adicionales de los del peso del vehículo.

Sensor: El tipo del sensor que se requiere debería ser común en el mercado, debe tener el precio más bajo posible, debe tener precisión en la distancia con la que detecta el objeto, y preferentemente no ocupar volúmenes grandes, mayores a 5 cm.

Las matrices de decisión son apreciadas en las tablas 4.4, 4.5 y 4.6.

Tabla 4.4 Matriz de decisión para la selección de actuador.

Actuadores	Más común en el mercado (15%)		Menor precio (35%)		Menor peso (20%)		Mayor torque (20%)		Mejor control (10%)		Puntuación ponderada	Intervalo
	calif.	porcent.	calif.	porcent.	calif.	porcent.	calif.	porcent.	calif.	porcent.		
Motorreductor	6	0.9	6	2.1	6	1.2	6	1.2	2	0.2	**5.6**	**1**
Servomotor	4	0.6	2	0.7	4	0.8	6	1.2	6	0.6	3.9	3
Motor a pasos	6	0.9	4	1.4	4	0.8	4	0.8	4	0.4	4.3	2

Tabla 4.5 Matriz de decisión para la selección de material mecánico.

Material	Manufactura menos costosa (20%)		Menor precio (35%)		Menor densidad (35%)		Mayor resistencia mecánica (10%)		Puntuación ponderada	Intervalo
	calif.	porcent.	calif.	porcent.	calif.	porcent.	calif.	porcent.		
MDF	6	1.2	6	2.1	6	2.1	2	0.2	**5.6**	**1**
Acrílico	4	0.8	4	1.4	4	1.4	4	0.4	4	2
Aluminio	4	0.8	2	0.7	2	0.7	6	0.6	2.8	3

Tabla 4.6 Matriz de decisión para la selección de sensor de proximidad.

Sensor	Más común en el mercado (20%)		Menor precio (50%)		Mayor precisión (20%)		Menor volumen promedio (10%)		Puntuación ponderada	Intervalo
	calif.	porcent.	calif.	porcent.	calif.	porcent.	calif.	porcent.		
Inductivo	6	1.2	4	2	2	0.4	4	0.4	4	2
Capacitivo	4	0.8	4	2	4	0.8	4	0.4	4	2
Ultrasónico	2	0.4	2	1	6	1.2	2	0.2	2.8	3
Optoelectrónico	6	1.2	6	3	4	0.8	6	0.6	**5.6**	**1**

De las puntuaciones obtenidas de las matrices anteriores, se usarán:

Como material, al MDF.
Como actuadores, a los motorreductores
Como sensor de proximidad, alguno de tipo optoelectrónico.

4.3 Diseño de configuración

El esquema principal del proyecto es el que se presenta a continuación (Figura 4.4). El sensor debe estar orientado hacia la parte frontal del vehículo. Las líneas representan conexiones electrónicas. Las punteadas representan conexiones sin intercambio de información (circuito de distribución de electricidad).

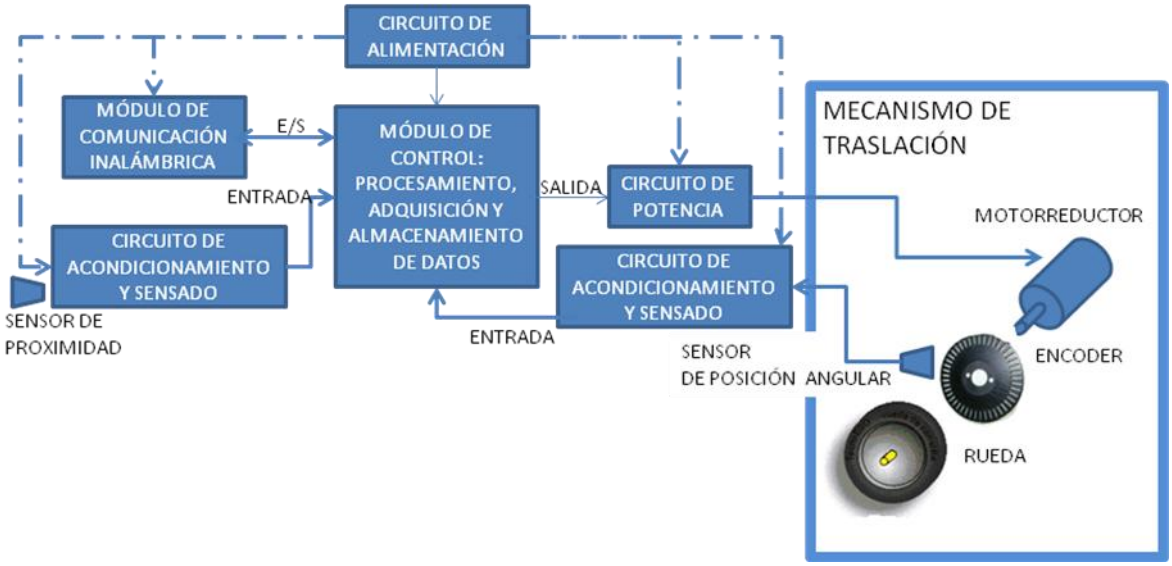


Figura 4.4 Interrelación entre los componentes.

Para visualizar aproximadamente la configuración de estos componentes, se empleará la convención que propone la tabla 4.7.

Tabla 4.7 Números identificadores de los componentes

Componente	Número identificador
PIC32 I/O Expansion Board con PIC32 USB Starter Kit II	1
Batería	2
Circuitos de sensado, acondicionamiento y potencia	3
Soporte	4

El diseño de las siguientes ilustraciones de las posibles configuraciones fue realizado con el software CATIA v5 R19.

1ª configuración

El vehículo consiste esencialmente en una plataforma rígida horizontal en cuya parte superior se colocan todos los componentes electrónicos. Las ruedas y los motores son ubicados en la parte inferior para hacer el contacto con el suelo y trasladar al vehículo.

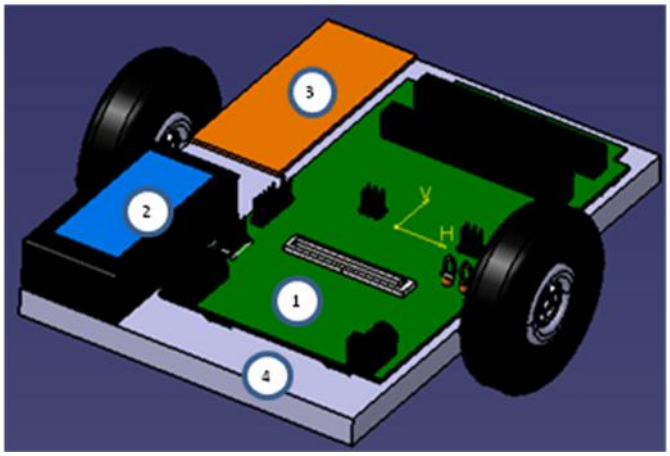


Figura 4.5 Primera configuración.

2ª configuración

El vehículo consiste plataforma rígida horizontal en cuya parte superior se colocan cuatro columnas pequeñas. Sobre éstas se coloca al I/O Expansion Board, y debajo de ella el resto de los componentes electrónicos. Las ruedas y los motores serán ubicados en la parte inferior para hacer el contacto con el suelo y trasladar el vehículo.

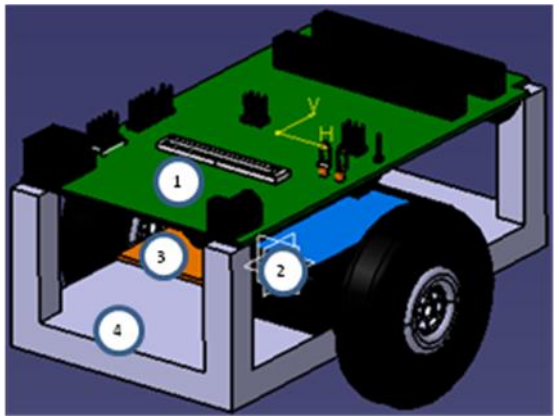


Figura 4.6 Segunda configuración.

3ª configuración

El vehículo consiste plataforma rígida horizontal y de forma perpendicular en su parte superior se coloca al I/O Starter Kit. En el espacio sobrante de la plataforma se colocan al resto de los componentes electrónicos. Las ruedas y los motores serán colocados en la parte inferior para hacer el contacto con el suelo y trasladar el vehículo.

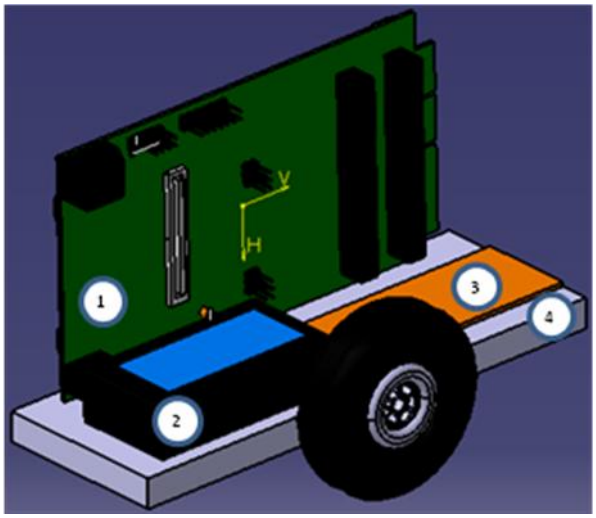


Figura 4.7 Tercera configuración.

Para poder determinar la mejor configuración se requiere conocer qué parámetros son los que se necesitan tomar en consideración:

Menor necesidad de manufactura: Mientras menos piezas y menos figuras diferentes de las geométricas se requieran, la construcción del vehículo requerirá menos tiempo.

Menor uso de material: A mayor material empleado, el peso del vehículo aumenta y más material se necesitará comprar, por lo que una adecuada configuración para el presente proyecto requerirá el menor material.

Mayor adaptabilidad: Aunque el proyecto requiere muy pocos componentes, el PIC32 I/O Expansion Board tiene la versatilidad de emplearse en más tareas mientras más componentes pueden ser conectados a éste. Una configuración que pueda adaptarse a nuevos módulos de conexión electrónica es el único que puede ser eficiente a nuevas necesidades o a futuros proyectos.

Se hará nuevamente una matriz de decisión para elegir la mejor configuración (tabla 4.8).

Tabla 4.8 Matriz de decisión para la selección de configuración de los componentes principales.

Configuración	Menor necesidad de manufactura (40%)		Menor uso de material (50%)		Mayor adaptabilidad (10%)		Puntuación ponderada	Rango
	calif.	porcent.	calif.	porcent.	calif.	porcent.		
Primera	6	2.4	2	1	4	0.4	3.8	2
Segunda	4	1.6	4	2	2	0.2	3.8	2
Tercera	4	1.6	6	3	6	0.6	**5.2**	**1**

De la matriz de decisión de la tabla 4.8, se tiene que la mejor configuración es la tercera de las propuestas.

La configuración puede sugerir que se puede generar un torque al aplicársele una fuerza perpendicular al I/O Expansion Board, que podría hacer caer al vehículo hacia alguno de sus lados. Esto se soluciona parcialmente con el grosor de las llantas o ruedas, además de que puede contrarrestarse con el peso del resto de los componentes.

4.4 Diseño de detalle

El diseño de detalle es un proceso que debe iniciarse en repetidas ocasiones. Esto se debe a que detallar una parte del modelo puede repercutir directamente en las variables anteriormente analizadas. En muchas ocasiones, se deben asumir conceptos para continuar con las siguientes etapas.

4.4.1 Diseño del mecanismo de traslación

El motorreductor tiene un eje que proporciona un par motor cuyo valor por lo general es proporcionado al comprador en sus especificaciones. Por simplicidad, el eje de la salida del motorreductor, entonces, hará girar directamente a la rueda, evitando manufacturar engranes o bandas, o tener que adquirirlos.

En el eje se planea colocar un encoder. El sensor más fácil de emplear para este propósito es un emisor-receptor (fotodiodo y fototransistor) infrarrojos en forma de herradura, comunes en muchas tiendas de electrónica, denominado optointerruptor. Se elige el modelo ITR8102 del catálogo de la tienda especializada Steren.

El ancho del disco del encoder será acotado por la misma separación de este sensor entre su fotodiodo y su fototransistor.

- **4.4.1.1 Cálculo de los parámetros mecánicos**

Se empieza, entonces, por la selección de los componentes más primordiales del vehículo: los motores.

Cálculo del momento de fuerza y la carga máxima

Para determinarlos, se requiere estimar la carga máxima con la que trabajarán. La tabla 4.9 muestra la aproximación del peso total del vehículo.

Tabla 4.9 Suma de los pesos de los diferentes componentes del vehículo.

Componente	**Peso aproximado [kg]**
PIC32 I/O Expansion Board	0.070
PIC32 USB Starter Kit II	0.040
ENUWI G2	0.030
Circuitos impresos de acondicionamiento, potencia, sensado y distribución	C_i
Batería	B
Motores y ruedas	M
Soporte y armazón	S
Total	**0.140 + C_i + B + M + S**

Para conocer las características que se deben conseguir en un motor se requiere entonces realizar algunos cálculos. El diagrama de cuerpo libre y las variables físicas del vehículo son mostrados en las figura 4.8 y 4.9.

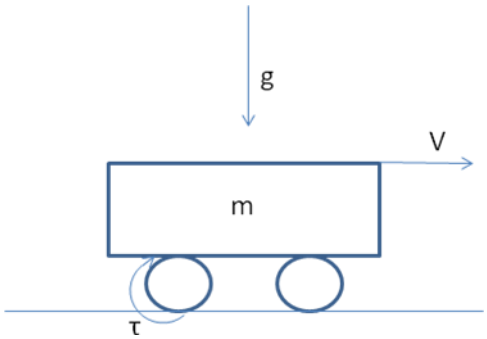


Figura 4.8 Variables físicas que interactúan en el vehículo.

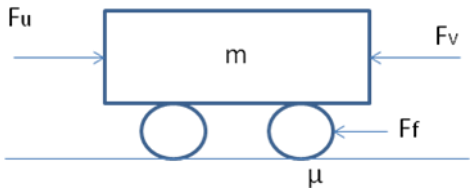


Figura 4.9 Diagrama de cuerpo libre del vehículo.

Del diagrama de cuerpo libre se sabe que la fuerza con la que se debe mover al motor debe vencer a la inercia F_v (o F_i en adelante) como a la fuerza de fricción estática F_f entre el suelo y las ruedas. Se puede omitir la fricción con el aire puesto que el área frontal del vehículo es despreciable y la velocidad con la que trabaja el vehículo es igualmente muy pequeña (menos de 10 cm/s).

$$F_u = F_i + F_f \tag{1}$$

La fuerza inercial se calcula mediante la expresión (2).

$$F_i = m_v\, a_v \tag{2}$$

Donde m_v y a_v son la masa del vehículo y la aceleración con la que se pretende trabajar al vehículo.

La aceleración del vehículo no es nula al momento de ponerse en marcha. Se puede simplificar el cálculo de la aceleración mediante la expresión (3), puesto que la masa es considerablemente pequeña.

$$a_v = \frac{(V_1 - V_0)}{t} \tag{3}$$

Donde V_1 es la velocidad final, V_0 la velocidad inicial y t el tiempo en el que se pretende alcanzar dicha velocidad.

Una aceleración muy grande para un vehículo de esta índole es la que se propone con los valores siguientes: $V_1 = 5$ cm/s, $V_0 = 0$ cm/s y $t = 1$ s. Si se sustituyen en la expresión (3) se obtiene:

$$a_v = \frac{\left(5\left[\frac{cm}{s}\right] - 0\left[\frac{cm}{s}\right]\right)}{1\,s} = 5\left[\frac{cm}{s^2}\right]$$

Esta aceleración es grande para los propósitos de este proyecto, sin embargo, se mantiene en este valor con la finalidad de obtener valores límite superiores para el cálculo del torque.

Puesto que la masa aproximada del vehículo es de más de 140 g (más el peso de todos los componentes), se sustituye la expresión obtenida de la tabla 4.4.1 y el valor obtenido de la aceleración del vehículo en la expresión (2).

$$F_i = \left(0.140 + C_i + B + M + S\right)(0.050)\left[\frac{kg\,m}{s^2}\right]$$
$$= 7 \times 10^{-3}\,[N] + 50 \times 10^{-3}\left(C_i + B + M + S\right)[N] \quad (4)$$

La fuerza de fricción estática es expresada como:

$$F_f = \mu\, \mathrm{N} \quad (5)$$

Donde *N* es la magnitud de la fuerza normal opuesta al peso del vehículo, y μ es el coeficiente de fricción estática entre las ruedas y el suelo.

Las ruedas son de plástico, y en diversas literaturas se emplean valores cercanos -entre el plástico y concreto- para el coeficiente de fricción, menores a 0.4. Como se desconoce el tipo de suelo en el que se usará el vehículo, se emplea un valor más alto.

$$\mu = 0.45$$

La magnitud de la fuerza *N* se calcula mediante la expresión (6):

$$\mathrm{N} = m_v\, g \quad (6)$$

Donde *g* es la magnitud de la aceleración de la gravedad terrestre. En la mayor parte de la literatura este valor es igual a 9.81 m/s^2 a nivel del mar. Este valor es el que se empleará, ya que es el máximo superior que puede adquirir la gravedad en la Tierra.

Sustituyendo tanto este valor como la expresión obtenida para la masa del vehículo en la expresión (5), se obtiene la siguiente:

$$F_f = 0.45\left(0.140 + C_i + B + M + S\right)(9.81)\,[N]$$
$$= 4.4145\left(C_i + B + M + S\right)[N] + 0.61803\,[N] \quad (7)$$

Con esta última, y junto con la (4), pueden sustituirse en la expresión (1).

$$F_u = 7 \times 10^{-3}\,[N] + 50 \times 10^{-3}\left(C_i + B + M + S\right)[N]$$
$$+ 4.4145\left(C_i + B + M + S\right)[N] + 0.61803\,[N]$$
$$= 625.03 \times 10^{-3}\,[N] + 4.4645\left(C_i + B + M + S\right)[N] \quad (8)$$

Esta es la fuerza máxima que se debe aplicar para hacer avanzar al vehículo.

Un sólo motor tiene que proporcionar esta fuerza, ya que durante los giros del vehículo es uno sólo quien debe realizar la tracción (direccionamiento diferencial). La fuerza se relaciona con el momento de fuerza del motor mediante la siguiente expresión:

$$\tau = F r \tag{9}$$

Donde r es el radio de la rueda del vehículo, y τ es el par aplicado a la rueda.

La fuerza que se requiere es la encontrada en la expresión (8), por lo que se tiene finalmente el momento de fuerza que se necesita encontrar en un motor para hacer la tracción del vehículo.

$$\tau = r\left(625.03 \times 10^{-3} + 4.4645\left(C_i + B + M + S\right)\right)[Nm] \tag{10}$$

- **4.4.1.1 Selección de los motores**

En el cálculo realizado se observa entonces que el par motor puede ser reducido por el factor que representa al radio del eje.

Para continuar con el resto del diseño de detalle, se harán los cálculos con ruedas de radio igual a 3 cm. Las ruedas son muy fáciles de conseguir en el mercado de diversas dimensiones, de múltiples acoples y hechos con diferentes materiales. Particularmente estas ruedas son conseguidas en su venta en Internet, específicamente en el portal comercial www.mercadolibre.com.mx.

Los pesos de las ruedas son considerados dentro del soporte del vehículo.

De lo cual, el cálculo se facilita y se tiene una primera estimación del torque necesario en un motor.

$$\tau = 0.03\left(625.03 \times 10^{-3} + 4.4645\left(C_i + B + M + S\right)\right)[Nm] \tag{11}$$

$$\tau = 18.75 \times 10^{-3}[Nm] + 133.93 \times 10^{-3}\left(C_i + B + M + S\right)[Nm] \tag{12}$$

El valor constante de la expresión (12) es el torque mínimo necesario para soportar los componentes principales. Sin embargo, el resto de los componentes cuentan con su propio peso, por lo que no pueden ignorarse.

El motorreductor escogido será el adquirido en el catálogo de la distribuidora Robodacta, con código de artículo B02 1:280. Las características de este motor mostradas en la tabla 4.10.

Tabla 4.10. Principales características del motor seleccionado.

Característica	Valor
Voltaje aliment. máximo	12 V
Corriente máx. consumida	600 mA (a 5 V)
Corriente sin carga	80 mA
Torque	4.6 kg·cm (a 5 V)
RPM	30 (a 5 V)
Peso	32 g

Figura 4.10 Motorreductor con código B02 1:280 de Robodacta.

El torque de este motor es muy alto para la aplicación, sin embargo, se elige porque puede asegurarse que la velocidad reducida es apropiada para mover a las ruedas.

4.4.2 Diseño electrónico

- 4.4.2.1 Selección del sensor de detección de objetos

El sensor elegido para el proyecto es el sensor GP2Y0D810Z0F de Sharp Corp., que se puede adquirir en la distribuidora Robodacta.

Las características más relevantes de este sensor son exhibidas en la tabla 4.11.

Con los componentes electrónicos más esenciales ya seleccionados, es posible elegir la batería.

Tabla 4.11 Especificaciones más notables del sensor de objetos.

Característica	Valor
Rango de detección	2 cm – 10 cm
Corriente consumida	5 mA
Masa	1.3 g
Voltaje de operación	2.7 V – 6.2 V
Dimensiones	21.6 mm x 8.9 mm x 10.4 mm
Voltaje de salida (presencia de objeto)	2.7 V – 6.2 V
Voltaje de salida (sin presencia de objeto)	0 V – 1.2 V

- **4.4.2.1 Selección de la batería**

Los parámetros de la selección de la batería son esencialmente tres: voltaje, corriente y duración suficiente.

Se necesita obtener la corriente que demandarán en total todos los circuitos empleados, como muestra la tabla 4.12.

Tabla 4.12 Estimación de la corriente empleada por los principales componentes electrónicos.

Componente	Corriente
PIC32 USB Starter Kit II	100 mA
ENUWI G2 (Puerto USB Host)	500 mA
Motores (2)	670*2 mA
Sensor de proximidad	5 mA
Total	**1945 mA**

En realidad los motores no consumen la corriente calculada en la tabla 4.11 puesto que sólo la consumen cuando actúan con el torque máximo. Las gráficas de caracterización del motor para estimar la corriente, velocidad o par, tampoco son proporcionadas por el distribuidor. Se puede hacer una estimación de la corriente consumida con el par requerido por la aplicación aproximándolas por medio de interpolaciones.

Esto es válido bajo cierta condición, ya que la corriente y el torque tienen un comportamiento casi lineal hasta el 50% del valor de la corriente nominal en la gran mayoría de los motores de C.D. Se asume, para realizar la función par-corriente, que el mínimo de corriente requerida es de 80 mA sin un torque que vencer, y que el máximo es de 600 mA al torque especificado de 4.6 kg·cm (451.1×10^{-3} [N m]).

Una función que describe este comportamiento es entonces la expresión (13), donde τ es el momento par en N·m, e i_m es la corriente consumida por el motor en mA.

$$i_m = \left(1152\left[\frac{mA}{Nm}\right]\right)\tau\,[Nm] + 80\,[mA] \quad \textbf{(13)}$$

Para los propósitos actuales, se tiene que el torque mínimo necesario obtenido es de 18.75×10^{-3} [N·m]. Se asume que para el resto de los componentes se requerirá un total de 70×10^{-3} [N·m] (una cifra alta, pero válida en caso de que el soporte resulte ser demasiado pesado, la batería tenga que ser muy grande, etc).

La corriente resultante resulta ser de 160 mA. Asumiendo un porcentaje de error aproximado de 5%, se tiene que la corriente consumida es aproximadamente de 170 mA.

Por lo tanto, una observación al usuario es que este vehículo no podrá emplearse en lugares cuyo coeficiente de fricción exceda el valor de 0.45.

La nueva suma es mostrada en la tabla 4.13.

Tabla 4.13 Estimación de la corriente requerida por el vehículo al torque necesario.

Componente	Corriente
PIC32 USB Starter Kit II	100 mA
ENUWI G2 (Puerto USB Host)	500 mA
Motores (2)	170*2 mA
Sensor	5 mA
Total	**945 mA**

Puede entonces elegirse una batería de menores prestaciones. Se selecciona a la batería con código PRT-10472 de Robodacta. Con estos nuevos datos es posible seguir con el diseño del resto de los circuitos.

Tabla 4.14 Especificaciones más notables de la batería.

