

UNIVERSIDAD NACIONAL DE MAR DEL PLATA

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

---

**Control a través de Internet de una red  
domótica X-10 mediante microcontroladores**

---

Trabajo presentado por

**Martín Pérez**

**Matrícula: 9639**

para obtener el Grado de

**INGENIERO ELECTRÓNICO**

*Director de la Tesis:* Ing. Pablo Daniel Agüero.  
*Codirector:* Ing. Alejandro Jose Uriz.

Mar del Plata

3 de Septiembre de 2012



# Resumen

El principal objetivo de este proyecto final consistió en mejorar la calidad de vida de personas con movilidad reducida. Para lograr dicho objetivo, se diseñó un dispositivo que permite comandar una red domótica desde una cama o una silla de ruedas. Mediante la red domótica se pueden controlar las luces, persianas, calefacción, o cualquier electrodoméstico que se encuentre en el hogar.

El dispositivo diseñado es comandado por medio de una página web. Se puede acceder a la misma desde una computadora u otro dispositivo que sea capaz de manejar el protocolo TCP/IP. Esto permite controlar y monitorizar múltiples artefactos eléctricos existentes en el hogar mediante Internet.

Después de estudiar detalladamente los protocolos domóticos existentes en el mercado, se decidió utilizar el protocolo X-10 para realizar el proyecto. Este es uno de los protocolos más antiguos que se están usando en aplicaciones domóticas. Con este protocolo se puede controlar cualquier dispositivo a través de un cableado compartido, para lo cual se utiliza la red eléctrica doméstica (120V o 220V y 60Hz o 50Hz). Por lo tanto, no se debe agregar ninguna red de cables para la automatización de una vivienda. Esto hace a X-10 sumamente versátil y económico. Cualquier toma corriente de la red eléctrica se puede convertir en un toma de la red domótica.

El sistema desarrollado posee dos piezas fundamentales: el módulo X-10 y el módulo Ethernet. El conjunto de los dos módulos fue denominado interfaz 802.3 / X-10

El módulo transceptor de X-10 puede ser utilizado para cumplir dos funciones distintas. Por un lado se lo puede configurar para formar parte de la interfaz 802.3/X-10. Cuando esto sucede, debe recibir los datos del módulo Ethernet y enviarlos por la red eléctrica. Por último, debe recibir la confirmación (*Hail Acknowledge*), enviada por el receptor de la orden, que indica que la transmisión fue realizada correctamente. Por otra parte, el módulo transceptor X-10 puede utilizarse como un actuador bidireccional X-10. Cuando se configura de esta forma es capaz de recibir una orden mediante la red eléctrica, ejecutarla y enviar la confirmación al transmisor.

El módulo Ethernet es la parte de la interfaz 802.3/X-10 que tiene a su cargo todas las tareas relacionadas con Internet. Se trata de un servidor web embebido en un microcontrolador que se conecta a la red mediante un *router* (o un *modem*) ADSL y tiene alojada una página de Internet. El usuario puede ingresar a dicha página web utilizando una computadora (o cualquier dispositivo capaz de manejar el protocolo TCP/IP.), para comandar los diferentes actuadores de la red y ver el estado de los mismos. Cuando el módulo Ethernet recibe una orden proveniente de la página web, la decodifica, y envía un mensaje al módulo X-10. El hardware del módulo está comandado por la pila TCP/IP de Microchip, que es un software especialmente diseñado para el desarrollo de aplicaciones de Internet embebidas en microcontroladores.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Domótica. Definición . . . . .	1
1.1.1. Caracterización de un Sistema Domótico . . . . .	3
1.2. Principales protocolos existentes en el mercado . . . . .	6
1.2.1. Protocolo EIB (Bus de Instalación Eléctrica) . . . . .	6
1.2.2. Protocolo Konnex (KNX) . . . . .	8
1.2.3. Protocolo X-10 . . . . .	9
1.3. Elección del protocolo domótico más adecuado para la realización del proyecto . . . . .	11
1.4. Modelo del sistema a desarrollar . . . . .	12
1.5. Conclusiones . . . . .	12
<b>2. El módulo transceptor X-10</b>	<b>15</b>
2.1. Principios de funcionamiento de X-10 . . . . .	15
2.2. Hardware necesario para la implementación del módulo . . . . .	19
2.2.1. Detector de cruce por cero . . . . .	20
2.2.2. Etapa de entrada/salida . . . . .	23
2.2.3. Amplificador de salida . . . . .	26
2.2.4. El filtro pasobandas . . . . .	27
2.2.5. Amplificador sintonizado y detector de envolvente . . . . .	28
2.2.6. Fuente de alimentación . . . . .	30
2.3. Microcontrolador . . . . .	32
2.3.1. Configuración del Oscilador . . . . .	35
2.3.2. Mapeo de pines . . . . .	36
2.3.3. Módulos temporizadores . . . . .	38
2.3.4. Módulo Output Compare . . . . .	40
2.4. Descripción del software asociado . . . . .	44
2.4.1. Programa principal . . . . .	44
2.4.2. Rutina de atención a la interrupción producida por la UART . . . . .	48
2.4.3. Función para ensamblar los bloques de datos X-10 . . . . .	50
2.4.4. Rutina de atención a la interrupción externa . . . . .	51
2.4.5. Transmisión de datos . . . . .	54
2.4.6. Recepción del código de start . . . . .	55
2.4.7. Recepción del código de casa y el código numérico . . . . .	55
2.5. Conclusiones . . . . .	56