Característica	Valor
Voltaje	7.4 V
Corriente hora	1000 mAh
Masa	85 g
Dimensiones	70mm x 35mm x 18mm

- **4.4.2.2 Diseño de los circuitos de potencia y de sensado**

El MCU del USB Starter Kit II no proporciona la suficiente corriente requerida para alimentar al motor. Se procede entonces a suministrar la corriente por medio de circuitos integrados, como los puentes H o los llamados *drivers* (controladores). El voltaje proporcionado por la batería no excede del valor máximo de los motores: puede ser conectada directamente al driver, o ser energizado mediante las salidas del mismo I/O Expansion Board pertinentes.

El driver elegido es el SN754410NE, una versión mejorada del comercialmente extendido L293D, que lo sustituye pin a pin, por lo que podría emplearse de manera equivalente. La función del driver es amplificar el voltaje y corriente de la señal de los pulsos de voltaje de los pines de salida del microcontrolador. El driver también maneja el sentido de giro del motorreductor y su marcha o detención de éste.

El sensor de detección de objetos tiene integrado una PCB muy pequeña que solamente requiere la conexión de tres pines: V_{IN}, V_{OUT} y GND. V_{OUT} es la señal digital que indica la presencia de un objeto o su ausencia. Es recomendable, de cualquier manera, acondicionar la señal para obtener niveles de voltaje lógicos "altos" TTL (5 V), además de acoplar impedancias.

El PIC32 I/O Expansion Board contiene tres reguladores de voltaje de 5 V, 9 V y 3.3 V. Las salidas de estos reguladores tienen una salida máxima de 1 A. Se aprovechan estas salidas para alimentar al driver, al mismo puerto USB, a los sensores y al resto de los componentes de los circuitos, alimentando directamente al Expansion Board con la batería.

El diseño electrónico se pretende que sea simple al no incluir mas que unos pocos componentes. La conexión del driver con el PIC I/O Starter Kit II, el sensor, los reguladores y la batería es expuesta en el apéndice 2.1.

- **4.4.2.3 Diseño de los circuitos de acondicionamiento**

El sensor Sharp es alimentado con 5 V, por lo que su voltaje de salida al detectar un objeto será de aproximadamente 5 V.

Este voltaje de salida es mandado a uno de los pines de algún puerto del MCU. Sin embargo, con el fin de disminuir la potencia de entrada en comparación de la de salida del sistema, se acopla este voltaje a un amplificador operacional en modo no inversor, mostrado en la figura 4.11.

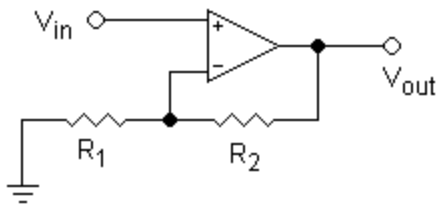


Figura 4.11 Amplificador operacional en configuración no inversora.

La expresión matemática para obtener el voltaje de salida del OPAM es:

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in} \quad \textbf{(14)}$$

Se elegirá a R_1 y R_2 de 10 kΩ, y de ser así, sustituyendo en la expresión anterior.

$$V_{out} = \left(1 + \frac{10 \times 10^3}{10 \times 10^3}\right)(5)\ [V] = 10\ [V]$$

La alimentación positiva del OPAM será de 5 V, por lo que el voltaje de salida es de solamente 5 V. Debido a que los voltajes de desvío de entrada y la velocidad de respuesta del OPAM son pequeños en contraste con los valores de voltaje deseados y los tiempos de las revoluciones de los motorreductores, no se toman en consideración para el diseño.

El esquema de la figura 4.12 muestra la manera en que se emplean los optointerruptores, que tendrán igualmente su propio circuito de acondicionamiento.

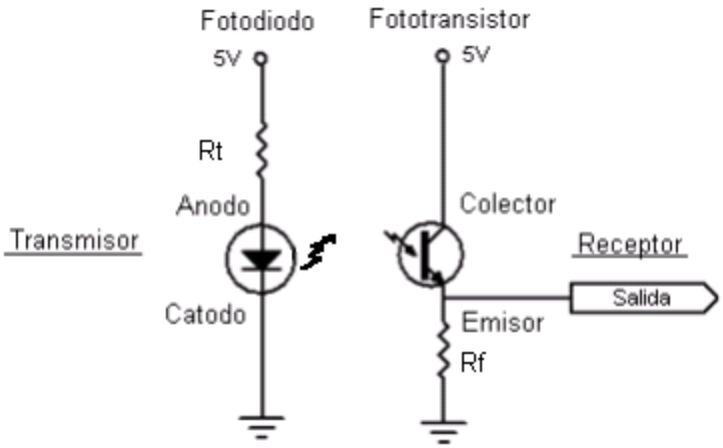


Figura 4.12 Esquema electrónico del sensor.

La corriente empleada para los transmisores es acotada por la resistencia *Rt*. Los rayos infrarrojos llegan al fototransistor, cerrando el circuito y dejando el paso a la corriente. La diferencia de potencial en *Rf* será de 5 V. Estos 5 V son los que se recibirán en los circuitos de acondicionamiento, funcionando de la misma manera que el propio sensor de detección de objetos.

El diagrama electrónico mostrado en la figura 4.13 muestra este funcionamiento en conjunto.

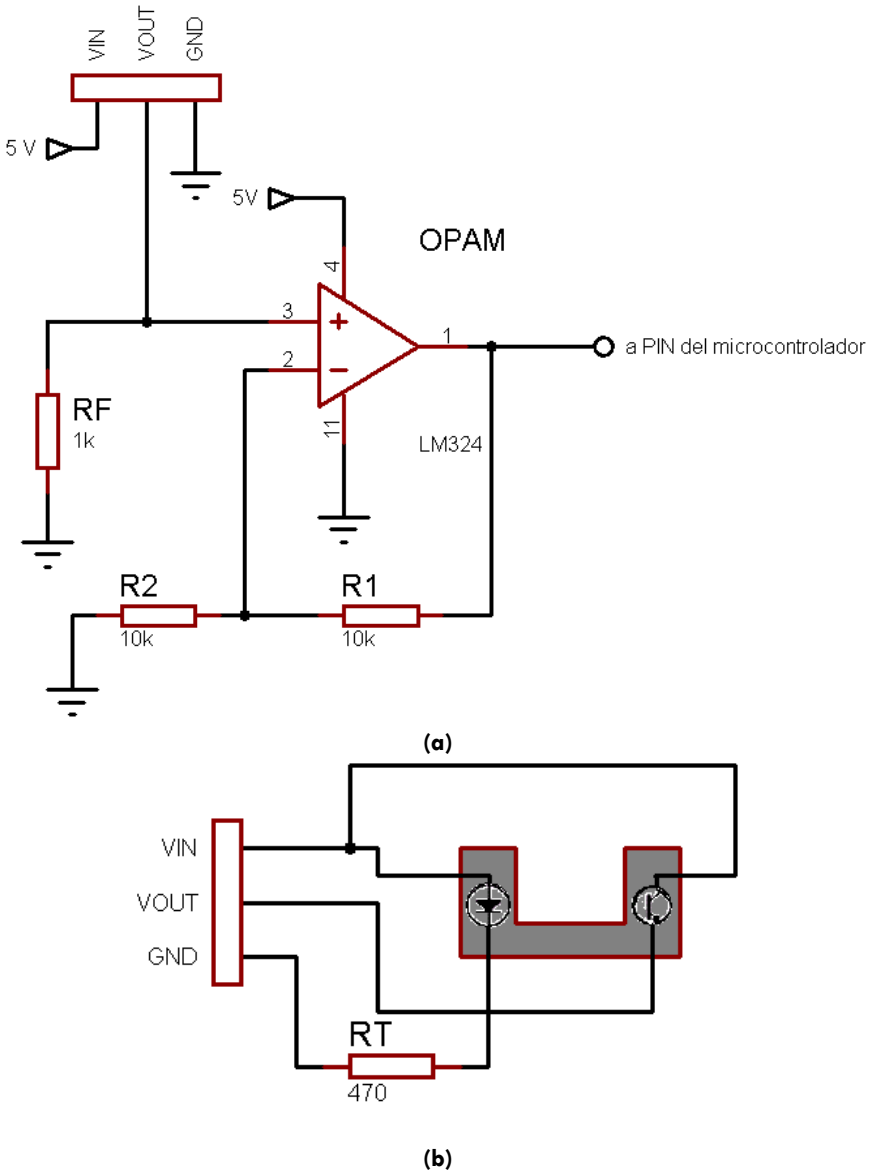


Figura 4.13 Diagrama electrónico del circuito de acondicionamiento para el sensor del encoder (a) y de la conexión del optointerruptor (b).

- **4.4.2.4 Diseño de la PCB principal**

La PCB de los circuitos de sensado, potencia, acondicionamiento y distribución debe ser lo suficientemente pequeña para que pueda incorporarse en el soporte del vehículo; las pistas son de ancho aproximado a 1 mm facilitando su elaboración en tarjeta fenólica y emplear la técnica de "planchado". La figura 4.14 muestra la PCB principal terminada, visualizada con el software POV Ray. El circuito impreso es mostrado en el apéndice 2.2 y fue diseñado con el software EAGLE.

Como se ve en la figura 4.14, a esta PCB se le incorporan headers de pines y mini molex machos con el fin de distribuir la energía eléctrica por medio de cables a los componentes que puedan estar lejanos, como los motores y los sensores. Más adelante se expone con detalle las características de estos cables.

Figura 4.14 PCB principal.

Dos PCB más pequeñas son diseñadas para el montaje de los sensores de los encoder al soporte del vehículo (figura 4.15). Nótese que a los optointerruptores se les corta la placa en donde tienen su barreno para los tornillos de ajuste. El esquemático de ellas es (b) representado en la figura 4.13.

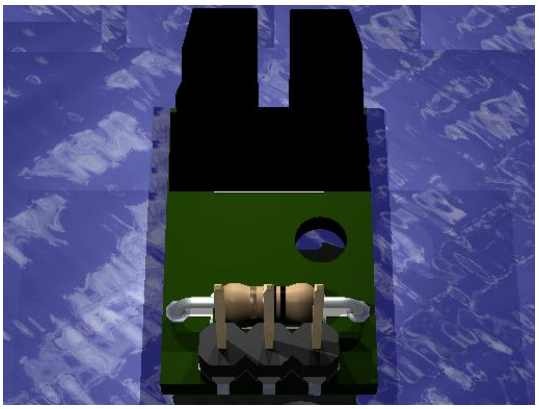


Figura 4.15 PCB de sostén de uno de los optointerruptores.

Diseñada la PCB, sumando los pesos de una tabla fenólica de iguales dimensiones con los componentes empleados, puede conocerse el peso total. Sus características más importantes son mostradas en la tabla 4.15.

Tabla 4.15 Características de la PCB principal.

Característica	Valor
Peso	32 g
Dimensiones	32 mm x 76 mm

El cableado, parte del circuito de distribución, es expuesto al obtenerse el diseño final del soporte mecánico.

4.4.3 Diseño del soporte mecánico

- 4.4.3.1 Configuración de la plataforma principal

El componente de mayores dimensiones es el PIC32 I/O Expansion Board. La plataforma principal es quien debe contenerlo: las medidas de ambos serán muy aproximadas.

De acuerdo a la configuración seleccionada para los componentes principales, el Expansion Board está colocado perpendicularmente en la parte superior de la plataforma. Esto facilita a que, de ser necesario, puedan conectársele más componentes o conexiones para futuras modificaciones. Por otra parte, se diseña de tal forma que esta tarjeta expansión pueda ser desmontada fácilmente de la aplicación para emplearse en alguna otra.

Se tiene que elegir además la manera en la que se colocarán los motores. Una configuración idónea sugiere que se posicionen ocupando el menor espacio posible y que las ruedas puedan estar en contacto con el terreno para desplazarse.

La configuración elegida para posicionar a los motores es mostrada en la figura 4.16. Los motores deben tener alineados sus ejes de salida. Puesto que los motores deben ser colocados debajo de la plataforma principal, dicha configuración da una idea del ancho que medirá aquella.

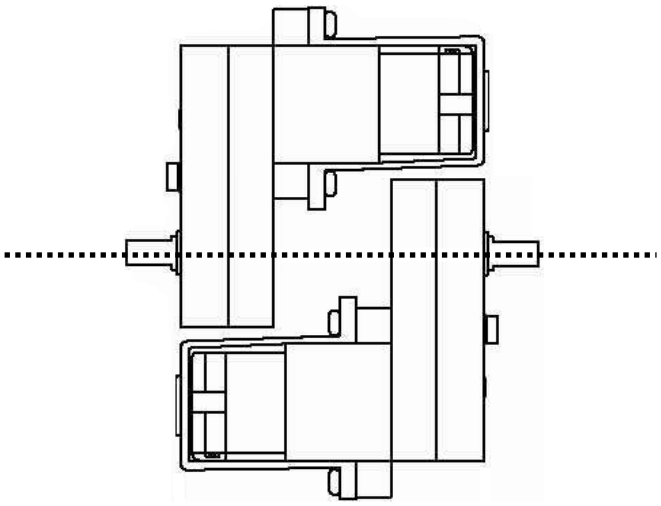


Figura 4.16 Vista superior de la configuración de la posición de los motores.

La plataforma diseñada consiste de 4 tablas unidas por tornillos, formando un rectángulo. Unas tablas más anchas son colocadas en su parte central con pegamento blanco (Resistol 850) para sostener a la PCB principal y a la batería. Estas ideas cumplen la especificación de realizar la plataforma ligera, lo que implica directamente en la utilización de menos material. El pegamento Resistol 850 es muy fácil de encontrar en el territorio mexicano, y puede soportar cargas aproximadas de más de 100 gramos (empleando las áreas de unión y el MDF). Puesto que el proyecto actual tiene estas características, se elige para realizar partes del ensamble.

Los planos de las piezas que conforman a la plataforma son mostrados en la sección 3 de los apéndices del presente trabajo, y en la figura 4.17 se muestra una parte del ensamble de ésta.

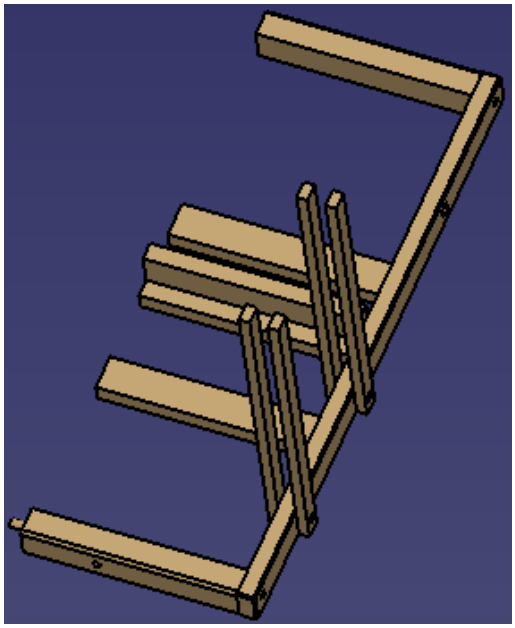


Figura 4.17 Ensamble de la parte izquierda del vehículo.

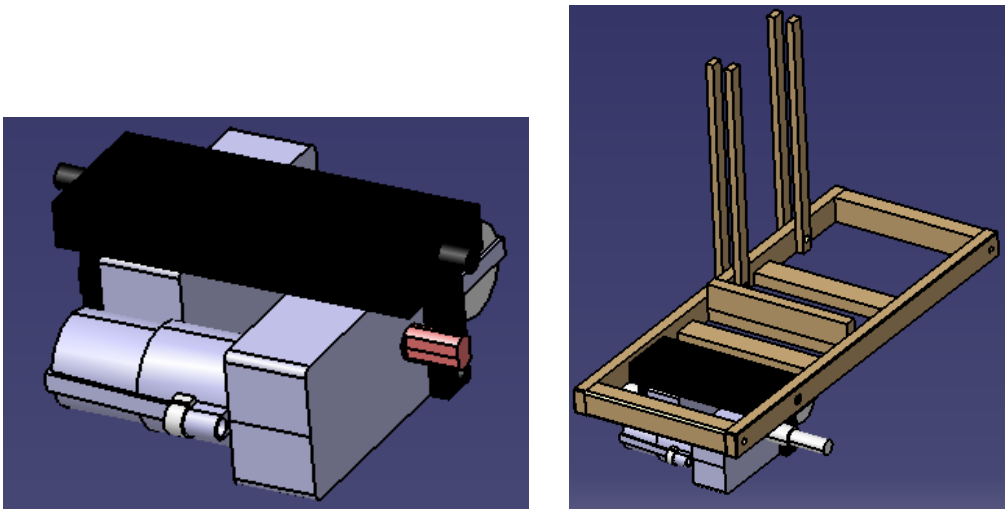
- **4.4.3.2 Configuración del mecanismo de traslación**

Tres elementos mecánicos serán construidos en un lugar especializado en corte de acrílico y otro especializado en el manejo de torno CNC con materiales plásticos. Uno de ellos es la tabla que logrará la alineación de los ejes de ambos motores, los otros dos son los acoples del eje de salida de los motorreductores a las ruedas, que se sugiere que sean realizados con PVC. Los planos de estas piezas son mostradas en el apéndice 3.4.

Se deben colocar los componentes del encoder, de tal forma que físicamente no se obstruyan entre ellos durante los giros de los motores.

Es importante, para el firmware de la aplicación, que los valores de las posiciones angulares (con respecto a uno sólo de los ejes de salida de los motores) de las ranuras de cada disco del encoder *sean lo más parecidos posible*, con el fin que se detecten simultáneamente las ranuras en ambos encoder.

La forma en que se acomodaron el disco, el sensor del encoder, los acoples de los ejes y las ruedas son mostrados en las figuras 4.18, 4.19, 4.20 y 4.21.



(a) (b)

Figura 4.18 Tabla de alineación de los motorreductores (a) y su ensamble a la plataforma principal (b).

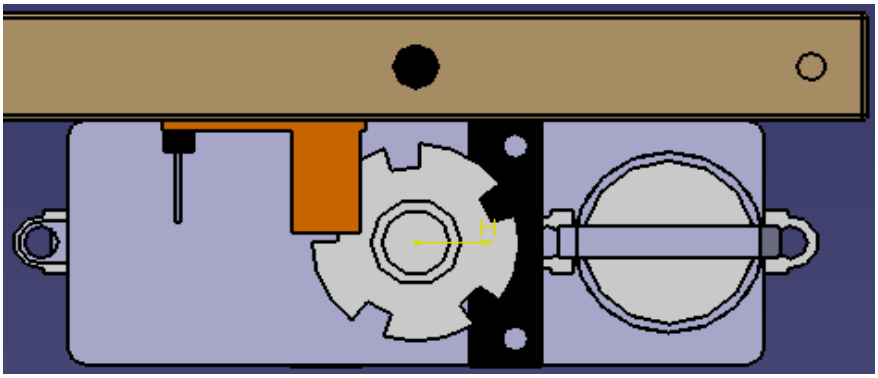


Figura 4.19 El disco del encoder ensamblado al acople del eje de la rueda.

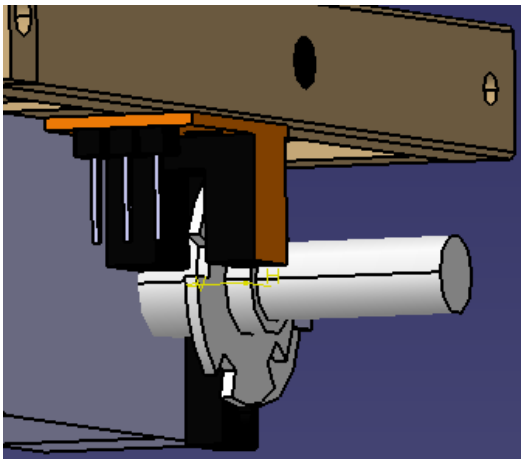


Figura 4.20 El disco del encoder sin rozamiento durante los giros del motorreductor.

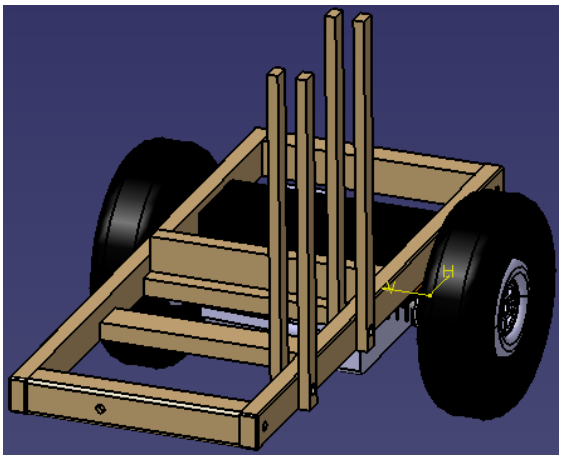


Figura 4.21 Ruedas colocadas a presión en los acoples de los ejes.

Para lograr equilibrio durante la traslación del vehículo se coloca una rueda loca en la parte frontal del vehículo. Un soporte fue diseñado para lograr que la rueda loca se posicione a la misma altura que las ruedas y la plataforma principal se mantenga en posición horizontal, paralelo al terreno en el que interactuará. El plano del soporte es mostrado en el apéndice 3.4 y su ensamble se muestra en la figura 4.22.

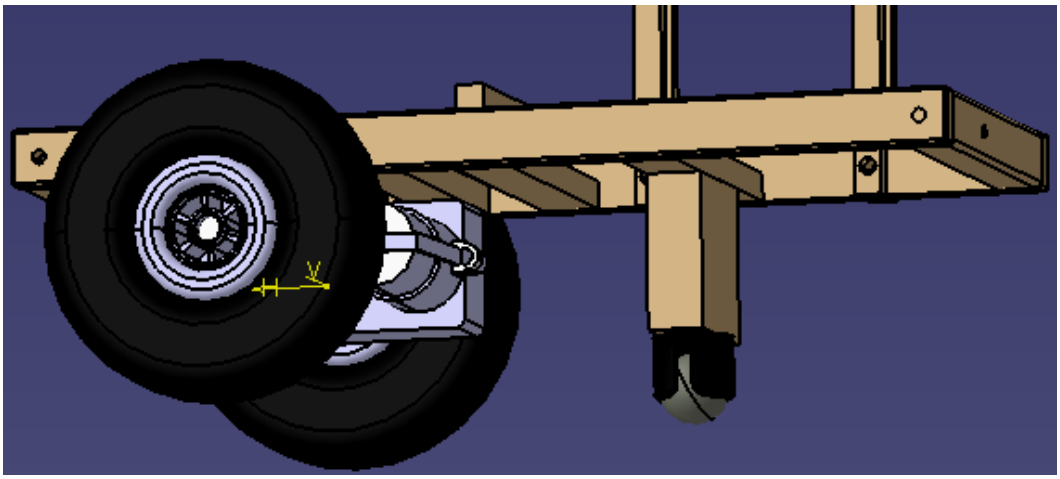


Figura 4.22 Ensamble de la rueda loca y su soporte.

El peso del soporte puede ser estimado con las densidades y el volumen.

Tabla 4.16 Determinación del peso total del soporte.

No. de componentes	Nombre	Código en apéndice	Densidad [kg/m^3]	Volumen [m^3]	Masa [kg]
1	Tabla frontal	3.1	780	4.80E-06	3.74E-03
1	Tabla izquierda	3.2	780	8.55E-06	6.67E-03
1	Tabla derecha	3.3	780	8.55E-06	6.67E-03
1	Tabla posterior	3.4	780	4.80E-06	3.74E-03
4	Tabla soporte superior	3.5	780	1.14E-05	8.89E-03
4	Tabla de unión	3.6	780	1.20E-05	9.36E-03
1	Soporte rueda loca	3.7	780	6.91E-06	5.39E-03
1	Tabla unión de motores	3.8	1190	1.10E-05	1.31E-02
2	Acoples eje-ruedas	3.9	1400	1.48E-06	2.07E-03
1	Rueda loca	-	-	-	0.06
2	Ruedas (llantas)	-	-	-	0.05
Total				**6.95E-05**	**0.1696621**

Con las masas de todos los componentes puede obtenerse el torque necesario sustituyendo todas las masas en la expresión (12).

$$\tau = 18.75 \times 10^{-3}\,[Nm] + 133.93 \times 10^{-3}(0.026 + 0.085 + 0.032 \times 2 + 0.169)\,[Nm]$$

$$\tau = 64.96 \times 10^{-3}\,[Nm] \qquad \textbf{(15)}$$

El par motor obtenido en (13) es menor al supuesto para asignar un valor a la corriente consumida por los motores (70 x 10^{-3} N·m). Se recuerda que incluso se asumió un coeficiente de fricción con el terreno excesivo, una aceleración inicial del vehículo mayor de la necesaria y magnitudes de las masas de sus componentes redondeadas a otras mayores. De esta forma, no es necesario iterar nuevamente el proceso del diseño del detalle electrónico-mecánico de la aplicación, por lo que los pasos expuestos durante el presente capítulo son los definitivos.

La batería es detenida por un cincho que puede ser sujetado por tornillos insertados en las perforaciones superiores de la tabla de alineación de los motorreductores; y también es detenida por una tabla colocada en sentido vertical, con el fin de que no se desplace hacia la parte frontal del vehículo y pueda hacer contacto con la PCB principal.

El ensamble completo del vehículo es mostrado en la figura 4.23, y la tabla 4.17 indica los números que identifican a los componentes principales.

Tabla 4.17 Componentes principales del ensamble.

Componente	Número identificador
Batería	1
Motorreductores	2
PCB principal	3
PIC32 I/O Expansion Board	4
PIC32 USB Starter Kit II	5
Plataforma principal	6
Rueda loca	7
Sensor de proximidad	8
Switch ON/OFF	9

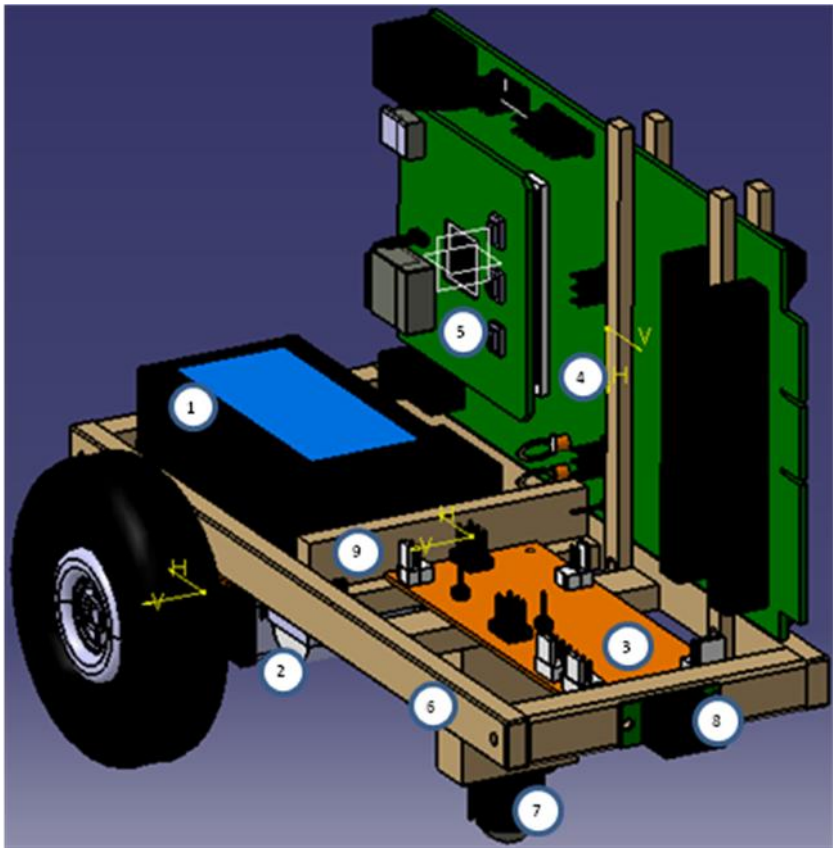


Figura 4.23 Ensamble completo del vehículo.

4.4.4 Cableado entre componentes físicos

La tabla 4.18 muestra los cables a elaborarse para lograr las conexiones electrónicas entre componentes y en la figura 4.26 se ilustra cómo deben ser conectados. Los hilos de un mismo cable son mostrados de un sólo color, pero son independientes entre sí.

Tabla 4.18 Cables entre componentes referidos en la figura 4.26.

Código color de cable	Descripción	Número de hilos	Longitud sugerida [mm]	Conectores en extremos
Naranja	Conecta el I/O Expansion Board con el driver de los motorreductores, controlando su estado activo y su dirección.	6	50	Header hembra 2 x 3 vías (ambos extremos)
Azul	Energiza desde la PCB diseñada principal a la PCB que contiene al optointerruptor que observa la posición angular de la rueda derecha. Devuelve el cambio de voltaje detectado por el sensor para ser acondicionado.	3	120	Molex hembra 3 vías – header hembra 3 vías
Morado	Energiza desde la PCB diseñada principal a la PCB que contiene al optointerruptor que observa la posición angular de la rueda izquierda. Devuelve el cambio de voltaje detectado por el sensor para ser acondicionado.	3	120	Molex hembra 3 vías – header hembra 3 vías
Verde	Energiza a la PCB con un voltaje de 7 V desde el I/O Expansion Board, y entrega la señal acondicionada de los sensores H21A1 a éste.	5	95	Header hembra 3 x 2 vías (ambos extremos sin conectar vía no. 6)
Rojo	Energiza desde la PCB diseñada principal a la PCB que contiene al sensor de objetos GP2Y0D810Z0F. Devuelve el cambio de voltaje detectado por el sensor para ser acondicionado.	3	60	Header hembra 2 x 3 vías (ambos extremos)
Café	Son dos cables de un solo hilo que proporcionan el voltaje de referencia (GND), desde el I/O Expansion Board a la PCB diseñada principal.	1	90 (c/u)	Header hembra 1 vía (ambos extremos)
Celeste	Cable de un solo hilo que proporcionan 5 V, desde el I/O Expansion Board a la PCB diseñada principal.	1	80	Header hembra 1 vía (ambos extremos)
Rojo - negro	Dos cables de dos hilos cada uno, que proporcionan la corriente y voltaje necesarios para que los motorreductores realicen sus giros.	2	120 (c/u)	Header hembra 2 vías – Molex hembra 2 vías

El último cable diseñado es el de alimentación principal. Este cable se encarga de alimentar desde la batería al I/O Expansion Board. De acuerdo a las especificaciones del presente proyecto, este cable conecta un switch que permite apagar completamente al vehículo.

El cable debe ser de dos hilos, dividido en dos partes desiguales. La primer parte (10 cm) tiene como extremo a un header de dos pines rectos, para aprovechar el molex hembra que tiene ya incluido la batería. El otro extremo es conectado al switch que es mostrado en la figura 4.4.16. Este switch puede ser aleatorio, pero es necesario que disponga de un pequeño soporte con en que atornillarlo o clavarlo a las tablas, como se muestra en la figura 4.4.17. Para el presente trabajo se empleó un mini switch de medidas 20 mm x 5 mm x 7.5 mm adquirido en la tienda de electrónica "El Primer Recurso", ubicado en la calle República del Salvador, Col. Centro, en el Distrito Federal, México.

El switch cierra o abre el flujo de la corriente del cable de alimentación. El switch alimenta al Expansion Board, por lo que se debe hacerle una extensión de 20 cm (la segunda parte del cable mencionado en el párrafo anterior) cuyo otro extremo tendrá un conector tipo plug macho de 5 mm de diámetro, que es el que requiere esta tarjeta de extensión para ser energizada. Este plug fue igualmente adquirido en la tienda Steren.

Las medidas del switch determinan la distancia entre algunas tablas de la plataforma, como se aprecia en la figura 4.24.

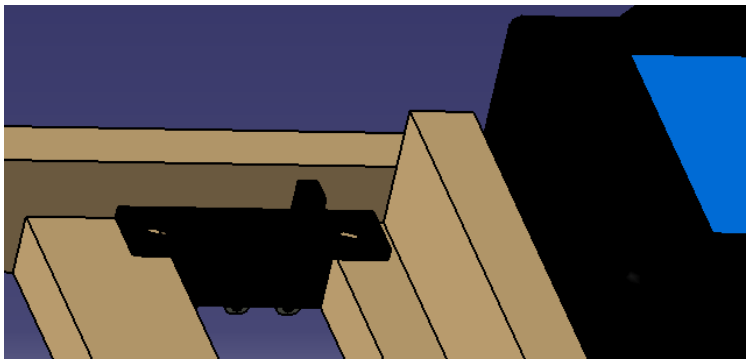


Figura 4.24 El switch principal.

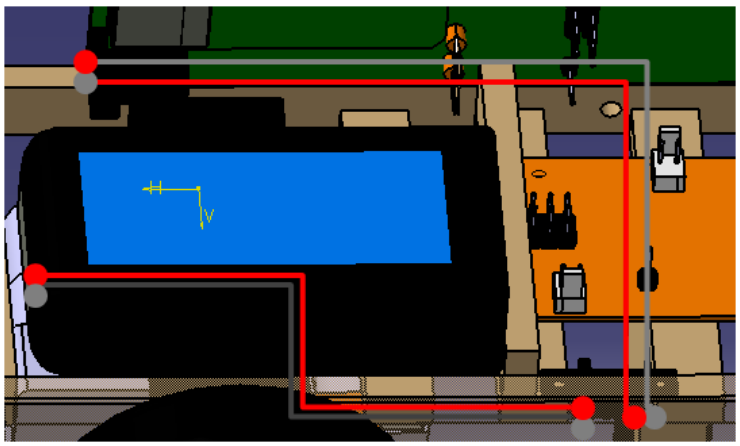


Figura 4.25 Conexión del switch principal entre la batería y el PIC32 I/O Expansion Board.

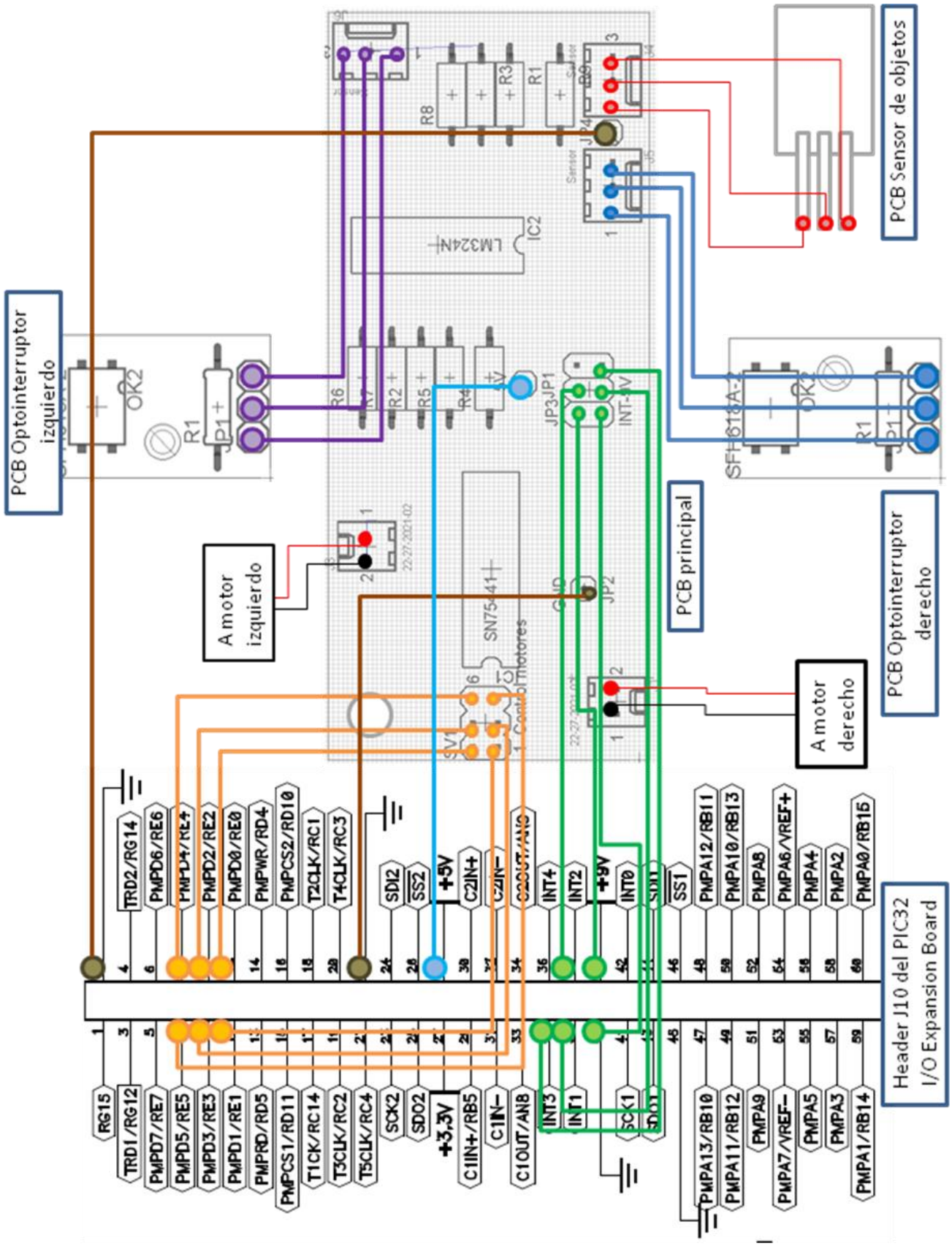


Figura 4.26 Cables para las conexiones electrónicas empleados.

4.4.5 Diseño del firmware

En los capítulos anteriores, se ha mostrado la manera en la que se pretendió entablar la comunicación por 802.11 del vehículo. Durante el presente subcapítulo se muestra la forma en la que sería posible manipular la información inalámbrica para el proceso mecatrónico de la aplicación.

En esencia, el firmware final desarrollado es representado en el diagrama de flujo de las figuras 4.27 y 4.28. Durante el resto del subcapítulo se aclaran con mayor detalle las subrutinas y procesos del programa.

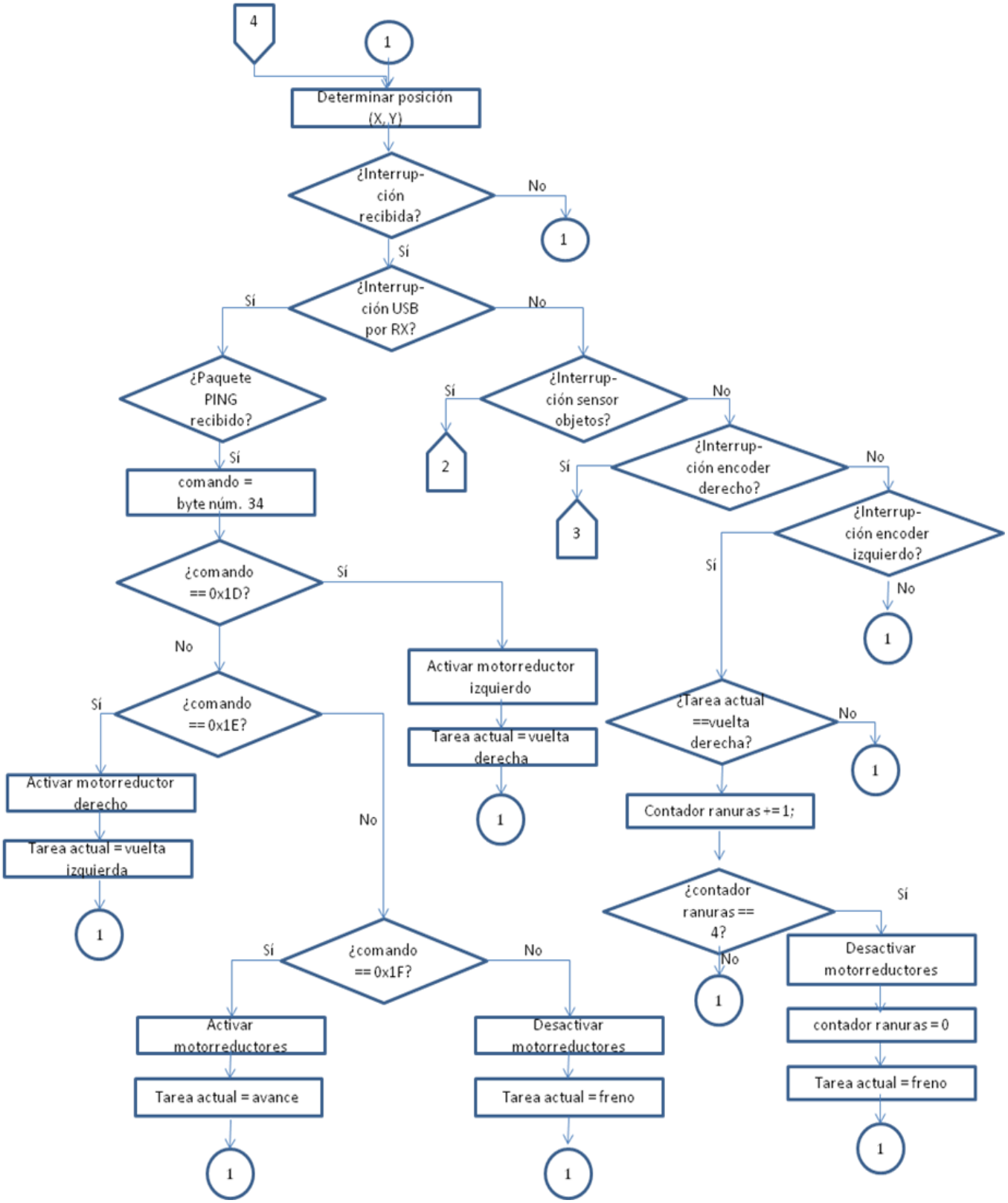


Figura 4.27 Diagrama de flujo del firmware, subproceso del programa principal.

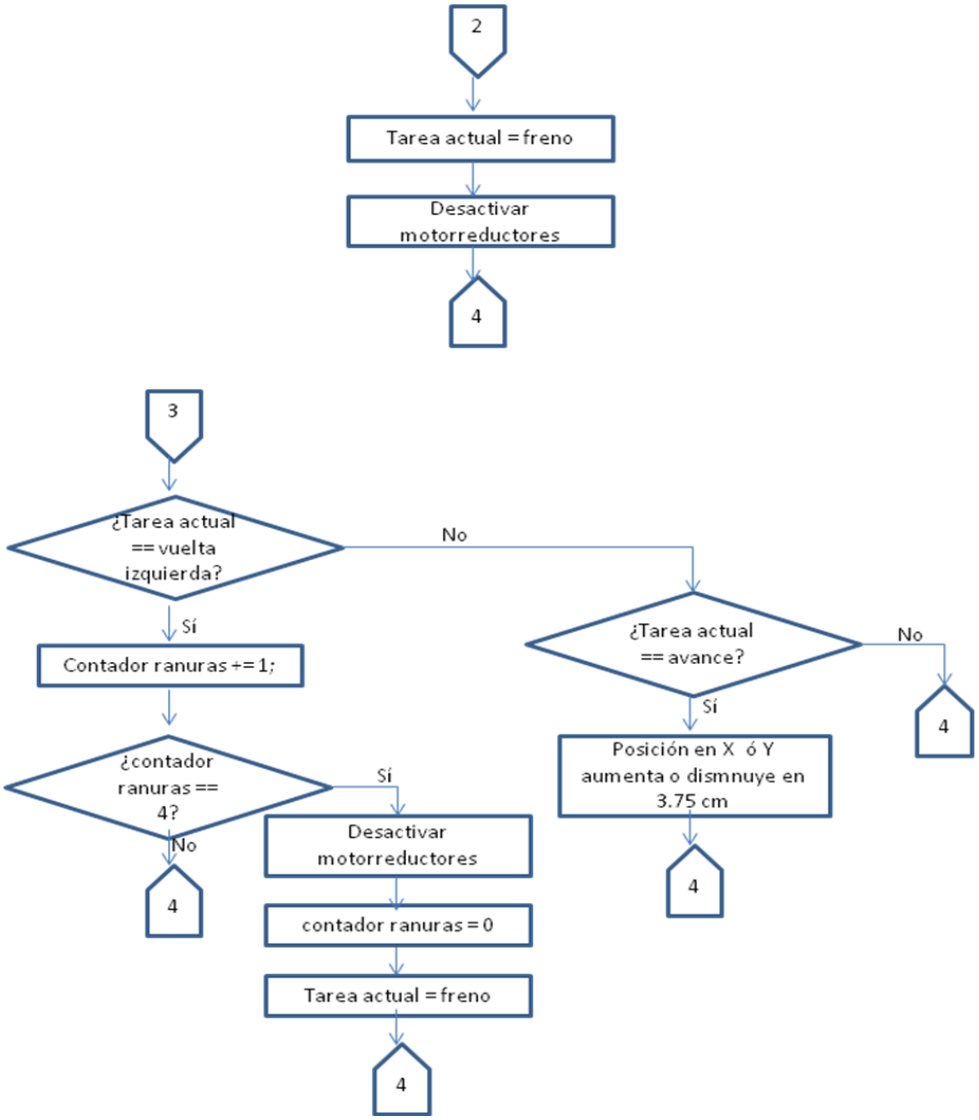


Figura 4.28 Continuación del diagrama de flujo de mostrado en la figura 4.27.

- **4.4.5.1 Traslación del vehículo**

El PIC32MX795F512L tiene diversos pines de salida y entrada digital, y cada uno tiene diversos nombres. El circuito de potencia diseñado requiere 6 salidas digitales. Se nombra al motorreductor izquierdo como motorreductor 1, al de la derecha motorreductor 2, con el fin de comprender la convención de la tabla 4.19.

Tabla 4.19 Pines empleados para la traslación del vehículo.

Propósito	Pin del MCU	I/O Expansion Board
Dirección horaria del motorreductor 1	RE4	PMD1
Dirección antihoraria del motorreductor 1	RE2	PMD2
Habilitar motorreductor 1	RE0	PMD4
Dirección horaria del motorreductor 2	RE1	PMD3
Dirección antihoraria del motorreductor 2	RE3	PMD5
Habilitar motorreductor 2	RE5	PMD0

Las secuencias que se desean obtener entonces deben ser logradas con diversas combinaciones de habilitar a los pines en unos o ceros lógicos.

Tabla 4.20 Combinaciones para la traslación del vehículo.

Movimiento	Pines en estado 1 lógico
Avance	RE1-RE2-RE0-RE5
Giro a la izquierda	RE1-RE5
Giro a la derecha	RE2-RE0
Freno	(ninguno)

En el apéndice 1.12 se muestran los códigos fuente de las rutinas para realizar los movimientos planteados.

- **4.4.5.2 Determinación de la posición espacial del vehículo**

El hecho de emplear encoder junto con los motores del vehículo es conocer la ubicación espacial del vehículo. Cuando el fototransistor recibe la luz infrarroja del fotodiodo, cerrará el circuito permitiendo pasar la señal de 5 V. Tras acondicionar esta señal, uno de los pines del MCU reconocerá este cambio de voltaje (la distancia recorrida) mientras el eje del motorreductor realiza sus revoluciones.

Los pines que se encargarán de verificar estos cambios de voltaje en ambos motorreductores son los pines RE9 y RA14, denominados INT2 e INT3 en el Expansion Board. Ambos pines actuarán en su modalidad de interrupciones externas.

Una de las maneras más fáciles de emplear las interrupciones externas es por medio de las macros que incluidos dentro del archivo `plib.h`, denominado `ConfigINTx()`, donde x es el número de la interrupción externa por emplear. Dentro de esta función se requiere especificar el número del vector de la interrupción por emplearse, así como la configuración del flanco (ascendente o descendente) para activarse. Sin embargo, requiere que se use la modalidad multivector de interrupciones y limpiar la bandera de interrupción dentro de ella. En el presente proyecto se incluye cuando a su vez, es incluido el archivo *`HardwareProfile`*`.h`.

El disco del encoder tiene 5 hendiduras o huecos. El haz del fotodido podrá captarse, entonces, cada 72°. Puesto que el radio de las ruedas empleadas es de 3 cm, se puede determinar el avance tangencial de la rueda con la expresión (15):

$$s = r\theta \tag{15}$$

Donde s es la longitud del arco de la rueda por conocer, r es el radio de la rueda y θ es el ángulo formado por el arco en radianes.

Se conocen los valores tanto del ángulo como del radio, sustituyendo en (15):

$$s = 0.03\left(\frac{72^{o}\cdot\pi}{180^{o}}\right)[m] = 0.0377\,[m]$$

Cada vez que el fototransistor permita el paso de la corriente, 5 V serán enviados al circuito de acondicionamiento, y posteriormente al MCU. Por lo tanto, cada cambio de voltaje en flanco ascendente, el firmware del MCU determinará que una rueda ha avanzado 3.77 cm.