---

## ÍNDICE GENERAL

<b>3. El Módulo Ethernet</b>	<b>59</b>
3.1. La pila TCP/IP de Microchip . . . . .	59
3.1.1. La estructura del stack . . . . .	60
3.1.2. Configuración de los servicios del stack con TCPIPConfig.h . . . . .	63
3.1.3. El formato MPFS . . . . .	67
3.1.4. Desarrollo de una aplicación web . . . . .	70
3.2. Hardware necesario para la implementación del módulo . . . . .	75
3.2.1. Microcontrolador . . . . .	75
3.2.2. Controlador Ethernet externo . . . . .	76
3.3. Modificación del software para la implementación del proyecto . . . . .	80
3.3.1. Definición del perfil de hardware . . . . .	80
3.3.2. La función InitializeBoard . . . . .	82
3.3.3. Configuración del servidor HTTP . . . . .	83
3.3.4. La página web y el archivo CustomHTTPApp.c . . . . .	86
3.3.5. El archivo X-10.C . . . . .	88
3.4. Conclusiones . . . . .	91
<b>4. Consideraciones finales</b>	<b>93</b>
<b>A. Diagramas esquemáticos del proyecto</b>	<b>97</b>
<b>Bibliografía</b>	<b>97</b>

# Capítulo 1

## Introducción

El principal objetivo de este proyecto final consistió en mejorar la calidad de vida de personas con movilidad reducida. Para lograr este objetivo, se propuso el diseño de un dispositivo que permita comandar una red domótica desde una cama o una silla de ruedas. Mediante la red domótica se podrán controlar las luces, persianas, calefacción, o cualquier electrodoméstico que se encuentre en el hogar.

De esta forma se podrá automatizar la vivienda, y el usuario podrá prescindir en gran medida de la colaboración de otra persona para la realización de muchas tareas domésticas, obteniendo así mayor independencia y comodidad. Uno de los requisitos del proyecto, consistió en que la automatización debería llevarse a cabo de una manera sencilla y económica.

El dispositivo propuesto debería ser comandado por medio de una página web. Se podría acceder a la misma desde una computadora u otro dispositivo que sea capaz de manejar el protocolo TCP/IP. Esto permitirá controlar y monitorizar múltiples artefactos eléctricos existentes en el hogar mediante Internet.

En este capítulo se presentarán los conceptos básicos de la domótica. En la siguiente sección se explicará qué es la domótica, cuales son los diferentes tipos de elementos que pueden estar presentes en una red domótica, y de que maneras pueden estar conectados entre sí. En la Sección 1.2 se describirán los principales protocolos existentes en la actualidad. Luego, en la Sección 1.3. Se justificará la elección del protocolo domótico utilizado en este proyecto. Se finalizará el capítulo presentando el diagrama en bloques del proyecto, y dando una explicación básica del funcionamiento de cada uno de los bloques que lo componen.