En el caso del avance, no es necesario registrar los cambios de cada fototransistor (sólo bastará uno para determinarse la distancia recorrida).

Cabe destacar que el vehículo determinará *si seguir avanzando* o *imponer su condición de regresar*. Aunque los router con 802.11x determinan que un radio de trabajo garantizado es de 100 m, el presente proyecto limitará esta zona a un cuadrado de 9 m de lado, cuyo centro es el AP. El vehículo determinará la posición espacial en la que se encuentra con un plano cartesiano, y en caso de que esté por sobrepasar 9 m en eje X o eje Y, se frenará y solamente responderá ante comandos de cambio de dirección.

En estos casos (cambios de dirección) sí es necesario obtenerse los cambios de voltaje generados por cada fototransistor para reconocer cuántos grados el vehículo ha girado hacia la derecha o a la izquierda.

Mientras el vehículo realiza el cambio de dirección, una de las ruedas permanece estática, la otra continúa girando. Para realizar una vuelta hacia la izquierda o hacia la derecha, la rueda activa debe describir la trayectoria correspondiente a 90°. La figura 4.29 lo muestra de manera gráfica.

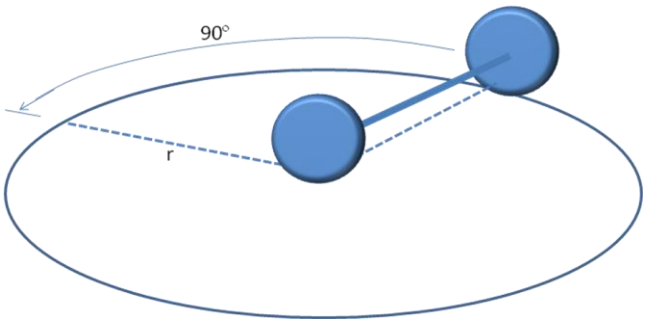


Figura 4.29 Comportamiento de las llantas del vehículo en un giro a la izquierda.

La distancia entre los centros de las ruedas es de 96 mm. Una de las ruedas se mantiene fija en su posición mientras la otra continúa realizando revoluciones. El ángulo de giro del vehículo es de 90° hacia la dirección izquierda o derecha (depende de la rueda que permanezca inmóvil).

Se requiere saber entonces, cuántas veces el PIC detectará los cambios de voltaje en flanco ascendente en el sensor del encoder. Para ello se determinará cuántas veces la longitud de un arco de la rueda definido por el ángulo entre ranuras del disco del encoder *cabe* en la longitud del arco descrito por el vehículo para cambiar de dirección. Matemáticamente puede expresarse como:

$$s_v = n\, s_r \tag{16}$$

Donde s_v es la longitud del arco descrito por el vehículo para girar, s_r es la longitud del arco entre las ranuras del encoder, y n es el número de ranuras que el MCU debe detectar para saber que ha realizado el cambio de dirección.

Sustituyendo los valores conocidos en la expresión (16), y resolviendo para n se tiene entonces que:

$$(90^\circ)\left(\frac{\pi}{180^\circ}\right)(0.096)[m] = n(72^\circ)\left(\frac{\pi}{180^\circ}\right)(0.03)$$

$$n = 4$$

De esta forma se sabe que, al realizar una rutina de giro a la izquierda o a la derecha, se deben detectar 4 ranuras (4 cambios ascendentes de voltaje por parte del sensor en herradura) para saber que se ha realizado correctamente el giro de 90°.

Estos cambios de dirección deben ser almacenados en la memoria del PIC, para conocer la posición en el que se encuentra actualmente el vehículo en caso de que a continuación avance.

Las rutinas de giros a la izquierda, derecha y avance son mostrados en el apéndice 1.12.

- **4.4.5.3 Detección de objetos frontales**

Se empleará otra interrupción externa para controlar al sensor Sharp. Este será manipulado por el pin RE8 del PIC, conociéndose en el I/O Expansion Board como INT1. Esta interrupción debe ser configurada con mayor prioridad que las demás, a excepción de la interrupción del módulo USB OTG.

En cuanto suceda un flanco en ascenso de voltaje en el sensor, será detectado por este pin del MCU. La interrupción generada hará que automáticamente ambos motorreductores sean inhabilitados.

El código que describe este comportamiento es mostrado en el apéndice 1.12, donde se descifra el comportamiento del comando PING detectado en los paquetes Data recibidos por el módulo USB OTG del PIC.

- **4.4.5.3 Interpretación de los comandos recibidos**

En el capítulo anterior se describió en detalle la composición en bytes de un comando PING junto con su cabecera estándar. Se programará al MCU para que interprete al valor hexadecimal del byte número 34 como una instrucción específica, mostradas en la tabla 4.21.

Tabla 4.21 Interpretación del valor del byte número 34 para los comandos.

Hexadecimal	Decimal	Comando a realizar
0x1D	29	Giro a la derecha
0x1E	30	Giro a la izquierda
0x1F	31	Avance
0x20	32	Freno total

La rutina de la adquisición e interpretación del comando por el módulo USB OTG es mostrado en el apéndice 1.12.

La ISR inmersa en el archivo `usb_host.c`, `_USB1Interrupt`, tiene una prioridad de 14, que es la máxima aceptada por MPLAB en MCU de 32 bits. Así se garantiza que un PING por 802.11 recibido por la tarjeta ENUWI G2, posteriormente reenviada por USB al módulo USB OTG del PIC, será tratado inmediatamente, por encima de las interrupciones de los sensores.

- **4.4.6 Interfaz usuario-aplicación**

Tanto GNU/Linux como Windows, los SO de las PC que usa el grupo de trabajo Valtroniks, cuentan con un modo texto de interfaz con el usuario. Para GNU/Linux es llamada *terminal*, en Windows es llamado *símbolo de sistema*. Se ha pensado en emplear ambas herramientas

para no tener que instalar o programar software adicional y para que el vehículo pueda emplearse en plataformas diferentes.

El comando PING en Windows, para manejar al vehículo, se ejecuta en el símbolo de sistema por medio de la siguiente sintaxis:

ping [dirección IP del vehículo] *–l* [comando]

Para ambientes UNIX, el comando PING necesita la siguiente sintaxis en terminal:

ping [dirección IP del vehículo] *–s* [comando]

En la figura 4.30 se visualiza cómo enviar el comando de giro a la izquierda al vehículo que tendría la dirección IP 192.168.1.65.

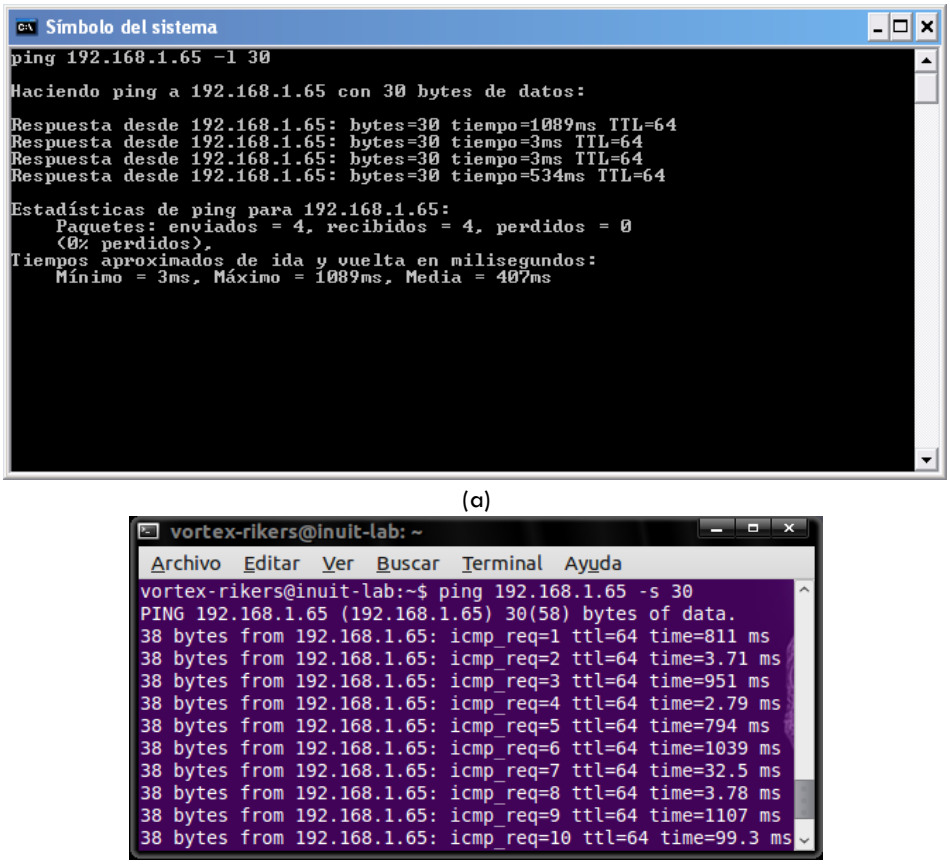


Figura 4.30 Símbolo de sistema (a) en Windows y terminal (b) en GNU/Linux (distro Ubuntu 10.10)

El MCU respondería ante dicho comando PING con la respuesta replicada de una PC (mandando el eco nuevamente por USB). La tarjeta de red recibiría esta instrucción, la retransmitiría por el estándar 802.11 al AP y finalmente a la estación de trabajo. Tanto terminal o símbolo de sistema muestran dicho eco. Se garantizaría así que el comando ha sido recibido y procesado correctamente por el PIC.

4.5 Restricciones de la aplicación

No pueden rebasarse los siguientes valores en caso de agregarse componentes adicionales al vehículo:

1. Que provoque que el torque de una de las llantas del vehículo exceda 70 x 10^{-3} [N·m] .
2. Ningún componente electrónico o mecánico puede reemplazarse por algún otro que excedan los pesos propuestos para el actual.
3. La batería puede intercambiarse por otra que proporcione más de 7 V y menos de 15 V (según especificaciones del PIC32 I/O Expansion Board), con una corriente mínima de 1 A.

El suelo o área de trabajo, en conjunto con las llantas, no pueden exceder un coeficiente de fricción estática mayor de 0.45. Debe ser una superficie plana, sin pendientes ni fisuras con anchuras mayores de 5 mm.

No se puede aplicar una torsión al PIC32 I/O Expansion Board

Durante su operación no se puede transportar manualmente al vehículo. De lo contrario perdería el valor de la ubicación espacial.

Debido a sus componentes electrónicos a la intemperie, no puede usarse esta aplicación con materiales mojados o conductores de electricidad. Se recomienda su uso para interiores.

No pueden existir objetos obstructores de menos de 5 cm de altura.

De contarse con las herramientas necesarias puede alterarse el firmware, sin embargo, no se garantiza el correcto funcionamiento del mismo para los propósitos de esta aplicación.

La manera adecuada de encender / apagar al vehículo es por medio del switch principal, por lo que se recomienda no jalar o desconectar de los cables para evitar daños a los componentes.

Capítulo 5

Análisis de resultados y conclusiones

En el presente capítulo se analizan los resultados obtenidos en los capítulos anteriores, se obtiene la conclusión del resultado del proyecto y se enuncia el replanteamiento de la hipótesis del proyecto para una posible solución final.

5.1 Análisis de resultados

5.1.1 Resumen de resultados

Como se pretendió durante el capítulo 3, la comunicación entre las tarjetas de red con el PIC32 USB Starter Kit II mediante USB fue insatisfactoria en el caso de la TPLINK 722n, y se logró una mejor respuesta (pero tampoco exitosa) en el caso de la ENUWI G2.

Las capturas hechas con Wireshark fueron realizadas en diferentes contextos. Para la tarjeta TPLINK, se realizaron 5, y fueron replicadas 3 de dichas capturas. A grandes rasgos, se implementó de la misma manera que el código fuente del apéndice 1.2 a 1.4, pero cambiando los bytes en cada captura ya que Wireshark no siempre mostraba los mismos bytes.

Pese a la repetición de idénticos bytes, la tarjeta nunca respondía con bytes que tuvieran un valor diferente de 0.

Esto llevó a una primera conjetura: que el proceso USB debe tener alguna característica que rompa completamente con la configuración de la tarjeta para funcionar.

La tarjeta ENUWI G2 tiene una estructura de comunicación mucho más sencilla, puesto que el Endpoint 0 es continuamente manipulado para el funcionamiento de la tarjeta, y en caso de requerirse o enviarse información por 802.11, ENUWI G2 hacía uso de otro endpoint.

De esta manera, la tarjeta ENUWI G2 respondía con exactamente los mismos bytes que los que se aprecian en Wireshark. Sin embargo, la tarjeta no responde ante transferencias IN con otros endpoints.

5.1.2 Análisis de resultados

Encontrar la razón del permanente estado de respuestas NAK de las dos tarjetas fue un proceso largo que hizo que se efectuaran diversas ampliaciones al código fuente. Se empezó por los siguientes razonamientos a partir de los resultados expuestos, para dar una respuesta a la pregunta de investigación:

1. Las tarjetas de red requieren un ciclo de intercambio de información muy específico. Un retardo en la respuesta hará que la tarjeta de red se “desactive” considerando que se encuentra en un puerto USB corrupto.

2. El PIC32 no necesariamente realiza las transferencias en cada frame. Esto implica que durante muchos frame, existe la posibilidad de que la tarjeta no reciba información.

3. Las capturas pueden ser *verdaderas* para un cierto momento, pero *es probable* que las tarjetas no siempre requieran estos primeros bytes de información. Es decir, si la tarjeta comenzó (observando en una captura) con una secuencia de bytes, puede

que cuente con un mecanismo propio que impida que las siguientes comiencen de la misma manera.

Una gran desventaja son las nulas especificaciones técnicas que el fabricante da sobre el funcionamiento interno de la tarjeta.

5.1.3 Verificación del análisis

Todos los búfer observados de cada captura no son completamente iguales. Sin embargo la tercera respuesta de los análisis no puede ser cierta del todo. Una tarjeta no puede contener un sistema de protección exclusiva para cada una de las computadoras a las que ha sido conectada.

De algunas modificaciones al código fuente, se observa que tramas sin emplearse no afectan la respuesta de parte del dispositivo. En Wireshark, y además empleando otros capturadores como SnoopyPro y USBlyzer, se observan tramas que no son empleadas para mandar paquetes, especialmente en la tarjeta TPLINK. La segunda respuesta queda inválida.

Queda solamente la primera de ellas.

5.2 Conclusiones

Se concluye entonces que existe algún sello referente al tiempo en las tarjetas y esto puede deberse al empleo del protocolo 802.11. La tarjeta cuenta con un método de sincronización que es muy específico, en el que intervienen temporizadores y otros ISR de una PC, debido a las frecuencias que se emplean en las versiones de 802.11. Con el PIC32 USB Starter Kit II no es posible tener un dominio total del puerto USB, ya que no se pueden controlar los tiempos de envío de información, o del tiempo de espera de las respuestas de parte de los dispositivos. Es por esto que Microchip ha desarrollado controladores para clases que cuentan con parámetros adicionales a su clase USB: tanto los HID, teclados, impresoras y dispositivos de almacenamiento masivo emplean principalmente endpoints de tipo Bulk, en los que no se les da prioridad a los tiempos de las transferencias. En realidad, los ejemplos en el MAL USB solamente emplean el controlador genérico con los mismos dispositivos creados por Microchip.

El proyecto no tuvo el resultado final deseado, pero durante su desarrollo se cumplieron varios de los sub-objetivos:

- Se logró hacer el diseño de la aplicación que servirá para realizar pruebas de otras hipótesis de acuerdo a las especificaciones establecidas.

- Se desarrolló el firmware para el control de la aplicación con lo que se alcanza la meta de manipular las herramientas al emplear al PIC32 USB Starter Kit II con interfaz USB y las tramas del estándar 802.11 para los proyectos a futuro.
- Se logró una colaboración íntegra en el que los campos de mecatrónica y computación se conjuntaron para desarrollar el proyecto y se analizaron soluciones desde ambas perspectivas.

5.3 Trabajo a futuro

La hipótesis en realidad fue una manera de comprobar que a una misma secuencia de comandos o bytes enviados por USB se reciben los mismos que lo fueron anteriormente.

Con la hipótesis descartada, se tiene que emplear la función que origina *realmente* a los paquetes Data en USB. La propuesta de este proyecto fue interesante al grupo de trabajo Valtroniks puesto que uno de sus principales objetivos a futuro es implementar módulos de código abierto en microcontroladores, con el fin de emplearlos en proyectos de mucha mayor complejidad: para la tarjeta de red TPLINK 772n, se piensa en emplear el módulo ATH5k dentro del MCU.

Este módulo del kernel Linux tiene la característica de tener un soporte oficial de parte de diversas organizaciones de software libre.

La idea es retomar el presente proyecto, y nuevamente realizar las pruebas con los mismos medios de captura y programación empleados durante el desarrollo de este: un acercamiento a la finalidad de emplear microcontroladores de 32 bits. Esto no se realizó desde un principio debido a que se requiere de la colaboración de más recursos humanos involucrados en este proyecto interdisciplinario, y de un periodo más largo de tiempo.

El proyecto, como es de suponerse, tendrá la capacidad de emplear múltiples dispositivos USB ya existentes, y en virtud de las características expuestas, a alguna de las versiones del estándar 802.11.

Apéndices

Sección 1. Software y códigos fuente

1.1 Plantilla de preparación de bits de configuración y la inclusión de los archivos para el manejo del USB Host Stack (archivo *main.c*)

```
#include <stdlib.h>
#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "usb_config.h"
#include "USB/usb.h"
#include "configuracion.h"
#include "USB/usb_host_generic.h"



#ifndef OVERRIDE_CONFIG_BITS

    #pragma config UPLLEN   = ON            // USB PLL Enabled
    #pragma config FPLLMUL  = MUL_20        // PLL Multiplier
    #pragma config UPLLIDIV = DIV_2         // USB PLL Input Divider
    #pragma config FPLLIDIV = DIV_2         // PLL Input Divider
    #pragma config FPLLODIV = DIV_1         // PLL Output Divider
    #pragma config FPBDIV   = DIV_1         // Peripheral Clock divisor
    #pragma config FWDTEN   = OFF           // Watchdog Timer
    #pragma config WDTPS    = PS1           // Watchdog Timer Postscale
    #pragma config FCKSM    = CSDCMD        // Clock Switching & Fail Safe Clock
Monitor
    #pragma config OSCIOFNC = OFF           // CLKO Enable
    #pragma config POSCMOD  = HS            // Primary Oscillator
    #pragma config IESO     = OFF           // Internal/External Switch-over
    #pragma config FSOSCEN  = OFF           // Secondary Oscillator Enable
    #pragma config FNOSC    = PRIPLL        // Oscillator Selection
    #pragma config CP       = OFF           // Code Protect
    #pragma config BWP      = OFF           // Boot Flash Write Protect
    #pragma config PWP      = OFF           // Program Flash Write Protect
    #pragma config ICESEL   = ICS_PGx2      // ICE/ICD Comm Channel Select
    #pragma config DEBUG    = OFF           // Debugger Disabled for Starter Kit

#endif // OVERRIDE_CONFIG_BITS
```

1.2 Variables y funciones de inicialización dentro de la aplicación (*main.c*) para la tarjeta de red TPLINK 722n

```
.
.
.
//_________________variables globales_____________
BOOL conectado=FALSE; //variable para determinar que el dispositivo es conectado

BYTE DIR=0x0;                //DIRECCION que se asigna al dispositivo durante la
enumeración

BYTE infoIN[50];             //un vector donde se guardan 18 bytes para la recepción por el
                             //Endpoint 0x83 (según capturas con Wireshark)

BOOL send=FALSE, read=FALSE, hacer=TRUE;

                             //SEND sirve para determinar que se inicializó una escritura
                             //correctamente (OUT). READ hace lo mismo para una lectura
                             //(IN) y HACER sirve para determinar que las dos cosas han
                             //sucedido

unsigned int lectura=1;      //De las capturas se observa que el host casi siempre
                             //hace lo mismo, cíclicamente:
                             //Host - tarjeta.Endpoint 0x83 Interrupt IN
                             //Host - tarjeta.Endpoint 0x04 Bulk OUT

                             //la variable lectura sirve para determinar en qué lugar de
                             //este ciclo se encuentra, y así saber cuál es la información
                             //que se debe mandar a continuación, al menos, en las primeras
                             //45 veces.
                             //Luego de las primeras 45 veces, cada Bulk OUT parece
                             //estabilizarse y sigue un pequeño algoritmo, y hay relación
                             //entre un Bulk OUT con el siguiente.

int main(void)

{

    //_________________variables locales_____________


       int value;     // ajustes de reloj tiempor real

       unsigned int i=0;                  //contador auxiliar

       BYTE paso=0;                       //Define cuál trama de asociación (paso+1)
                                          //802.11 se debe mandar

       BOOL sumaA11=TRUE;                 //El byte 12 por mandar siempre aumenta en 1 en
                                          //cada Bulk OUT, salvo en determinadas
                                          //ocasiones. Para saber si aumentarlo o no,
                                          //esta variable lo dirá

       BOOL configurando=TRUE;            //El evento USB_GENERIC_ATTACH inicia otro
                                          //bucle. En primer instancia hay que saber //si
                                          //"configurar" o no al dispositivo.
                                          //"Configurarlo" ya no es enumerarlo de
                                          //nuevo, sino mandar las 14 transferencias al
                                          //Endpoint 0 con 4096 bytes
```

```
        long mas4;                          //variable que sirven para guardar relación
                                            //entre los Bulk OUT al Endpoint //0x04

        BYTE infoOUT[92];                   //vector donde se almacena la información a
                                            //enviar al Endpoint 0x04



        BYTE OUTtamanio=18;                 //El número de bytes a enviar con infoOUT al
                                            //Endpoint 0x04

        BYTE *asociacion=(BYTE *)malloc(sizeof(BYTE));      //matriz dinámica con el que
                                            //mandar información al Endpoint 0x01, que esla
                                            //información a enviar al Access Point

        BYTE *respuesta=(BYTE *)malloc(sizeof(BYTE));       //búfer donde almacenar la
                                            //información o bytes que provenga desde el
                                            //Access Point



        for(i=0;i<18;i++) infoOUT[i]=0;

//_____________Configuración del Reloj del PIC32MX795F512L_____________

    value=SYSTEMConfigWaitStatesAndPB( GetSystemClock() );
    CheKseg0CacheOn();// Habilitar caché
    INTEnableSystemMultiVectoredInt();
    value = OSCCON;
    while (!(value & 0x00000020))
    {
        value = OSCCON;    // estabilizar
    }


       DBPRINTF("Hola desde el PIC32\n");
       conectado = FALSE;

    USBInitialize(0);        //forma de invocar a la función USBHostInit()
```

1.3 Bucle principal de la aplicación para la tarjeta TPLINK (*main*.c)

```
while(1)                                    //bucle principal

    {

        USBTasks();          //MAQUINA DE ESTADOS DEL BUS, INDISPENSABLE INVOCARLA SIEMPRE

        while(conectado)     //loop de tareas por hacer cuando se ha terminado de enumerar
                             //a la tarjeta
       {
                    if(configurando)
                    {
                           DBPRINTF("Entro a configurando\n");

                           Mandar4096();  //función para mandar los 12 * 4096 bytes

                           configurando=FALSE;   //se ha "configurado" dispositivo
                           lectura=0;
                           send=TRUE; read=TRUE;
                    }


                    if(send&&read)
                    {
                           lectura++;     //avanzar un paso en las primeras 45 sesiones
                                          //"arbitrarias"
                           hacer=TRUE;

                    }
                    else hacer=FALSE;

                    DBPRINTF("LECTURA %d\n",lectura);
                    sumaA11=TRUE;

                    /*Las primeras 45 transacciones Interrupt IN a 0x83 y Bulk OUT a 0x04
                    son muy irregulares como se vieron en las capturas. Vamos a replicar
                    los  Bulk OUT tal cual, quizá tengan también algo que ver en la
                    configuración del funcionamiento de la tarjeta.

                    //Algunos casos de estas 45 se omitieron, puesto que solamente
                    //aumentaban su byte número 12 en 1, y la variable sumaA11 no cambia


                    if(lectura<46)
                    {
                    switch(lectura)
                    {
                           case 1:
                                  for(i=0;i<18;i++) infoOUT[i]=0x0;
                                  infoOUT[3]=0x0a;infoOUT[9]=0x02;infoOUT[10]=0x01;
                                  infoOUT[14]=0x03;infoOUT[15]=0x04;

                                  sumaA11=FALSE;
                                  break;
                           break;

                           case 2:
                                  infoOUT[4]=0x05;infoOUT[6]=0x10;infoOUT[14]=0x02;
                                  infoOUT[15]=0x01;infoOUT[15]=0x20;
                           break;
```

```
                        case 3:

                        //Todos estos bytes en cada caso son tomados de las
                        //capturas realizadas con Wireshark

                        .
                        .
                        .
                        //Al llegar al caso 45 se está por finalizar los vaivenes
                        //iniciales, entonces, se prepara de una vez al paquete
                        //que se mandará al Endpoint 0x01, que corresponde al HOST- AP
                        //PROBE RESPONSE
                        //este paquete se mandará muchas veces hasta que el Access
                        //Point mande información a la tarjeta que el Host leerá
                        //por el Endpoint 0x82


                        case 45:
                               infoOUT[14]=0x21;
                               paso=1;
                               //del paquete 27881   capturado con Wireshark
                               asociacion=(BYTE*)realloc(asociacion,100*sizeof(BYTE));
                               respuesta=(BYTE *)realloc(respuesta,380*sizeof(BYTE));
                               for(i=0;i<100;i++) asociacion[i]=0;
                               for(i=0;i<380;i++) respuesta[i]=0;


asociacion[0]=0x60;asociacion[2]=0x7e;asociacion[3]=0x69; asociacion[4]=0x5;
asociacion[7]=0x58;asociacion[8]=0x1;
asociacion[17]=0xff;asociacion[20]=0x40;asociacion[24]=0xff;asociacion[25]=0xff;
                               .
                               .
                               .

                        //Todos los bytes de las tramas son tomados de las
                        //capturas realizadas con Wireshark


                        break;

                        default:
                        break;
                }//fin de switch(lectura)
                }//fin de if 45 paquetes
```

1.4 Funciones de *main*.c (continuación de Apéndice 1.3)

```
//ESTO SERÁ LO QUE CONTINÚE POR LA ETERNIDAD

                    //ALGORITMO entre cada Bulk-OUT

                    else
                    {
                           lectura=46;    //evitar que lectura llegue a desbordarse

                           //esto es el algoritmo, luego de los 45 vaivenes iniciales,
                           //cada BulkOut guardan relación

                           if(conteo%3!=0)
                           {
                                  infoOUT[14]=0x40;
                                  infoOUT[15]=0x7c;
                                  conteo++;
                           }
                           else
                           {
                                  conteo=1;
                                  if(mas4!=0xfc) mas4+=0x04;
                                  else
                                  {
                                         mas4=0;
                                         infoOUT[13]++;
                                  }
                                  infoOUT[14]=0x21;
                                  infoOUT[15]=mas4;
                           }
                    }

//___________RUTINAS CICLICAS__________
//aquí propiamente se mandan los Interrupt IN a 0x83 y Bulk OUT a 0x04

              if(hacer)
              {
                    if(sumaA11&&infoOUT[11]!=0xff) infoOUT[11]++;
                    //el byte número 12 la mayoría de las veces aumenta en 1

                    else
                    {
                           infoOUT[10]++;
                           infoOUT[11]=0;
                    }      //si llega a 0xFF aumenta en 1 el byte número 11

              //macro para determinar si la bandera rx está igualada a 1

                    if(!USBHostGenericRxIsBusy(DIR)&&lectura!=10)
                    {
                           //entonces el host sigue leyendo algo anterior
                           DBPRINTF("POR LEER\n");

                           if(USBHostGenericRead3(DIR,&infoIN,18)!=USB_SUCCESS)
                                  DBPRINTF("ALGO FALLO CON LA LECTURA\n");
                           else
                           {
                                  DBPRINTF("LECTURA CORRECTA\n");
                                  read=FALSE;
```

```
                        }
                    }
                    else DBPRINTF("SIGO OCUPADO CON LA LECTURA\n");

                    DBPRINTF("\n");
        //macro para determinar si la bandera tx está igualada a 1
                    if(!USBHostGenericTxIsBusy(DIR)&&lectura!=12)
                    {
                        //entonces el host sigue esperando el handshake de la tarjeta
                        DBPRINTF("POR ESCRIBIR\n");

                        if(USBHostGenericWrite4(DIR,&infoOUT,OUTtamanio)!=USB_SUCCESS)
                            DBPRINTF("ALGO FALLO CON LA ESCRITURA\n");
                        else
                        {
                            DBPRINTF("ESCRITURA CORRECTA\n");
                            send=FALSE;
                        }
                    }
                    else DBPRINTF("SIGO OCUPADO CON LA ESCRITURA\n");

            } //fin de if hacer

//________PAQUETES ESPECIALES PARA MANDAR O RECIBIR DEL ACCESS POINT_________

                    if(paso>0&&paso<5)
                    {

                //Ya se sobrepasaron los vaivenes iniciales, ahora se intenta también mandar
                //los paquetes para conectarse a la WLAN

                        if(!USBHostGenericTxIsBusy(DIR))
                        {
                            USBHostGenericWrite1(DIR,asociacion,sizeof(asociacion));
                            //mandar el paquete al Endpoint 0x01
                        }

                        if(!USBHostGenericRxIsBusy(DIR))
                        {
                            USBHostGenericRead2(DIR,respuesta,sizeof(respuesta));
                            //leer la información que manda la tarjeta desde el
                            Endpoint 0x82
                        }


                    switch(paso)
                    {

                            case 1:
                            if(respuesta[48]==0x80&&respuesta[49]==0x80&&
                            respuesta[50]==0x80 &&respuesta[51]== 0x80 &&
                            respuesta[52]==0x50)
                            // ha mandado el probe response!!!!
                            {
                            paso++;
                            respuesta=(BYTE *)realloc(respuesta,92*sizeof(BYTE));
                            asociacion=(BYTE *)realloc(asociacion,50*sizeof(BYTE));

                            //preparar paquete con los mismos bytes para la
                            //siguiente petición de asociación 802.11x
                            //del paquete 39701  de Wireshark: paq authentication
```

```
                                    for(i=0;i<50;i++) asociacion[i]=0;
                                    for(i=0;i<92;i++) respuesta[i]=0;

                                    asociacion[0]=0x2e; asociacion[2]=0x7e;
                                    asociacion[3]=0x69; asociacion[4]=0x5;
                                    asociacion[7]=0x26; asociacion [8]=0x1;
```

//preparar el resto de los bytes replicados en el vector o búfer Asociacion para cada trama del proceso de asociación, aumentar en 1 la variable Paso, verificar posible respuesta en el vector Respuesta con los bytes en las posiciones 51 y 52

```
                                }
                                break;

                                case 2:
.
.
.
//bytes replicados del paquete 43077
.
.
.
                                break;

                                case 3:
.
.
.
//bytes replicados del paquete 45076
                                .
                                .
                                .

                                break;
                        }//fin de switch

                        }//fin de if(paso>0)

                //Siempre revisar la máquina de estados por eventos acumulados o para avanzar
                //máquina de estados del bus

                USBTasks();

                }//fin de while(CONECTADO)
}//fin de bucle principal

return 0;
} // fin de main
```

1.5 Manejador de eventos de la aplicación (*main.c*) para ambas tarjetas

```
BOOL USB_ApplicationEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
    int i;                          //contador auxiliar
    switch( event )
    {
            case EVENT_GENERIC_ATTACH:
                    conectado=TRUE;
                    DIR=address;
                    DBPRINTF("SE HA CONECTADO LA TARJETA. SE LE ASIGNO LA DIRECCION:");
                    DBPRINTF(" %d\n",address);
                    return TRUE;
            break;

            case EVENT_GENERIC_DETACH:
                    conectado=FALSE;
                    //DBPRINTF("SE DESCONECTO UN DISPOSITIVO");
                    return TRUE;
                    break;

            case EVENT_GENERIC_RX_DONE:   //Evento RX DONE
//Este evento es cuando el PIC recibió el número de bytes especificados
                    read=TRUE;
                    //DBPRINTF("RXDONE\n");      //tarjeta un paquete de handshake

                    //leer lo que se recibió

                    for(i=0;i<50;i++) DBPRINTF("%d\n",(unsigned int)infoIN[i]);
                    DBPRINTF("\n");
                    return TRUE;
                    break;

            case EVENT_GENERIC_TX_DONE:   // Evento TX DONE
//ha mandado un paquete ACK como respuesta. De recibir un NAK, intenta de nuevo

                    send=TRUE;
                    //DBPRINTF("TXDONE\n");

                    return TRUE;
                    break;

        case EVENT_VBUS_REQUEST_POWER:
            // The data pointer points to a byte that represents the amount of power
            // requested in mA, divided by two.  If the device wants too much power,
            // we reject it.
            return TRUE;
                    break;

        case EVENT_VBUS_RELEASE_POWER:
            // Turn off Vbus power.
            //This means that the device was removed
            conectado = FALSE;
            return TRUE;
                    break;

        case EVENT_HUB_ATTACH:
            return TRUE;
                    break;

        case EVENT_UNSUPPORTED_DEVICE:
```

```
            return TRUE;
                    break;

        case EVENT_CANNOT_ENUMERATE:
            return TRUE;
                    break;

        case EVENT_CLIENT_INIT_ERROR:
            return TRUE;
                    break;

        case EVENT_UNSPECIFIED_ERROR:   // This should never be generated.
             DBPRINTF("UNSPECIFIED ERROR");
            return TRUE;
                    break;

        default:
                    return TRUE;
            break;
    }

    return FALSE;
}
```

1.6 Funciones Mandar 4096 (*main.c*) y TarjetaConfig (*configuracion.h*) para tarjeta TPLINK 722n

```
void Mandar4096()
{
	WORD tamanio=4096; //tamaño del búfer a mandar
	WORD contador_wValue = 0x5010; //wValue inicial como se obtuvieron en las capturas
	BYTE i=1;
	BYTE comando[4096]; //búfer con bytes a mandar por transferencias de control SET

	DBPRINTF("Mandando 4096\n");

	while(TRUE)
	{
			TarjetaConfig(i,comando);///40.................30.1050..0000.0010
	if(USBHostIssueDeviceRequest(gc_DevData.ID.deviceAddress, USB_SETUP_TYPE_VENDOR,
	0x30,contador_wValue,0x0000,tamanio,&comando,USB_DEVICE_REQUEST_SET,
	gc_DevData.clientDriverID )==USB_SUCCESS)
		{
			contador_wValue+=0x0010;	//como se aprecia en las capturas
			i++;

			DBPRINTF("caso no . %d \n",(unsigned int)i);
			//esta leyenda indica que se inicializó correctamente el envío del
			//búfer con los bytes

			if(i==13) tamanio=2128;
			if(i==14) break;

		}
		USBTasks();  //avanzar máquina de estados o verificar eventos USB
	}
		USBHostIssueDeviceRequest(gc_DevData.ID.deviceAddress, USB_SETUP_TYPE_VENDOR,
0x31,0x9030,0x0000,0x0,NULL, USB_DEVICE_REQUEST_SET,gc_DevData.clientDriverID );
		//el ultimo búfer mandado no contiene información

		//free(comando);

}//fin de Mandar 4096


void TarjetaConfig(BYTE opcion,BYTE configuracion[])
{
	unsigned int i=0;
	switch(opcion)
	{
		case 1:
for(i=0;i<4096;i++) configuracion[i]=0x0;

configuracion[0]=0x37;configuracion[1]=0x5c;configuracion[2]=0x36;configuracion[3]=0x5c;
configuracion[4]=0x36;configuracion[5]=0x5c;configuracion[6]=0x36;configuracion[7]=0x5c;
configuracion[8]=0x36;configuracion[9]=0x5c;configuracion[10]=0x36;configuracion[11]=0x5c;
configuracion[12]=0x36;configuracion[13]=0x5c;configuracion[14]=0x36;configuracion[15]=0x5c;
configuracion[16]=0x36;configuracion[17]=0x5c;configuracion[18]=0x36;configuracion[19]=0x5c;
configuracion[20]=0x36;configuracion[21]=0x5c;configuracion[22]=0x36;configuracion[23]=0x5c;
configuracion[24]=0x36;configuracion[25]=0x5c;configuracion[26]=0x36;configuracion[27]=0x5c;
configuracion[28]=0x36;configuracion[29]=0x5c;configuracion[30]=0x36;configuracion[31]=0x5c;
configuracion[137]=0x90;configuracion[138]=0x9e;configuracion[139]=0x9c;
configuracion[141]=0x90;configuracion[142]=0x9b;configuracion[143]=0xd4;
```

```
.
.
.
```
//se replican todos los bytes observados en las capturas
```
.
.
.

break;

        case 2:
        for(i=0;i<4096;i++)
        {
                configuracion[i]=0x0; //la segunda ronda de 4096 bytes son únicamente ceros
        }
break;

        case 3:
.
.
.
```
//se replican todos los 4096 bytes de cada paquete, y se asignan por referencia al vector Configuracion
```
}//fin de TarjetaConfig
```

1.7 Modificaciones al controlador (*usb_host_generic.c*) para tarjeta TPLINK 722n (extractos)

```
BOOL USBHostGenericEventHandler ( BYTE address, USB_EVENT event, void *data, DWORD size )
{
.
.
.
case EVENT_TRANSFER:
        if ( (data != NULL) && (size == sizeof(HOST_TRANSFER_DATA)) )
        {
            DWORD dataCount = ((HOST_TRANSFER_DATA *)data)->dataCount;

               if ( ((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x82 ||
               ((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x83))                  )
               {
                gc_DevData.flags.rxBusy = 0;
                gc_DevData.rxLength = dataCount;

                USB_HOST_APP_EVENT_HANDLER(gc_DevData.ID.deviceAddress,
                EVENT_GENERIC_RX_DONE, &dataCount, sizeof(DWORD) );
            }

            else if ( ((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x01||
                            ((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x04)
            {
                gc_DevData.flags.txBusy = 0;

                USB_HOST_APP_EVENT_HANDLER(gc_DevData.ID.deviceAddress,
                EVENT_GENERIC_TX_DONE, &dataCount, sizeof(DWORD) );
            }
.
.
.
} //fin de USBHostGenericEventHandler

//creación de la function USBHostGenericRead3

BYTE USBHostGenericRead3( BYTE deviceAddress, void *buffer, DWORD length )
{
.
.
.
    if (gc_DevData.flags.rxBusy)
       return USB_BUSY;

    // Set the busy flag, clear the count and start a new IN transfer.
    gc_DevData.flags.rxBusy = 1;
    gc_DevData.rxLength = 0;

//El endpoint ahora es fijo, en este caso, el 0x83

    RetVal = USBHostRead( deviceAddress, 0x83, (BYTE *)buffer, length );

    if (RetVal != USB_SUCCESS)
    {
        gc_DevData.flags.rxBusy = 0;    // Clear flag to allow re-try
    }
```

```
    return RetVal;

} // fin de USBHostGenericRead3

//se programa una función similar llamada USBHostGenericRead2, cambiando el endpoint fijo
//por 0x82

//creación de la function USBHostGenericWrite1

BYTE USBHostGenericWrite1( BYTE deviceAddress, void *buffer, DWORD length )
{
.
.
.
    if (gc_DevData.flags.txBusy)
       return USB_BUSY;

    // Set the busy flag, clear the count and start a new IN transfer.
    gc_DevData.flags.txBusy = 1;
    gc_DevData.txLength = 0;

//El endpoint ahora es fijo, en este caso, el 0x01

    RetVal = USBHostWrite( deviceAddress, 0x01, (BYTE *)buffer, length );

    if (RetVal != USB_SUCCESS)
    {
        gc_DevData.flags.rxBusy = 0;    // Clear flag to allow re-try
    }
    return RetVal;

} // fin de USBHostGenericWrite1

//se hace una función similar llamada USBHostGenericWrite4, cambiando el endpoint fijo por
//0x04
```

1.8 Modificaciones a la capa USB Host Stack (*usb_host*.c) para ambas tarjetas

```
USB_ENDPOINT_INFO * _USB_FindEndpoint( BYTE endpoint )
{
.
.
.
    while (pInterface)
    {
        // Look for the endpoint in the currently active setting.

        while (pEndpoint)
        {
//modificar para mostrar toda la lista de endpoint y sus tipos de transferencia USB

               pEndpoint = pInterface->pCurrentSetting->pEndpointList;
              DBPRINTF("Este endpoint %d tiene transferencias de tipo: %d",
               pEndpoint->bEndpointAddress,pEndpoint-> bmAttributes.bfTransferType);

            if (pEndpoint->bEndpointAddress == endpoint)
            {
                // We have found the endpoint.
                return pEndpoint;
            }
            pEndpoint = pEndpoint->next;
        }
.
.
.
}//fin de _USB_FindEndpoint


void _USB1Interrupt( void )
{
.
.
.
//modificar para mostrar  Handshake

       if (pBDT->STAT.PID == PID_ACK)
       {
              DBPRINTF("¡ACK!");
.
.
.
       else if (pBDT->STAT.PID == PID_NAK)
       {
              DBPRINTF("¡NAK!");
.
.
.
}//fin de _USB1Interrupt
```

1.9 Variables, funciones de inicialización y bucle principal (*main.c*) para la tarjeta de red ENUWI G2

```
BOOL conectado=FALSE;         //variable para determinar que el dispositivo está conectado
BYTE DIR=0x0;          //DIRECCION que se asigna al dispositivo durante la enumeración
BYTE infoIN[50];       //un vector donde se guardarán los bytes para la recepción por el
                       //Endpoint 0x83

int main(void)

{

    //_________________configuracion inicial_____________

       int value;                                  //sirve solamente para ajustar al reloj
       unsigned int i=0;                    //contador auxiliar

       BYTE paso=0;           //La variable paso servirá solamente cuando se empiecen a
                              //mandar paquetes al Access Point

       BYTE infoOUT[92];      //vector donde se almacena la información a enviar al Endpoint
                              //0x04



       BYTE OUTtamanio=18;    //El número de bytes a enviar con infoOUT al Endpoint 0x04

       for(i=0;i<92;i++) infoOUT[i]=0;

//_________________Configuración del Reloj del PIC32MX795F512L_____________

       value=SYSTEMConfigWaitStatesAndPB( GetSystemClock() );

    // Habilitar caché
    CheKseg0CacheOn();
    INTEnableSystemMultiVectoredInt();
    value = OSCCON;
    while (!(value & 0x00000020))
    {
        value = OSCCON;    // estabilizar
    }

//_________Mensaje de comprobación del funcionamiento (visibles sólo con MPLAB)_________

       DBPRINTF("Hola desde el PIC32\n");
       conectado = FALSE;

//______INICIALIZAR EL USB HOST LAYER_________

    USBInitialize(0);               //forma de invocar a la función USBHostInit()
//_____________________________________


//configurar desde el inicio la trama Probe Request según capturas con Wireshark

infoOUT[0]=0x34;infoOUT[1]=0x80;infoOUT[2]=0x0;infoOUT[3]=0x0;infoOUT[4]=0x0;infoOUT[5]=0x0;
infoOUT[6]=0x0;infoOUT[7]=0x0;infoOUT[8]=0x0;infoOUT[9]=0x0;infoOUT[10]=0x0;infoOUT[11]=0x0;
infoOUT[12]=0x0;infoOUT[13]=0x0;infoOUT[14]=0x8a;infoOUT[15]=0x3;

infoOUT[16]=0x0;infoOUT[17]=0x0;infoOUT[18]=0x0;infoOUT[19]=0x0;infoOUT[20]=0x0;infoOUT[21]=
0x0;infoOUT[22]=0x0;infoOUT[23]=0x0;infoOUT[24]=0x0;infoOUT[25]=0x0;infoOUT[26]=0x0;
```

```
infoOUT[27]=0x0;infoOUT[28]=0x0;infoOUT[29]=0x0;infoOUT[30]=0x0;infoOUT[31]=0x0;

.
.
.// así hasta el byte 83

    while(1)                                //loop principal
    {
        USBTasks();                 //MAQUINA DE ESTADOS DEL BUS, SIEMPRE INVOCARLA

           while(conectado)  //loop de tareas por hacer cuando se ha terminado de enumerar
                             //a la tarjeta
               {
                      PostConfiguracion();

//macro para determinar si la bandera rx está igualada a 1
                      if(!USBHostGenericRxIsBusy(DIR)&&lectura!=10)
                      {
                             if(USBHostGenericRead3(DIR,&infoIN,18)!=USB_SUCCESS)
                                    DBPRINTF("ALGO FALLO CON LA LECTURA\n");
                             else DBPRINTF("LECTURA CORRECTA\n");
                      }
                      else DBPRINTF("SIGO OCUPADO CON LA LECTURA\n");

//macro para determinar si la bandera tx está igualada a 1
                      if(!USBHostGenericTxIsBusy(DIR)&&lectura!=12)
                      {
                             if(USBHostGenericWriteC(DIR,&infoOUT,84)!=USB_SUCCESS)
                                    DBPRINTF("ALGO FALLO CON LA ESCRITURA\n");
                             else DBPRINTF("ESCRITURA CORRECTA\n");
                      }
                      else DBPRINTF("SIGO OCUPADO CON LA ESCRITURA\n");


               //Siempre revisar la máquina de estados*/

               USBTasks();
               }//fin de while(CONECTADO)
       }//fin de bucle principal
return 0;
}       //fin de main
```

1.10 Funciones de PostConfiguracion (*usb_host_generic.c*) y TarjetaConfig (*configuracion.h*) para la tarjeta de red ENUWI G2

```
BOOL rxBusy0 = FALSE;
int conteo=0;
BYTE comando[10];
.
.
.

void PostConfiguracion()
{
	BYTE GXbRequestType=0xc0,GXbRequest=0x05;
	WORD GXwValue=0xff44,GXwIndex=0x0,GXsize=4;
	BOOL GXSetGet=1;//GET es 1, set 0
	TarjetaConfig(&GXbRequestType,&GXbRequest,&GXwValue,&GXwIndex,&GXsize,
	&GXSetGet,comando,conteo);
	if(!rxBusy0)
	{
	//	DBPRINTF("conteo = %d\n",conteo);

		if(GXSetGet==1)
		{
			reading=TRUE;
		}
		rxBusy0=TRUE;
		USBHostIssueDeviceRequest(gc_DevData.ID.deviceAddress,
		GXbRequestType,GXbRequest, GXwValue,GXwIndex,GXsize,&comando,
		GXSetGet,gc_DevData.clientDriverID );
	}
}

#define GXGET 1
#define GXSET 0

void TarjetaConfig(BYTE *GXbRequestType,BYTE *GXbRequest,WORD *GXwValue,WORD *GXwIndex,WORD
*GXsize,BOOL *GXSetGet, BYTE comando[],int conteo)
{
	unsigned int i=0;
	switch(conteo)
	{
		case 0:
			*GXbRequestType=0xc0;*GXbRequest=0x05;
			*GXwValue=0xff44;*GXwIndex=0x0;*GXsize=4;
			*GXSetGet=1;//GET es 1, set 0
		break;
		case 1:
		*GXbRequestType=0x40;
		*GXwValue=0xff50;
		*GXsize=1;
		*GXSetGet=GXSET;
		comando[0]=0xc0;
		break;
		case 2:
		*GXbRequestType=0xc0;
		*GXSetGet=GXGET;
		break;
		case 3:
//hasta la transferencia 1056 se replican las transferencias de control
}
```