### 1.1. Domótica. Definición

El término domótica proviene de la unión de las palabras *domus* (que significa casa en latín) y *tica* (que significa automática en griego). La domótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda.

Algunos de los elementos que se pueden controlar en una casa mediante un sistema de domótica son: la iluminación, la climatización, puertas, ventanas, persianas, electrodomésticos, etc. Por lo tanto, permite dar respuesta a los requerimientos que plantean los cambios sociales y las nuevas tendencias de nuestra forma de vida, facilitando el diseño de casas y hogares más confortables, personales, polifuncionales y flexibles.

La domótica contribuye a mejorar la calidad de vida del usuario:

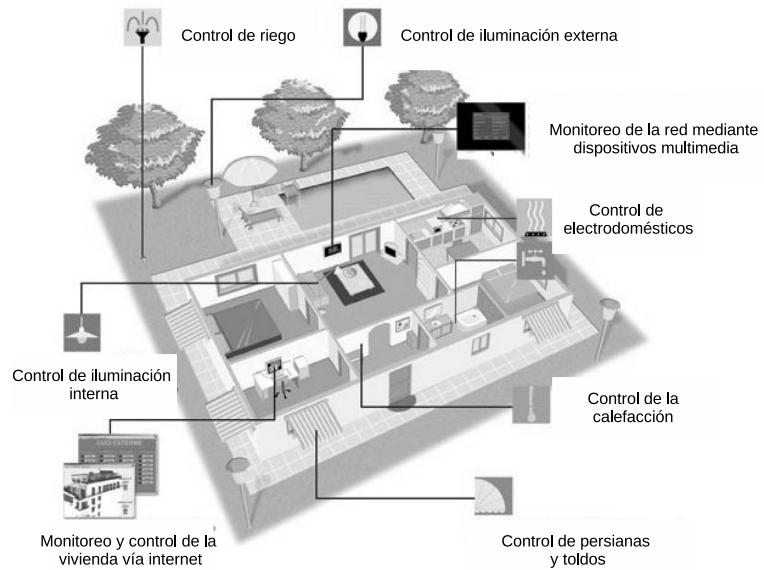


Figura 1.1: Prestaciones que brinda una red domótica

- **Facilitando el ahorro energético:** gestionando inteligentemente la iluminación, climatización, agua caliente sanitaria, el riego, los electrodomésticos, etc., aprovechando mejor los recursos naturales, utilizando las tarifas horarias de menor coste, y reduciendo de esta manera la factura energética. Además, mediante la monitorización de consumos, se obtiene la información necesaria para modificar los hábitos y aumentar el ahorro y la eficiencia.
- **Fomentando la accesibilidad:** facilita el manejo de los elementos del hogar a las personas con discapacidades de la forma que más se ajuste a sus necesidades, además de ofrecer servicios de tele-asistencia para aquellos que lo necesiten.
- **Aportando seguridad a personas, animales y bienes:** controles de intrusión y alarmas técnicas que permiten detectar incendios, fugas de gas o inundaciones de agua, etc.
- **Garantizando las comunicaciones:** recepción de avisos de anomalías e información del funcionamiento de equipos e instalaciones, gestión remota del hogar, etc.

El campo de la domótica ha evolucionado considerablemente en los últimos años, y en la actualidad ofrece una oferta más amplia. Hoy en día, la domótica aporta soluciones dirigidas a todo tipo de viviendas, se ofrecen más funcionalidades por menos dinero, más variedad de productos, y gracias a la evolución tecnológica, son más fáciles de usar y de instalar. En definitiva, la oferta es mejor y de mayor calidad, y su utilización es ahora más intuitiva para cualquier tipo de usuario. En la Figura 1.1 se representa una vivienda en la que se pueden ver las diferentes prestaciones que brinda una red domótica.

### 1.1.1. Caracterización de un Sistema Domótico

Para detallar las características de un sistema domótico, se comenzarán describiendo los tipos de elementos de los que está compuesto. Luego, en la Subsección **Tipo de Arquitectura**, se enumerarán las diferentes formas de distribuir dichos elementos (arquitectura del sistema). Por último, en la Subsección **Tipos de enlace** se presentarán las diferentes maneras con las que se puede realizar la comunicación entre los componentes del sistema.