1.11 Modificaciones al controlador genérico (*usb_host_generic.c*) para emplear la tarjeta de red ENUWI G2

```
BOOL rxBusy0 = FALSE;
int conteo=0;
BYTE comando[10];
.
.
.

BOOL USBHostGenericEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
       BYTE i;
.
.
.

//modificados los endpoint donde sucedieron los eventos para notificar a la aplicación

if ( ((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x83)
            {
                gc_DevData.flags.rxBusy = 0;
                gc_DevData.rxLength = dataCount;

                USB_HOST_APP_EVENT_HANDLER(gc_DevData.ID.deviceAddress,
EVENT_GENERIC_RX_DONE, &dataCount, sizeof(DWORD) );
            }
            else if ( ((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x0C)
            {
                gc_DevData.flags.txBusy = 0;

                USB_HOST_APP_EVENT_HANDLER(gc_DevData.ID.deviceAddress,
EVENT_GENERIC_TX_DONE, &dataCount, sizeof(DWORD) );
            }

//agregado el evento de manejar los eventos del Endpoint 0 y leer la respuesta de la
tarjeta, para comparar que sean iguales a los de las capturas observadas

            else if(((HOST_TRANSFER_DATA *)data)->bEndpointAddress == 0x0)
            {
                for(i=0;i<8;i++) DBPRINTF("%d",comando);
                rxBusy0=FALSE;
                conteo++;
                return FALSE;
            }
.
.
.}// fin de USBHostGenericEventHandler

//creación de la function USBHostGenericRead3

BYTE USBHostGenericRead3( BYTE deviceAddress, void *buffer, DWORD length )
{
.
.
.
    if (gc_DevData.flags.rxBusy)
       return USB_BUSY;
```

```
    // Set the busy flag, clear the count and start a new IN transfer.
    gc_DevData.flags.rxBusy = 1;
    gc_DevData.rxLength = 0;

//El endpoint ahora es fijo, en este caso, el 0x83

    RetVal = USBHostRead( deviceAddress, 0x83, (BYTE *)buffer, length );

    if (RetVal != USB_SUCCESS)
    {
        gc_DevData.flags.rxBusy = 0;    // Clear flag to allow re-try
    }
    return RetVal;

} // fin de USBHostGenericRead3

//se programa una función similar llamada USBHostGenericRead9, cambiando el endpoint fijo
//por 0x89

//creación de la function USBHostGenericWriteC
BYTE USBHostGenericWrite1( BYTE deviceAddress, void *buffer, DWORD length )
{
.
.
.
    if (gc_DevData.flags.txBusy)
       return USB_BUSY;

    // Set the busy flag, clear the count and start a new IN transfer.
    gc_DevData.flags.txBusy = 1;
    gc_DevData.txLength = 0;

//El endpoint ahora es fijo, en este caso, el 0x0C

    RetVal = USBHostWrite( deviceAddress, 0x0C, (BYTE *)buffer, length );

    if (RetVal != USB_SUCCESS)
    {
        gc_DevData.flags.rxBusy = 0;    // Clear flag to allow re-try
    }
    return RetVal;

} // fin de USBHostGenericWrite1

//se hace una función similar llamada USBHostGenericWrite5, cambiando el endpoint fijo por
//0x05
```

1.12 Firmware de la aplicación mecatrónica (vehículo de prueba) por incluir en *main*.c

```
//Aquí se incluyen los archivos necesarios en el proyecto
//Aquí se definen las variables globales para la comunicación

//el firmware del proyecto requiere incluir lo siguiente en el archivo main.c

#define vuelta_izquierda            0x1E
#define vuelta_derecha              0x1D
#define avance                      0x1F
#define freno                       0x20

#define Gira_Derecha                5
#define Gira_Izquierda              34
#define Avanza                      39
#define Frena                       0

#define Xmas                        1
#define Xmenos                      4
#define Ymas                        2
#define Ymenos                      8

BYTE contador_ranuras =     0;     //variable de conteo de ranuras del encoder
BYTE tarea_actual     =     freno; //define la tarea que el usuario mandó por 802.11
BYTE plano     =            2;     //indica hacia qué dirección está orientado el carro
                                   //2: eje Y+, 1: eje X+, 4: eje X-, 8: eje Y-
double posicionX      =     0;     //posición en eje X del carro
double posicionY      =     0;     //posición en eje Y del carro

BOOL espera_ranura    =     FALSE; //variable que indica que se tiene que esperar hasta
                     //que se detecte nuevamente la ranura del encoder derecho (necesario
                     //para ubicar espacialmente al vehículo

int main(void)
{
       //Aquí se inicializan las variables locales de la comunicación empleada

       TRISE = 768;   //RE0 a RE5 salidas, RE8 y RE9 entradas

       //Aquí se indican la inicialización como habilitación de las interrupciones del PIC
       INTEnableSystemMultiVectoredInt();

       ConfigINT1(EXT_INT_PRI_9 | RISING_EDGE_INT | EXT_INT_ENABLE); //int Externa 1
       ConfigINT2(EXT_INT_PRI_7 | RISING_EDGE_INT | EXT_INT_ENABLE); //int Externa 2
       ConfigINT3(EXT_INT_PRI_7 | RISING_EDGE_INT | EXT_INT_ENABLE); //int Externa 3

       //Estas fueron interrupciones habilitadas con flanco de subida

       //Aquí se programa la rutina de ajuste del reloj y la inicialización módulo USB OTG
       while(1)
       {
              //Aquí se espera la conexión de la tarjeta
              USBTasks();
              while(conectado)
              {
                     //control de la tarjeta de red para iniciar su empleo
                     //control de la tarjeta de red para intentar conectarse a WLAN
                     //control de la tarjeta de red para mantenerla funcionando
```

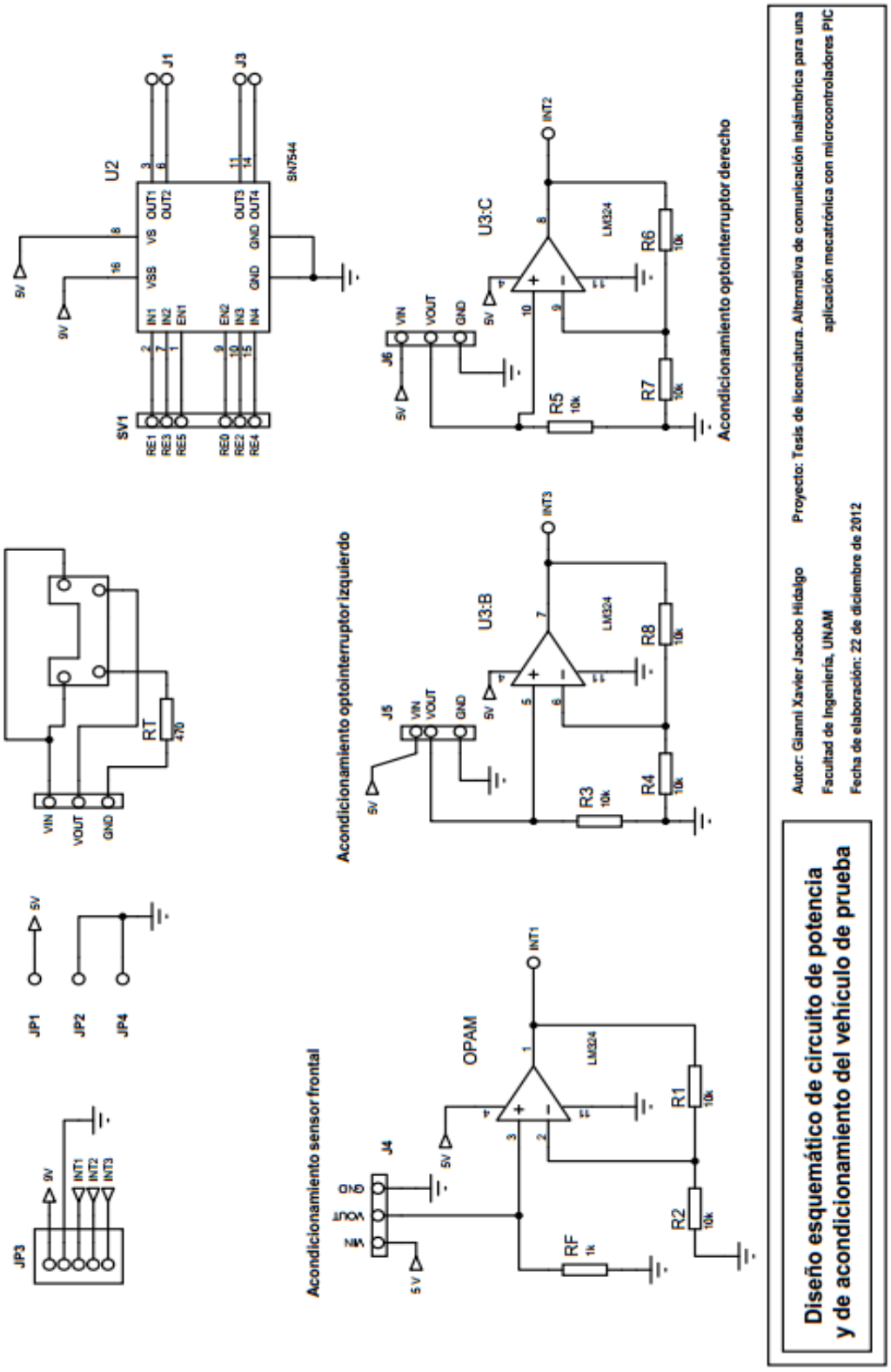
```
                    //control de la tarjeta de red para mantenerla conectada a WLAN


//FUNCIONES DEL FIRMWARE DE LA APLICACIÓN DEL VEHICULO (DESPUES DEL CONTROL DE LA
//COMUNICACIÓN INALÁMBRICA

//comprobar si la información corresponde a un PING recibido
                    if(infoIN[0]==0x88&&infoIN[39]==0x0)
                    {
                            //byte con el que se define el comando recibido por WLAN (el
                            //número de bytes que conforman el PING)

                            switch(infoIN[34])
                            {
                                    case vuelta_izquierda:

//se requiere volver a detectar la ranura, para no perder los cm recorridos

                                    if(!espera_ranura&&tarea_actual==avance)
                                    {
                                            espera_ranura = TRUE;
                                            tarea_actual=vuelta_izquierda;
                                    }
                                    else if(tarea_actual==freno) PORTE=Gira_Izquierda;
                                    break;

//en caso de que el vehículo esté "frenado", se puede dar vuelta

                                    case vuelta_derecha:
                                    if(!espera_ranura&&tarea_actual==avance)
                                    {
                                            espera_ranura = TRUE;
                                            tarea_actual=vuelta_derecha;
                                    }
                                    else if(tarea_actual==freno) PORTE=Gira_Derecha;
                                    break;

                                    case avance:
// primero determiner si el vehículo no está a punto de llegar a la frontera

                                    if(PreCalculo())
                                    {
                                            PORTE=Avanza;
                                            tarea_actual=avance;
                                    }
                                    else
                                    {
                                            PORTE=Frena;
                                            tarea_actual=freno;
//el vehículo está por llegar a la frontera, no puede avanzar, sólo dar vueltas
                                    }
                                    break;

                                    case freno:
//el vehículo necesita detector las ranuras derechas para poder frenar y no perder la
//posición avanzada
                                    if(tarea_actual==avance) espera_ranura=TRUE;
                                    tarea_actual=freno;
                                    break;
                                    }
```

```
                        }//fin de switch[34]
                    }//fin de if PING
                }//fin de while conectado
        }//fin de main


//FUNCIONES DE INTERRUPCION DEL FIRMWARE, SE REQUIERE DEFINIRLAS EN CONFIGURACION.H

        void __ISR(_EXTERNAL_1_VECTOR, ipl7) INT1Interrupt()
        {
                //objeto frontal detectado
                PORTE = Frena;          //apagar todo el puerto E
                tarea_actual = freno;
//por precaución, el vehículo se detiene antes de colisionar
                mINT1ClearIntFlag();  //limpiar bandera de interrupción
        }

        void __ISR(_EXTERNAL_2_VECTOR, ipl7) INT2Interrupt()
        {
                //ranura del disco del encoder derecho detectada
                if(espera_ranura)
                {
//se ha detectado la ranura esperada, ahora sí es posible realizar los giros o frenar
                        if(tarea_actual==vuelta_izquierda) PORTE=Gira_Izquieda;
                        else if(tarea_actual==vuelta_derecha) PORTE=Gira_Derecha;
                        else PORTE=Frena;

                        espera_ranura=FALSE;
                        Ajuste_Posicion();
                }
                else if(tarea_actual==vuelta_izquierda&&!espera_ranura)
                {
//si ya no se requiere esperar la ranura, es porque ya se encuentra girando el vehículo
                        contador_ranuras++;
                        if(contador_ranuras==4)
                        {
//4 ranuras son necesarias detectarse para saber que el vehículo ha girado 90 grados
                                PORTE=Frena;
                                tarea_actual=freno;
                                contador_ranuras=0;
                                plano*=2; //cambiar la dirección a la que apunta el carro
                                                //en sentido antihorario
                                if(plano==16) plano=1;
                        }
                }
                else if(tarea_actual==avance)
                {
//el vehículo está avanzando, ha recorrido 3.75 cm más, agregarlos a la posición espacial

                        Ajuste_Posicion();

//puede seguir avanzando el vehículo (no sobrepasa la frontera)?
                        if(PreCalculo())
                        {
                                PORTE=Avanza;
                                tarea_actual=avance;
                        }
                        else
                        {
                                PORTE=Frena;
                                tarea_actual=freno;
```

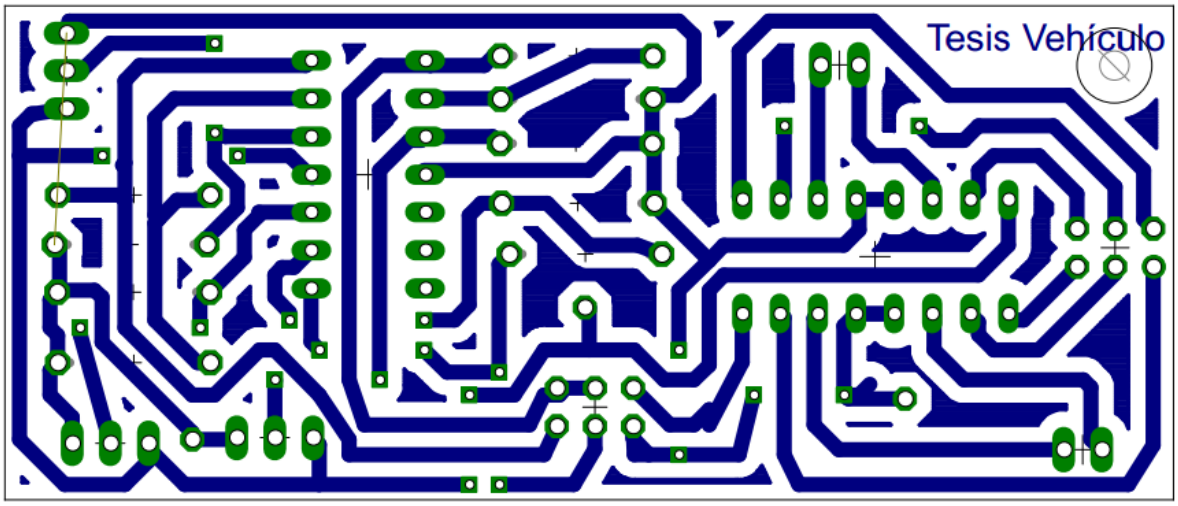
```
                }
        }
        mINT2ClearIntFlag();
}



void __ISR(_EXTERNAL_3_VECTOR, ipl7) INT3Interrupt()
{
        //ranura del disco del encoder izquierdo detectada
        if(tarea_actual==vuelta_derecha&&!espera_ranura)
        {
                contador_ranuras++;
                if(contador_ranuras==4)
                {
                        PORTE=Frena;
                        tarea_actual=freno;
                        contador_ranuras=0;
                        plano>>1;
                        if(plano==0) plano=8;
                }
        }
        mINT3ClearIntFlag();
}

//FUNCIONES ADICIONALES

void Ajuste_Posicion()
{
        switch(plano)
        {
                case Xmas:
                posicionX = posicionX + 3.75;
                break;
                case Xmenos:
                posicionX = posicionX - 3.75;
                break;
                case Ymas:
                posicionY = posicionY + 3.75;
                break;
                case Ymenos:
                posicionY = posicionY - 3.75;
                break;
        }
}

BOOL PreCalculo()
{
        if((   posicionX + 3.75) > 900||(posicionX - 3.75)< -900
                ||(posicionY + 3.75 ) > 900
                ||(posicionY - 3.75)<900) return FALSE;
        else    return TRUE;
}
```

Sección 2. Esquemas electrónicos

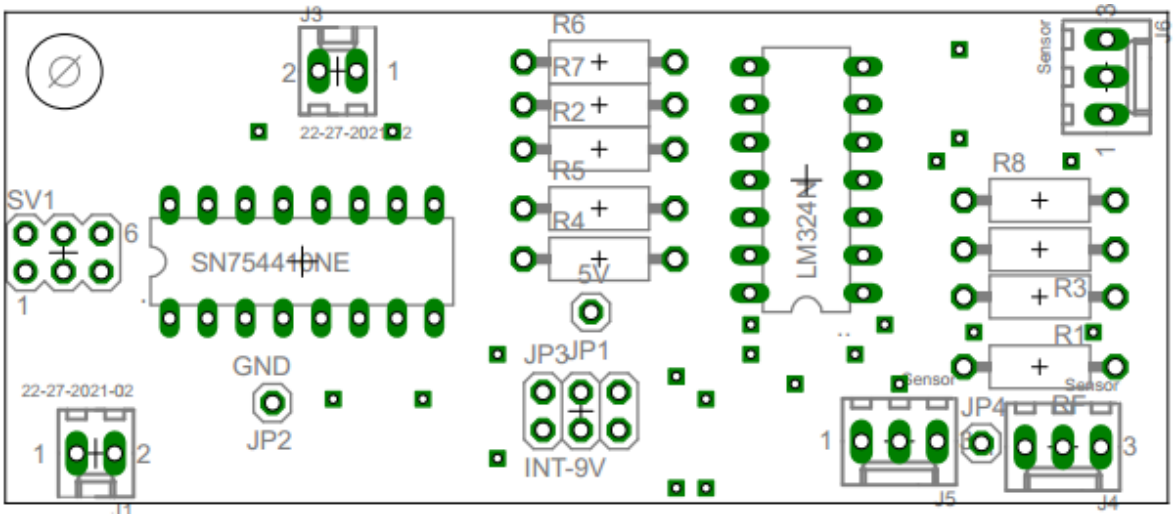
2.1 Diagrama electrónico de la aplicación



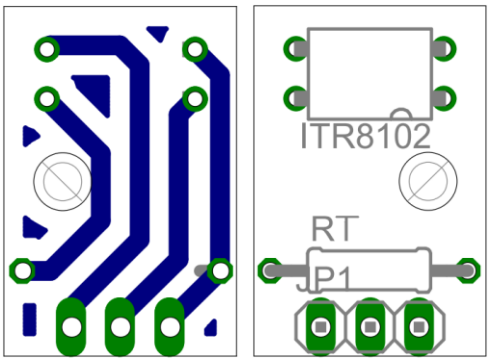
2.2 PCB principal y de optointerruptores



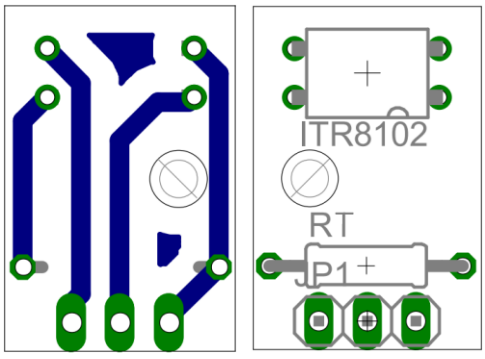
PCB principal



Componentes PCB principal



PCB optointerruptor derecho



PCB optointerruptor izquierdo

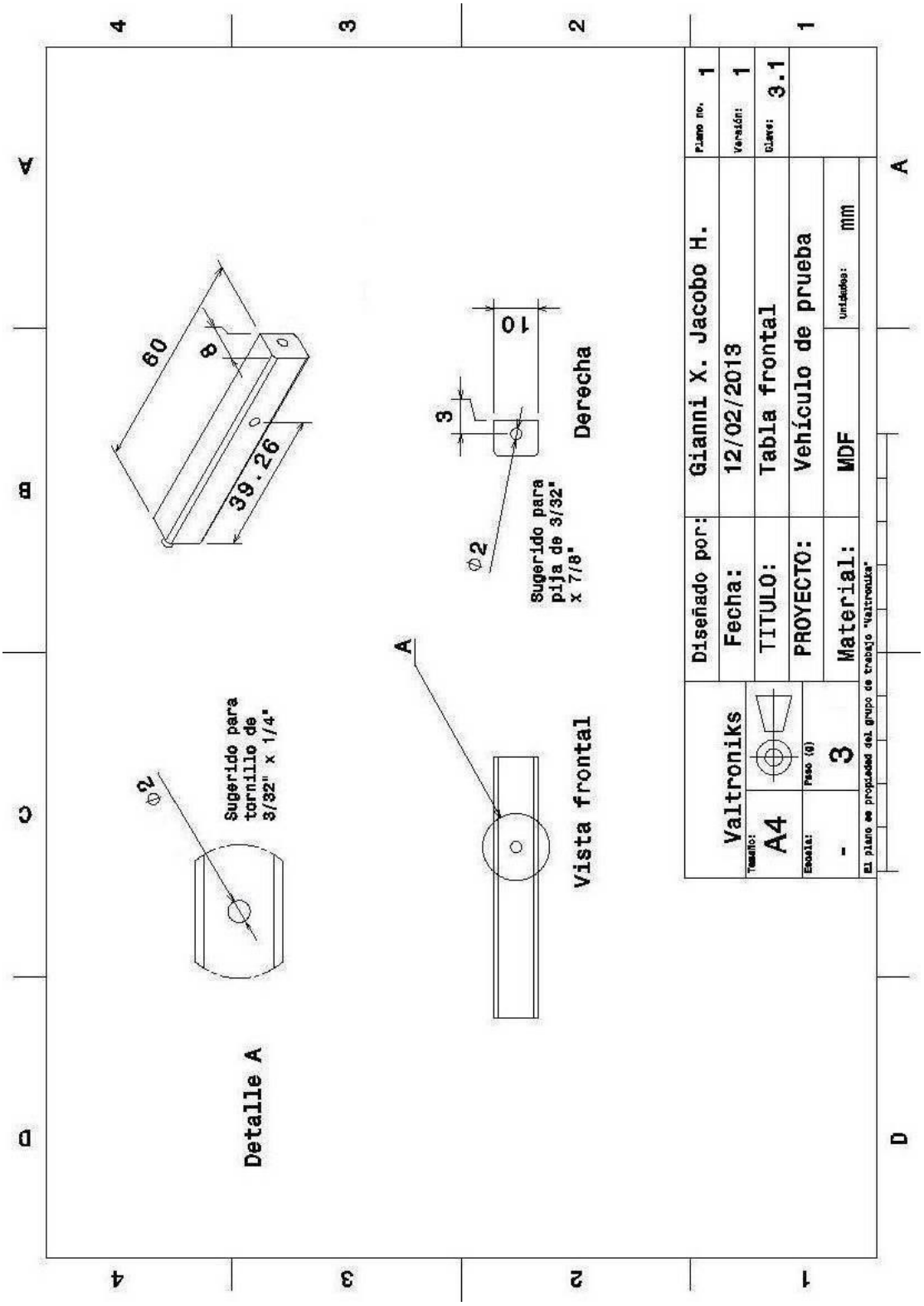
2.3 Lista de materiales electrónicos

PCB principal		
9	Resistencias 10k	R1 – R8, RF
1	LM324	U3
1	SN754410NE	U2
3	Molex macho 3 x 1 pines	J4, J5, J6
2	Molex macho 2 x 1 pines	J1, J3
2	Header 2 x 3 pines	SV1, JP3
3	Header 1 pin	JP1, JP2, JP4
1	Tarjeta fenólica 78 mm x 33 mm	-
1	GP2Y0D810Z0F Sharp con PCB	-

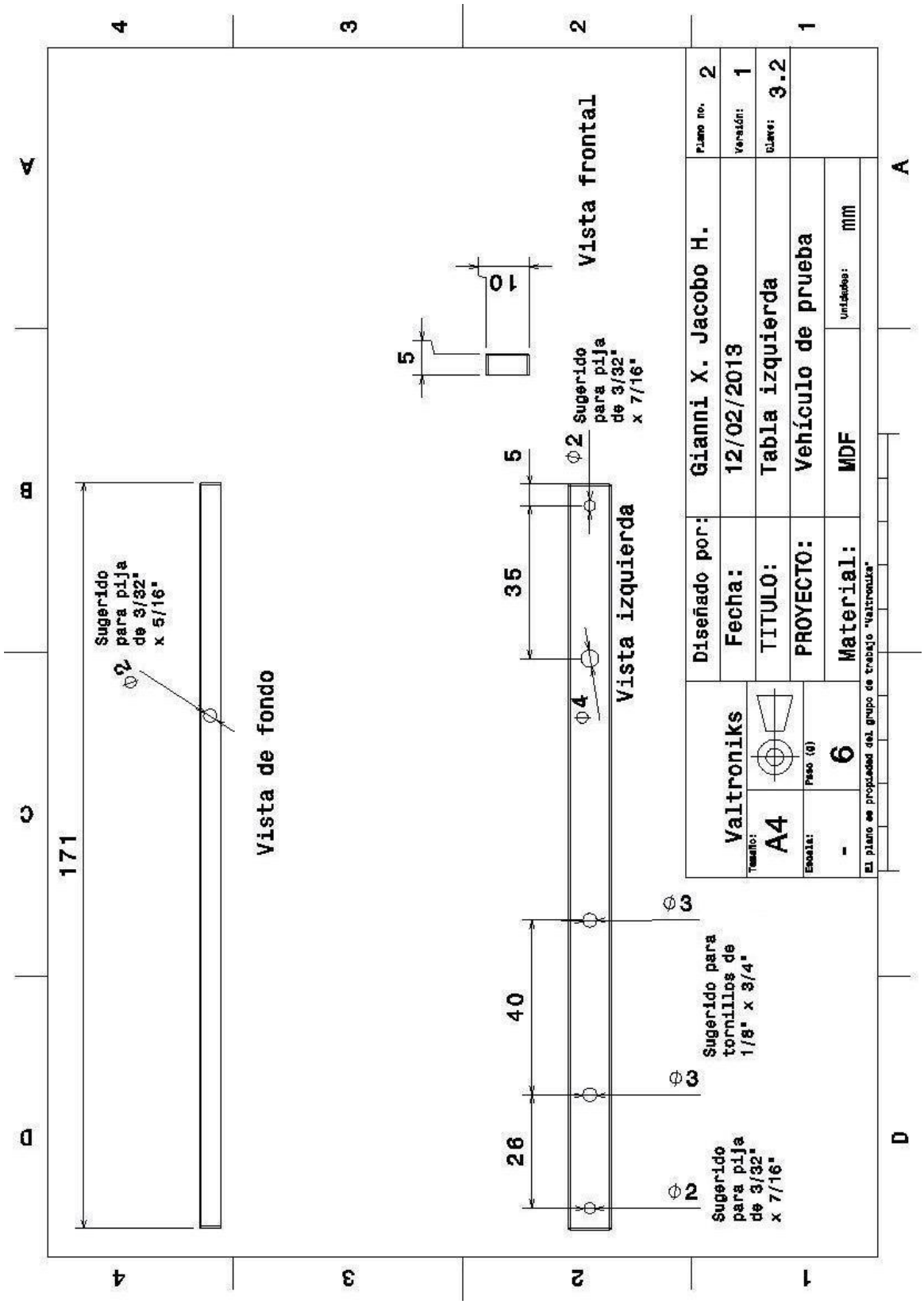
PCB optointerruptor		
1	Resistencia 470 Ω	RT
1	ITR8102	-
1	Header 3 x 1 pines	-
1	Tarjeta fenólica 18 mm x 12 mm	-