#### Tipos de elementos

En un sistema domótico se pueden encontrar los siguientes elementos:

- **Controladores:** Son los dispositivos encargados de gestionar toda la información sobre los otros elementos del sistema. Dentro de este grupo se encuentran las interfaces necesarias para interactuar con los usuarios u otra aplicación. Las personas que utilizan el sistema pueden interactuar con el mismo de varias formas. La más tradicional es a través de teclados o de pantallas táctiles que necesitan el contacto con los dedos, pero esto no siempre es posible de operar por personas con movilidad restringida. Para solucionar este último inconveniente se desarrollaron otros tipos de controladores que detectan los movimientos del cuerpo y los utilizan para establecer el contacto con el controlador.

También existen dispositivos que poseen cámaras que registran señales corporales, o movimientos del ojo y se valen de los mismos para decodificar las órdenes. El usuario puede también proporcionar comandos a través de la succión o manejo oral de una vara electrónica conectada a un dispositivo receptor de comandos cuando la movilidad está totalmente restringida. Una modalidad relativamente nueva es el uso de comandos de voz, siendo su uso en algunos casos limitado a un cierto número de órdenes. En sistemas más desarrollados, es posible decir un texto completo cuando el sistema identifica la voz del usuario.

- **Sensores:** Los sensores son los elementos encargados de recoger la información de los diferentes parámetros de la red domótica, y enviarla al controlador para que actúe en consecuencia. Existe una gran variedad de sensores o detectores, siendo los más comúnmente utilizados: el termostato de ambiente, el detector de gas, los detectores de humo y calor, la sonda humedad y los sensores de presencia. Los sensores no se conectan por lo general a la red eléctrica sino que llevan una pila incorporada, con una duración de dos a cinco años. Esto supone una mayor flexibilidad respecto a otros dispositivos como los actuadores a la hora de ser introducidos en la vivienda domótica, ya que así se pueden instalar en cualquier lugar, aunque esté lejos de una toma de corriente.

- **Actuadores:** Son los dispositivos capaces de recibir una orden del controlador y realizar la acción pertinente. Entre los más comúnmente utilizados están: los contactores (o relés de actuación), los contactores para base de enchufe, las electroválvulas de corte de suministro (gas y agua), las válvulas para la zonificación de la calefacción por agua caliente, y sirenas o elementos zumbadores para el aviso de alarmas en curso. Estos dispositivos suelen estar distribuidos por toda la vivienda, y pueden admitir baterías. En algunos casos, el sensor y el actuador están integrados en el mismo dispositivo.

## Tipo de Arquitectura

Dependiendo de la distribución de los elementos domóticos y los controladores de estos, se pueden distinguir tres tipos de arquitecturas diferentes:

- **Arquitectura centralizada:** existe un único dispositivo controlador que recibe la información de múltiples sensores y se encarga de procesar los datos del entorno. Una vez procesada la información se generan las órdenes oportunas y se envían a los actuadores. Los sensores y actuadores se comunican solamente con el controlador y no entre sí. El controlador es el “corazón” de la vivienda, si falta todo deja de funcionar. En la Figura 1.2 se puede observar el diagrama de este tipo de arquitectura.

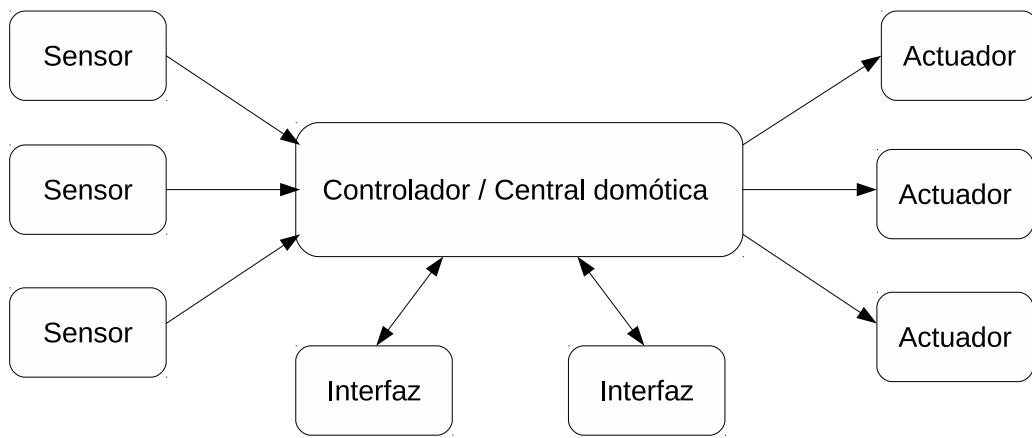


Figura 1.2: Arquitectura centralizada.

- **Arquitectura distribuida:** no existe ningún controlador centralizado. Cada elemento posee su propio controlador y es capaz de realizar las órdenes oportunas sobre los actuadores del sistema. Las instalaciones diseñadas con este tipo de arquitectura son fácilmente ampliables. El riesgo está distribuido, es decir, si un elemento deja de funcionar, el resto del sistema sigue operando. Sin embargo, el precio es una desventaja muy importante, aunque es cada vez más competitiva respecto a la arquitectura centralizada. En la Figura 1.3 se puede observar el diagrama de este tipo de arquitectura.

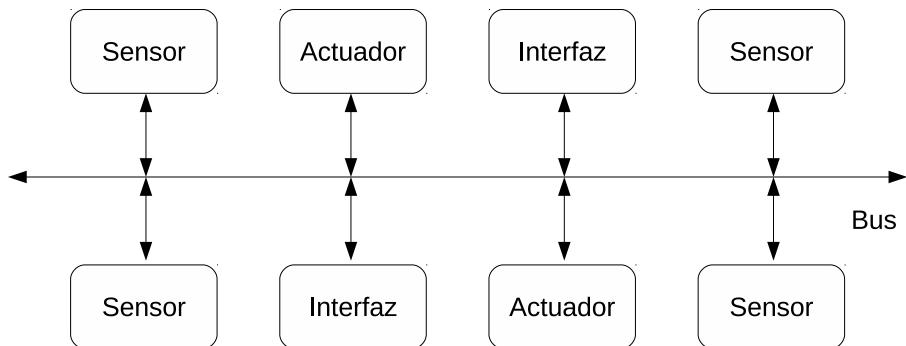


Figura 1.3: Arquitectura distribuida.

- **Arquitectura mixta:** se trata de un término medio entre las dos arquitecturas. En un sistema que utiliza este tipo de arquitectura, se puede disponer de un controlador central o varios controladores descentralizados. Los sensores y actuadores pueden también ser controladores como en un sistema con arquitectura distribuida. Existen sistemas que utilizan una arquitectura descentralizada pero que poseen módulos o islas de control centralizadas. En la Figura 1.4 se puede observar el diagrama de este tipo de arquitectura.