Sección 3. Planos

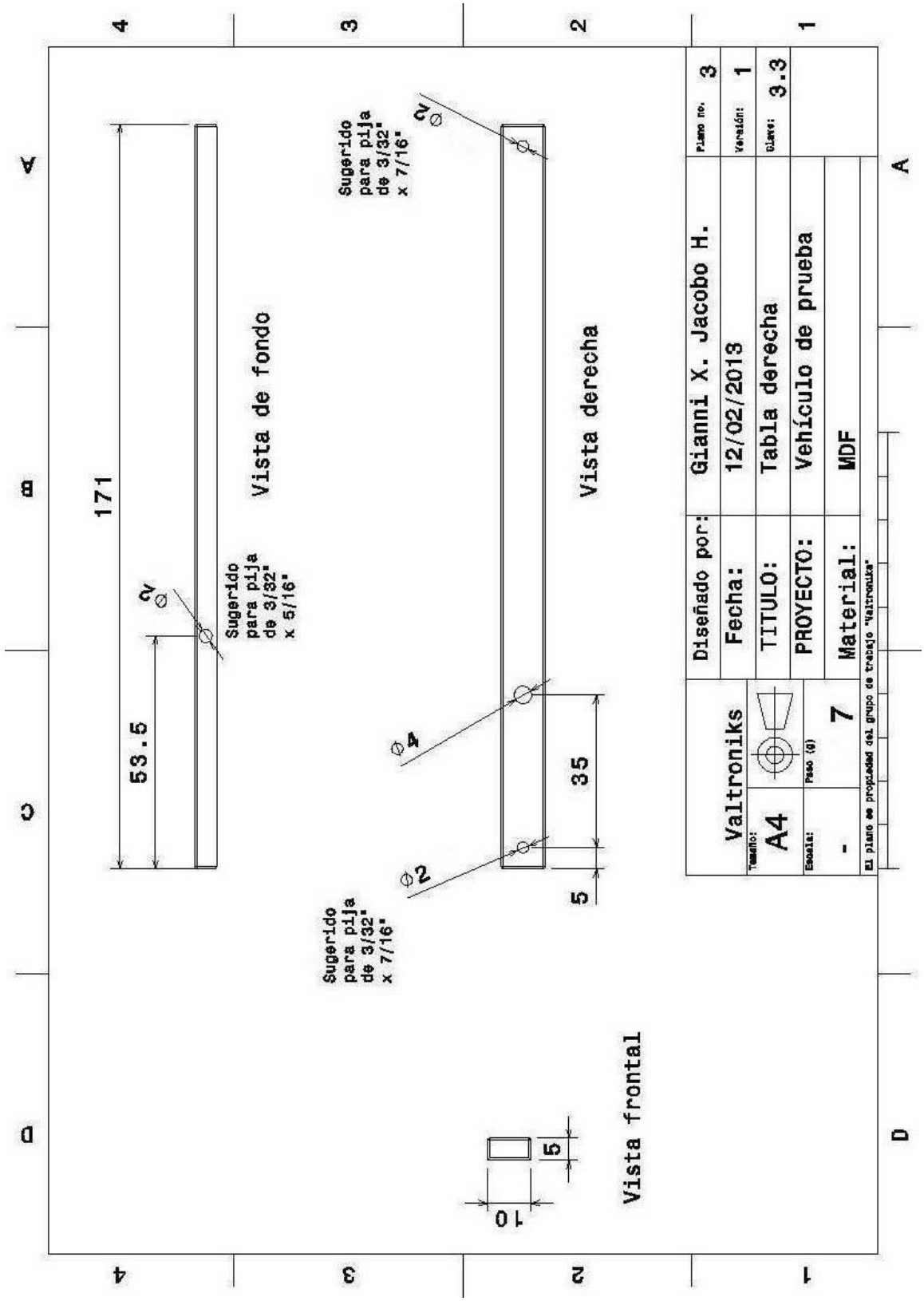
3.1 Tabla frontal



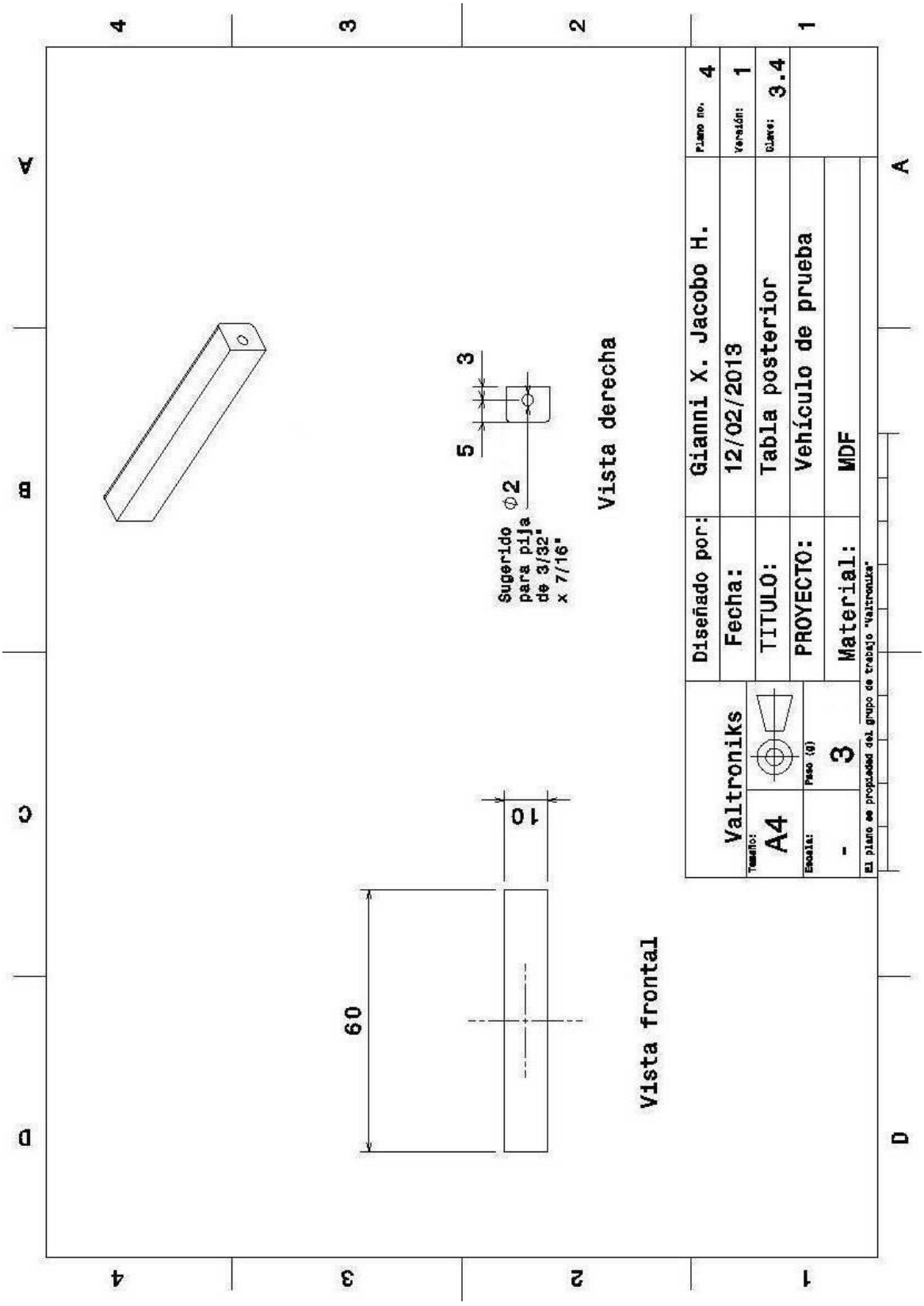
3.2 Tabla lateral izquierda



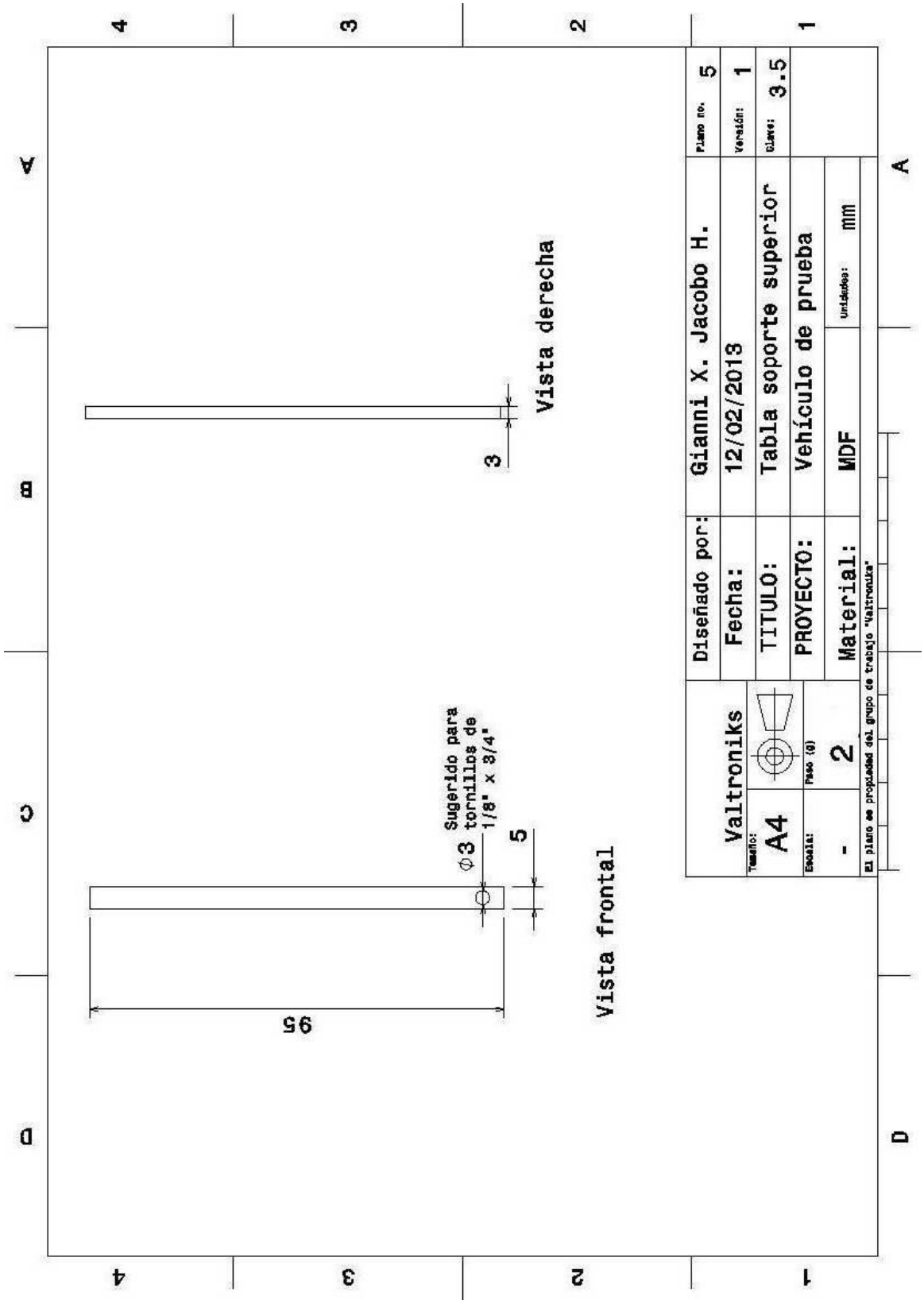
3.3 Tabla derecha



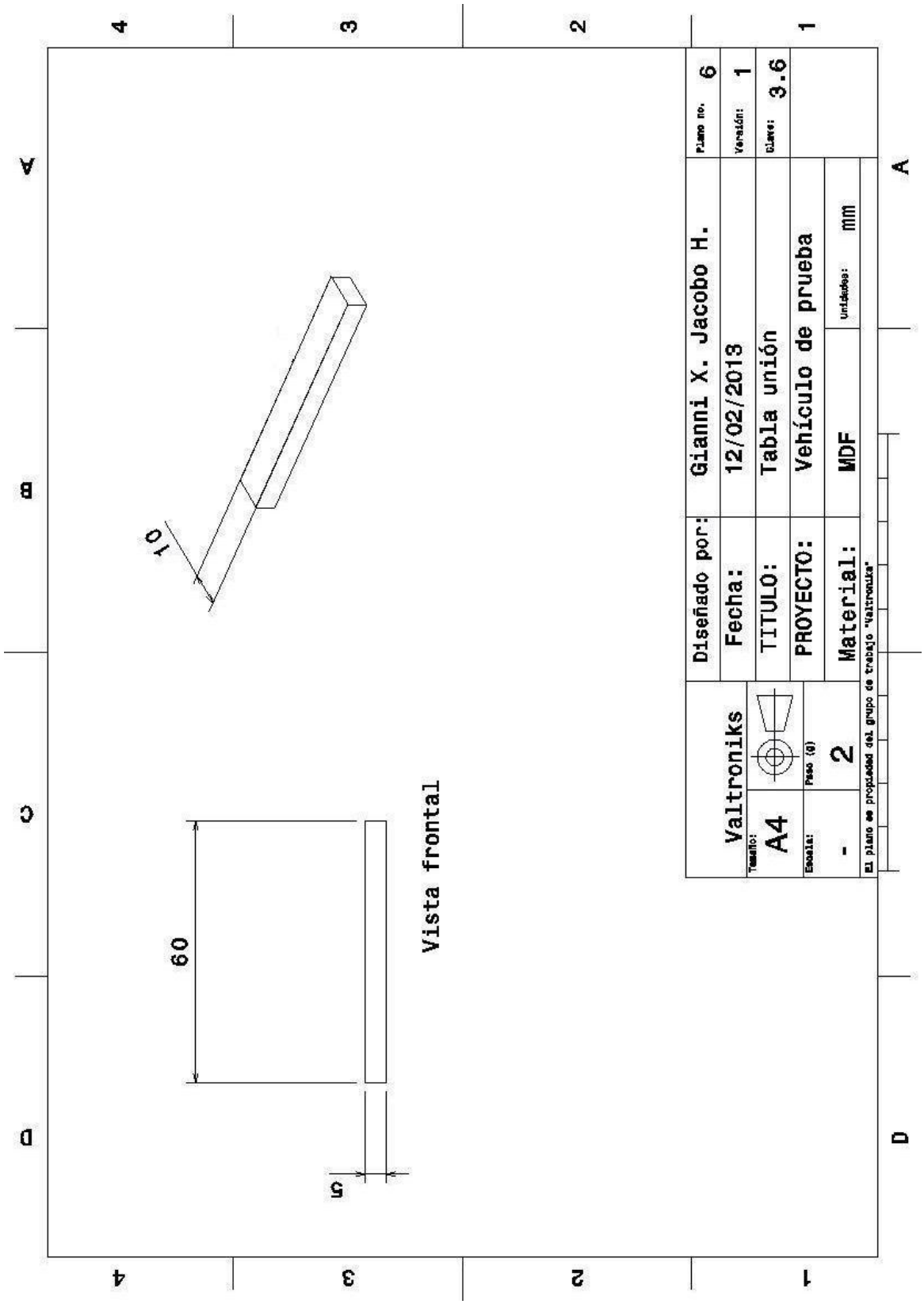
3.4 Tabla posterior



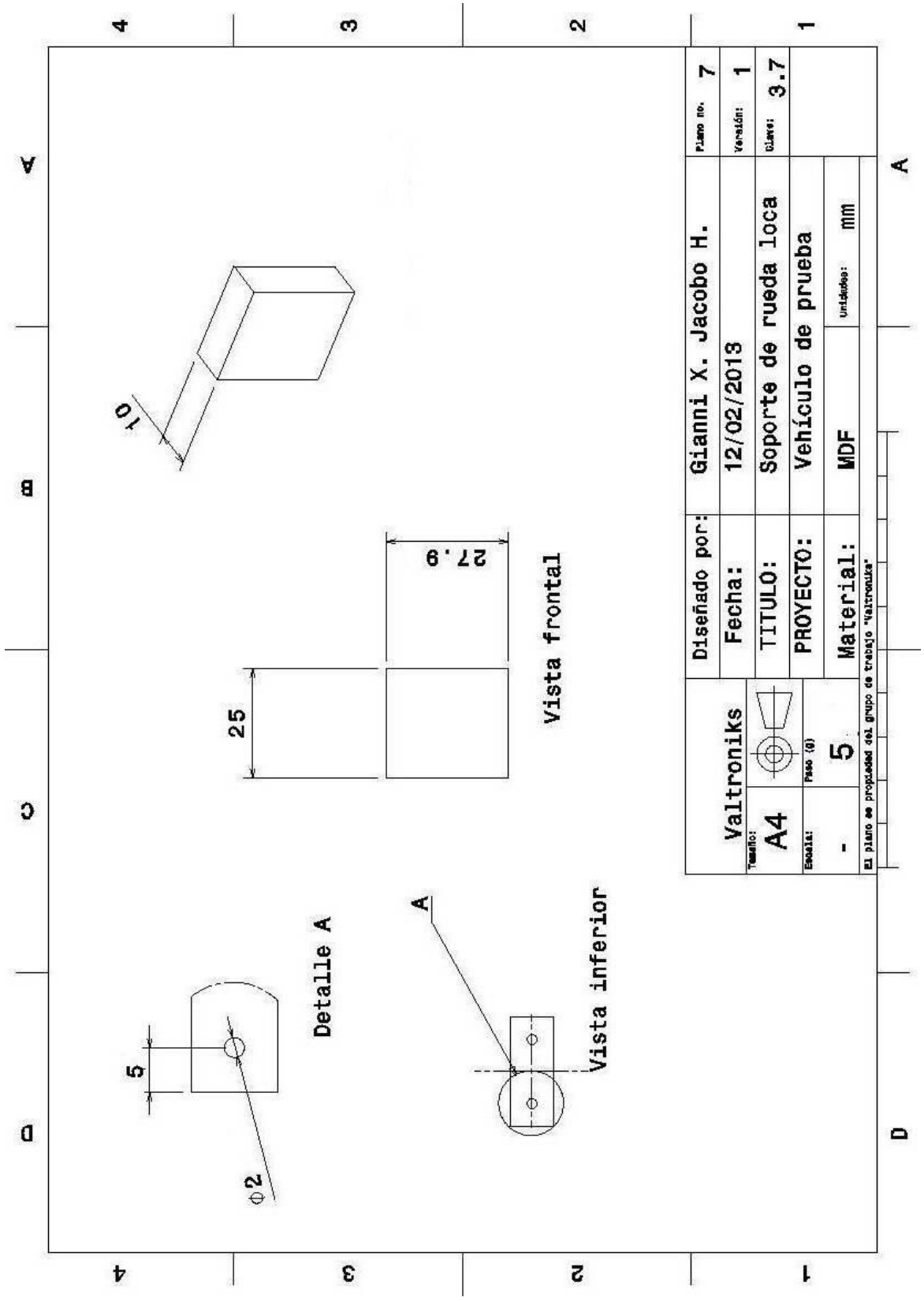
3.5 Tabla soporte superior



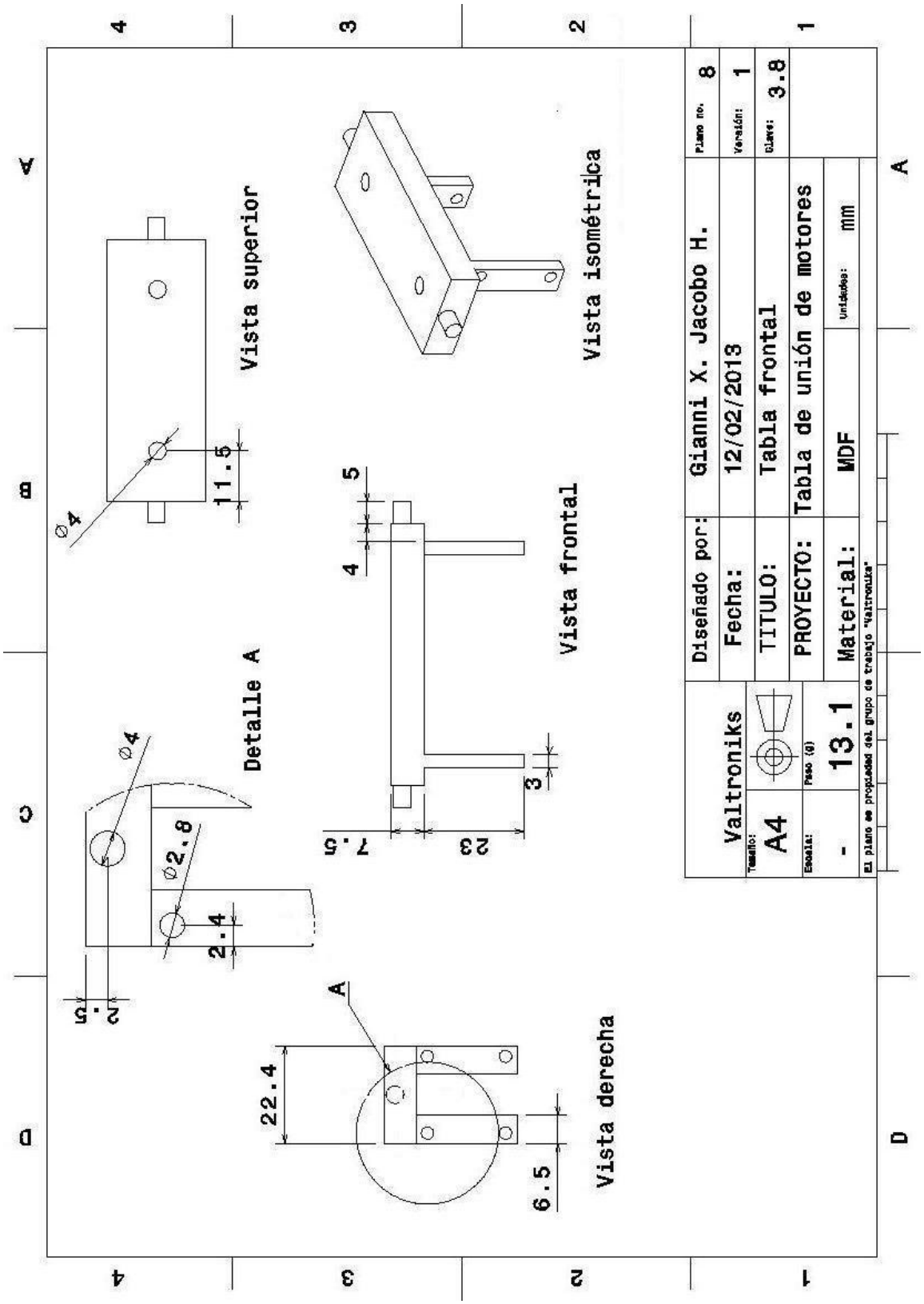
3.6 Tabla unión



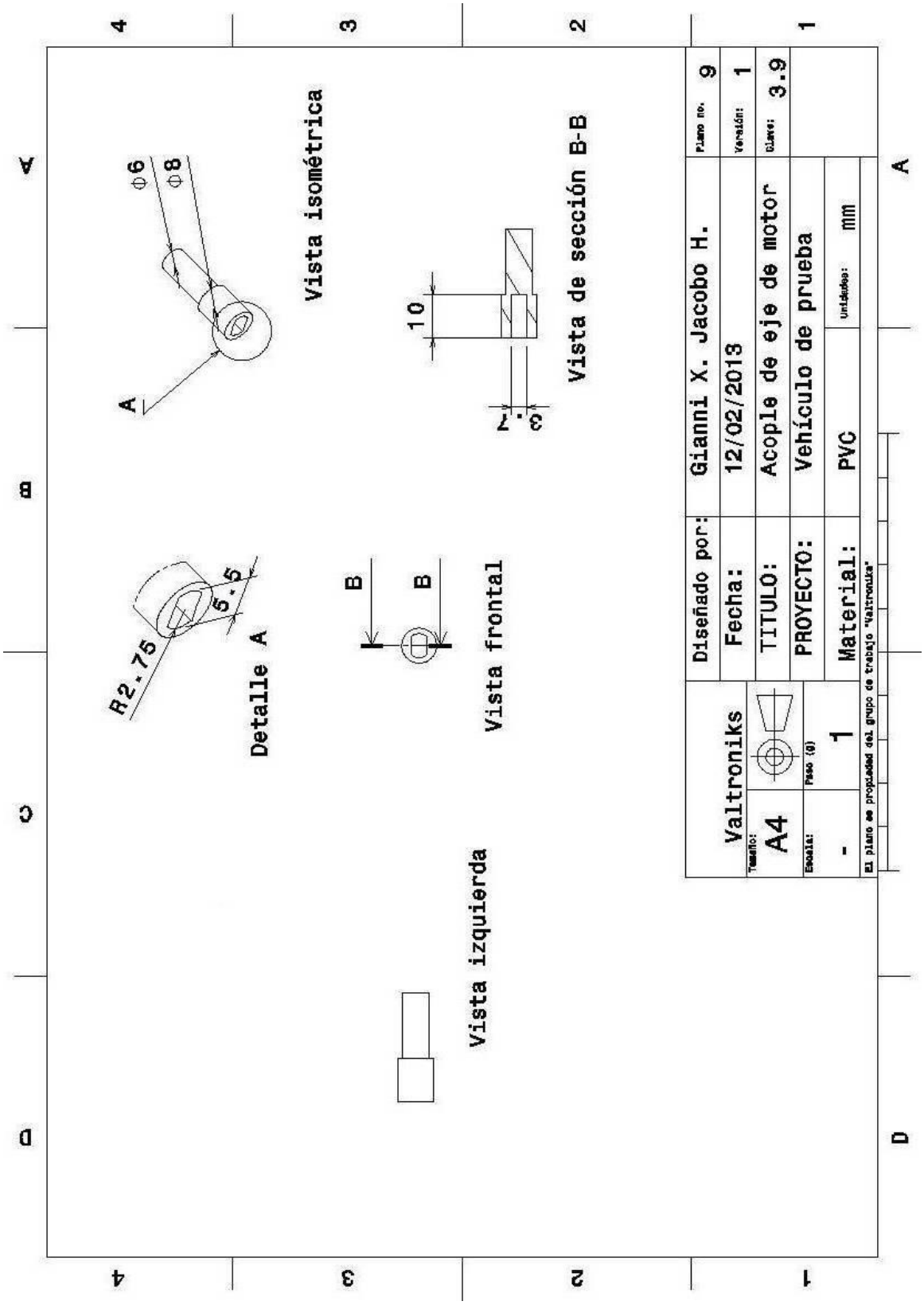
3.7 Soporte de rueda loca



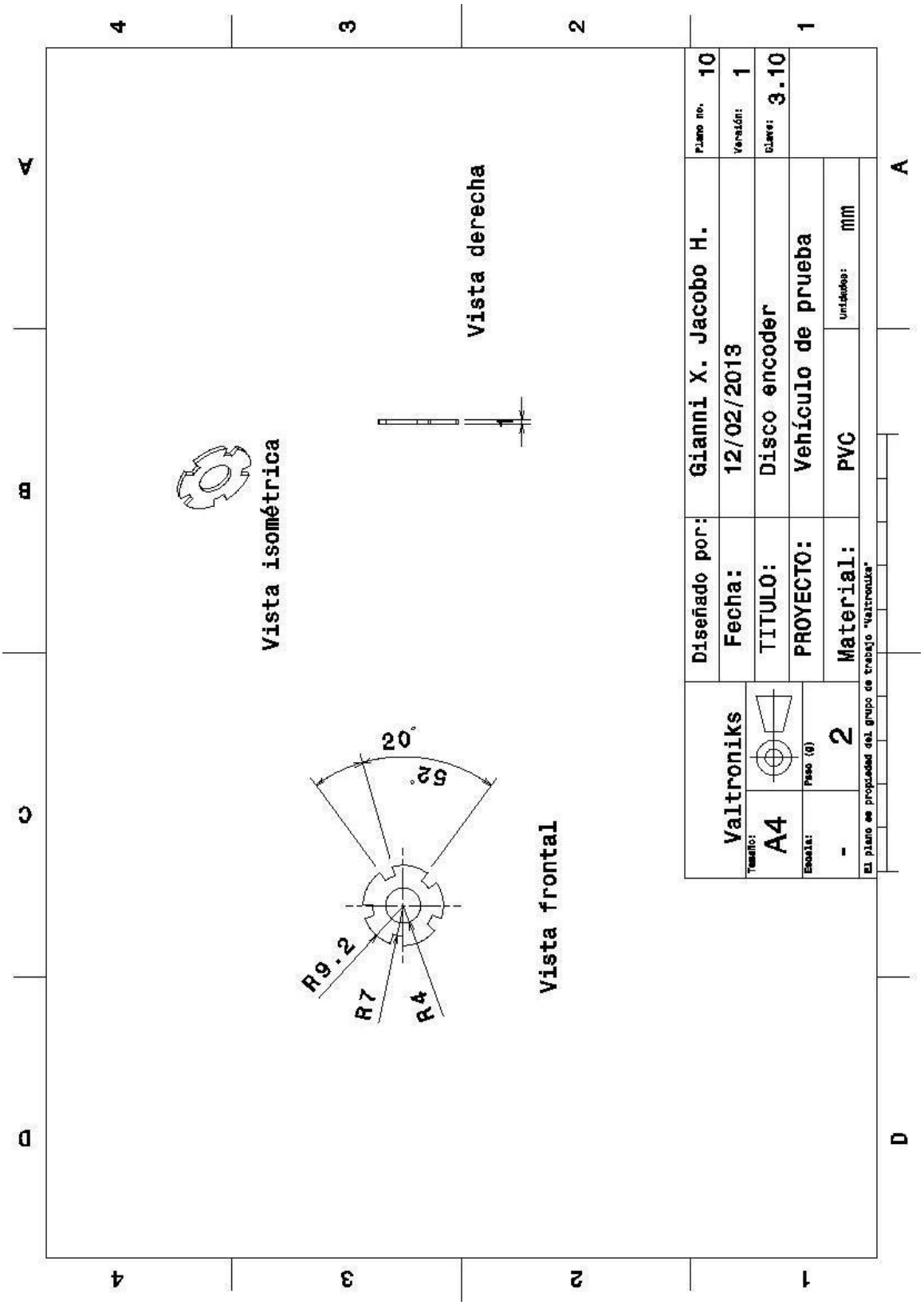
3.8 Tabla de unión de motores



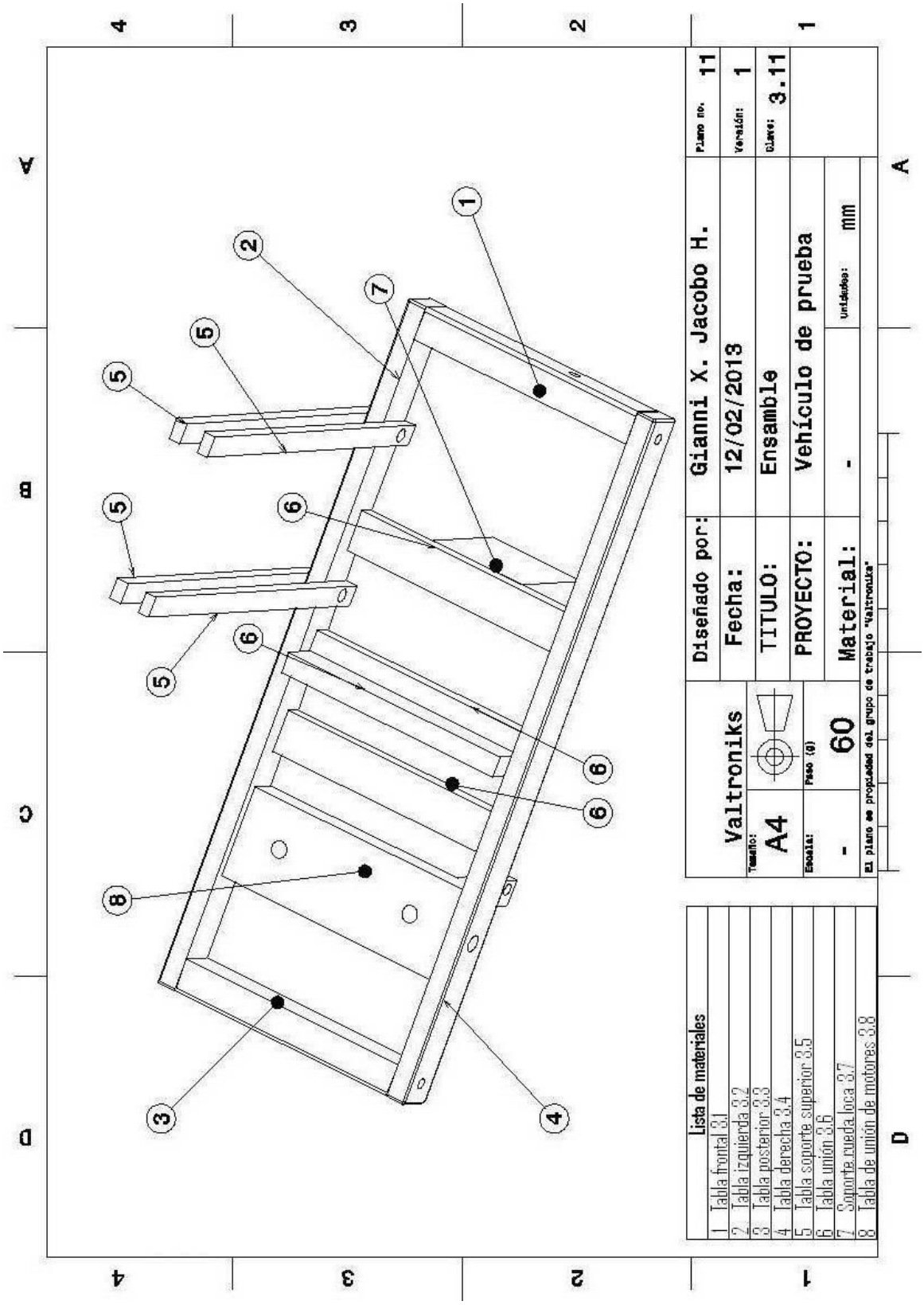
3.9 Acople de eje de motor



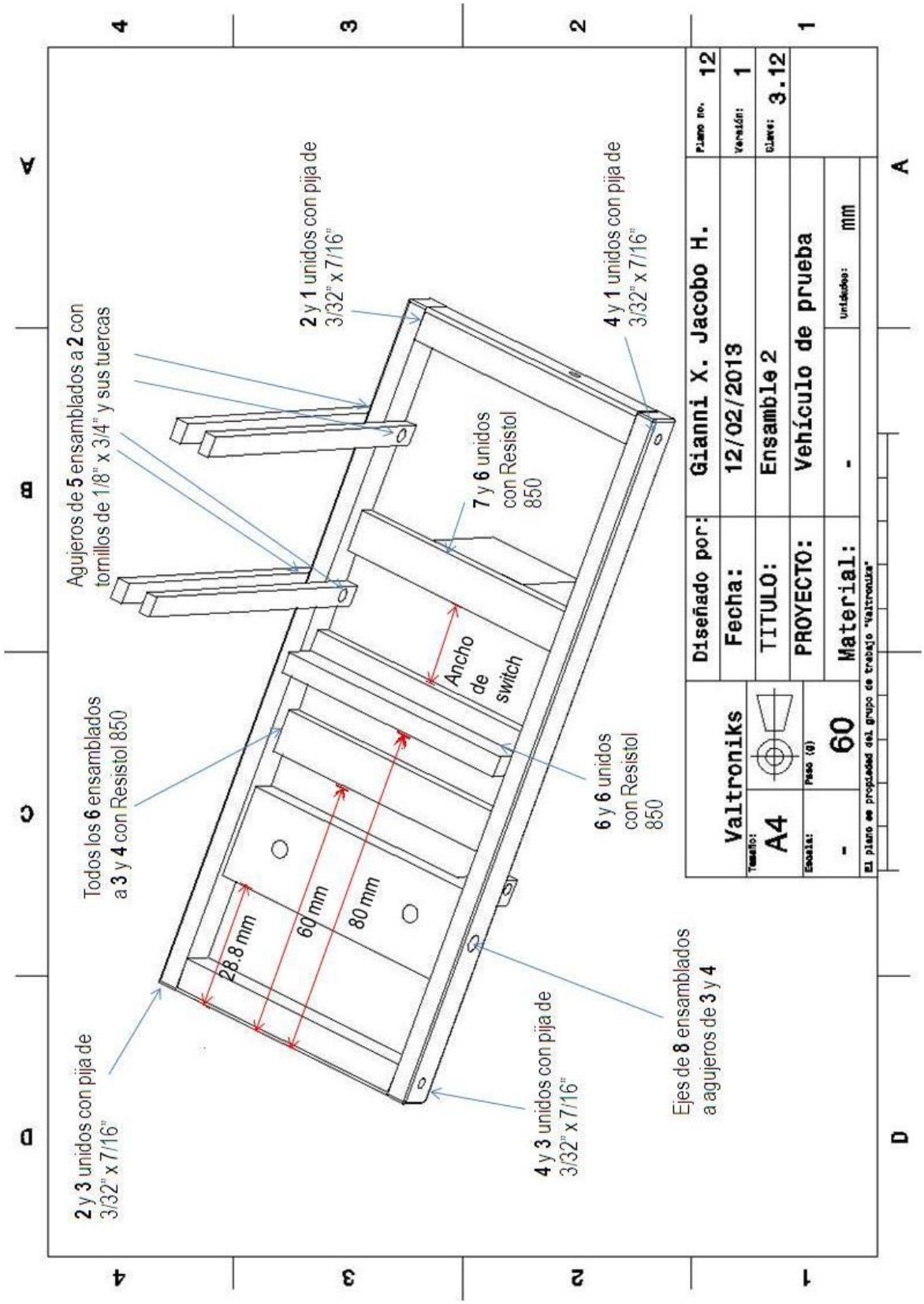
3.10 Disco encoder



3.11 Ensamble (partes)



3.12 Ensamble 2



Referencias

HYDE. J. *USB Design by Example: A Practical Guide to Building I/O Devices*. USA. Intel Press, 2001

AXELSON. J. *USB Complete. Everything You Need To Develop Custom USB Peripherals*. USA. Lakeview Research Ilc, 2005

Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V. *USB 2.0 Specification* [en línea]. Disponible en Internet:
http://www.usb.org/developers/docs/

Microchip Technology Incorporated. *PIC32MX5XX/6XX/7XX Family Data Sheet. High-Performance, USB, CAN and Ethernet 32-bit Flash Microcontrollers*. USA (2010). Formato PDF disponible en Internet:
http://ww1.microchip.com/downloads/en/DeviceDoc/61156F.pdf

Microchip Technology Incorporated. *PIC32 USB Starter Kit II User's Guide*. USA (2009). Formato PDF disponible en Internet:
http://ww1.microchip.com/downloads/en/devicedoc/61158a.pdf

Microchip Technology Incorporated. *Starter Kit I/O Expansion Board Information Sheet*. USA (2011). Formato PDF disponible en Internet:
http://ww1.microchip.com/downloads/en/DeviceDoc/51950B.pdf

PEACOCK. C. (2010), *Beyond Logic. USB in a NutShell. Making sense of the USB Standard,* [en línea]. Consultada en Internet el 22 de febrero de 2013.
http://www.beyondlogic.org/usbnutshell/usb1.shtml

OTTEN. K. *USB Embedded Host Stack*. USA. Microchip Technology Incorporated, 2009. Formato PDF disponible en Internet:
http://ww1.microchip.com/downloads/en/AppNotes/USB_Host_Stack_01140a.pdf

CALDWELL. B. *USB Embedded Host Stack Programmer's Guide*. USA. Microchip Technology Incorporated, 2008. Formato PDF disponible en Internet:
http://ww1.microchip.com/downloads/en/AppNotes/01141a.pdf

OTTEN. K., CALDWELL. B. *Generic Client Driver for a USB Embedded Host*. USA. Microchip Technology Incorporated, 2008. Formato PDF disponible en Internet:
http://ww1.microchip.com/downloads/en/AppNotes/01143a.pdf

ULRICH, K. *Diseño y Desarrollo de Productos, Enfoque Multidisciplinario*. México, 3a ed., McGraw Hill, 2006.

VILAS, J. *Sistema multimedia para la enseñanza de los sensores de proximidad*. Departamento de tecnología electrónica. Universidad de Vigo. Consultada el 25 de febrero de 2013. Disponible en Internet:
http://www.dte.uvigo.es/recursos/proximidad/Sensores_Proximidad.swf

PEÑUELAS, U. *Metodología para diseño mecatrónico*. Tesis de maestría. México. Facultad de Ingeniería, UNAM, 2007. Disponible en Internet:
http://www.dgbiblio.unam.mx/

Evaluación de métricas de encaminamiento basadas en LQI para un protocolo de encaminamiento en redes IEEE802.15.4.
http://upcommons.upc.edu/pfc/bitstream/2099.1/6026/1/memoria.pdf
(27 Feb. 2012).

El estándar IEEE 802.15.3.
http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo5.pdf
(2 Mar. 2012).

IANA. *IP Option munbers. IP números de opciones.* Formato: xml.
www.iana.org/assignments/ip-parameters
(23 Abr. 2012).

Seguridad en redes Inalámbricas: Una guía básica. Formato: PDF
http://dns.bdat.net/seguridad_en_redes_inalambricas/x187.html
(4 Mar 2012).

Redes inalámbricas. Estándar IEEE 802.11. Formato PDF
http://gsyc.es/~mortuno/rii/02-802.11.pdf
(7 May. 2012).

RFC 791 Internet Protocol. Formato: txt
http://www.ietf.org/rfc/rfc791.txt
(4 Sep 2012).

Apuntes de la asignatura de Redes de Datos impartida por el M.C. Alejandro Velázquez Mena semestre 2010-2.

Apuntes de la asignatura de Redes de Datos impartida por el M.C. Marco A. Vigueras Villaseñor semestre 2012-1.

Apuntes de la asignatura de Redes de Datos impartida por la M.C Ma. Jaquelina López Barrientos semestre 2012-2.

Glosario

Adaptadores de red: Son dispositivos electrónicos que permiten que una computadora intercambie información con otras. Se usan especialmente en aquellas que carezcan de esta capacidad o se requiere que el intercambio se realice de otra forma, por ejemplo, de manera inalámbrica o alámbrica.

Bit Stuffing (Relleno de bit)**:** Es una adecuación a NZRI, agregando un bit extra cada cierto número de "0" lógicos, con el objetivo de descifrar adecuadamente los bytes de información.

Canal de comunicación: Medio de transmisión de datos. En 802.11 se divide el medio en canales no superpuestos

Controlador informático: En SO Windows, son bloques de instrucciones que permiten a una computadora conocer la forma en la que se debe trabajar con el hardware integrado a ella. En GNU/Linux, los controladores son llamados *módulos*. Sin el controlador, una computadora ignora cómo proceder con el funcionamiento de un nuevo dispositivo.

Debug (depurar)**:** Es una simulación de ejecutar el programa para encontrar errores en la programación informática.

Dominio de broadcast: Es cuando un dispositivo envía datos a todos los que puedan escucharlo, sin un destino en particular. Para la capa de enlace de datos, la dirección broadcast es 0xFF FF FF FF FF FF.

Encapsulamiento: Proceso mediante el cual se agrega información necesaria de cada protocolo a los datos para ser enviados por la red

Firmware: Son instrucciones grabadas internamente que controlan directamente los circuitos electrónicos y el hardware restante. Es el software de nivel más bajo en un dispositivo electrónico.

Interrupciones informáticas: Son señales eléctricas enviadas a un control principal, que lo obliga a abandonar su proceso actual para ejecutar algún otro, llamado comúnmente Rutina de Servicio de Interrupción (ISR). Las señales eléctricas son transferidas por líneas llamadas IRQ (Petición de Interrupción). Sin embargo también familiarmente se le dice así cuando la interrupción es interpretada por el control y se conoce su origen o naturaleza.

Línea de vista: Se refiere a que entre el emisor y receptor de ondas se forme una línea recta sin obstrucciones.

MAL (Microchip Application Libraries)**:** Es un conjunto de diferentes proyectos y archivos que pueden ser tomados como plantillas para la elaboración de otros. Su principal finalidad es mostrar al usuario las diversas tareas que pueden realizarse empleando a los PIC u otras creaciones de Microchip.

Modo Ping-Pong: A grandes rasgos, se emplean registros para las transferencias USB, intercalándose entre ellos para que, mientras el módulo USB OTG trabaja con uno, el MCU pueda manipular otro. Esta manera de interactuar es denominada modo *ping-pong*, mejorando la eficiencia del manejo de la información.

NZRI (No Retorno a Cero Invertido)**:** Es una técnica empleada para codificar señales binarias, consistente en alterar el valor de algunos "1" lógicos.

Puerto: Interfaz para enviar y recibir datos, de forma lógica o física.

Punto de acceso (AP)**:** Dispositivo que permite interconectar equipos para formar una red de manera inalámbrica.

PWM (Modulación de Ancho de Pulso)**:** De manera simple, consiste mantener un nivel lógico de voltaje durante determinada fracción de una frecuencia en el tiempo conocido como ciclo de trabajo. Esto hace que las señales digitales "aparenten" tener cualquier valor de voltaje (o señal analógica).`

Resistencia a la tracción: Es una característica de los materiales, empleada para comparar su ductilidad. Se mide en unidades de esfuerzos o presiones.

Router: Dispositivo que encamina paquetes de datos en la red.

Salto: En redes, un salto es cuando un paquete de datos pasa de un dispositivo a otro.

Software demo: Es uno que tiene limitantes en comparación a su versión completa. Las versiones completas normalmente tienen un precio monetario o necesitan una donación económica para funcionar en su totalidad.

Unidad de Datos de Protocolo (PDU)**:** Es el tamaño del mensaje completo de cada capa.

Vector: En los MCU PIC, son espacios de memoria en donde las interrupciones son direccionadas en la memoria Flash.