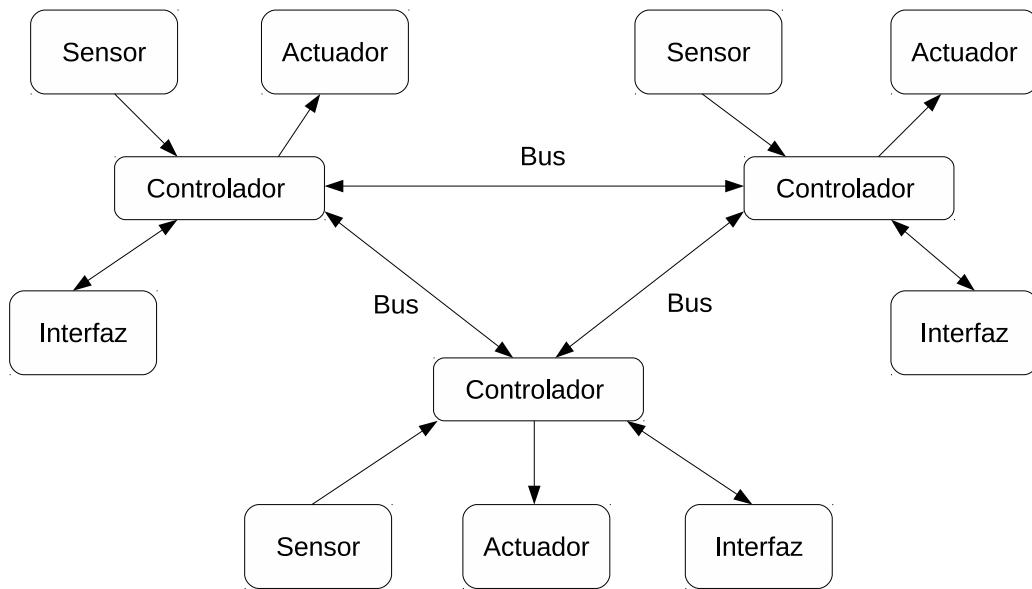


Figura 1.4: Arquitectura mixta.

## Tipos de enlaces

Dependiendo del tipo de enlaces que se utilicen en el sistema domótico se pueden clasificar en dos categorías:

- **Cableados:** Se trata de los sistemas en los que todos los sensores, actuadores y controladores están conectados entre sí mediante cables. Se puede catalogar a este tipo de sistemas en función del cableado que utilizan:

**Cableado exclusivo:** En los hogares normalmente no existe ningún otro cableado que no sea el de la red eléctrica o el telefónico. Un sistema con cableado exclusivo es aquel que utiliza una instalación de cableado exclusivamente para la red domótica. Esto implica cablear toda la vivienda para añadir un nuevo bus de comunicaciones que permita el envío de datos entre los dispositivos domóticos. Resulta práctico cuando se aplica a viviendas nuevas mientras se construyen, ya que no supone un gran coste adicional.

**Cableado compartido:** Cuando no hay posibilidad de agregar una nueva red de cables a la vivienda se puede optar por esta solución, que consiste en utilizar un cableado ya existente y compartirlo. Concretamente se suele utilizar la línea eléctrica. La tecnología se llama *Power Line Carrier* (PLC), funciona mediante la modulación de una onda portadora cuya frecuencia oscila entre los 20 y 200 KHz inyectada directamente en el cableado eléctrico.

- **Inalámbricos:** Cuando no es posible cablear la vivienda y tampoco utilizar la tecnología PLC se puede optar por utilizar tecnologías inalámbricas. Estas tecnologías permiten que el dispositivo domótico no necesite estar en un lugar fijo, ya que puede comunicarse con el sistema desde cualquier lugar dentro del alcance del receptor. Algunas de las mas utilizadas son:

**RF:** Se denomina así a todo canal de transmisión inalámbrico. En este grupo entrarían todos los enlaces cuya tecnología fue desarrollada por un fabricante y no se encuentra bajo ningún estándar.

**ZigBee:** Es un sistema ideal para redes domóticas, específicamente diseñado para reemplazar la proliferación de sensores/actuadores individuales. ZigBee fue creado para cubrir la necesidad del mercado de un sistema a bajo coste. Es un estándar de bajo consumo, seguro y fiable, diseñado para redes inalámbricas que utilizan pequeños paquetes de información.,

**Bluetooth:** Es una especificación industrial para redes inalámbricas de área personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz.

## 1.2. Principales protocolos existentes en el mercado

Un protocolo es un método establecido para intercambiar datos en un sistema domótico. Es el método por el cual dos elementos del sistema se comunican, una especificación que describe la manera en la que viajan los datos, la forma de sincronización, la estructura del mensaje, el tipo de comprobación de errores que se utilizará, etc. De acuerdo a su estandarización, los protocolos pueden ser:

- **Estándar:** Un protocolo estándar es un conjunto de reglas que han sido ampliamente usados e implementados por diversos fabricantes, usuarios y organismos oficiales (e.g. IEEE, ISO, ITU). Idealmente, un protocolo estándar debe permitir a los dispositivos comunicarse entre sí, aun cuando estos sean de diferentes fabricantes.

- **Propietarios:** Se trata de protocolos de comunicaciones cuya propiedad es de una sola organización o individuo. Son desarrollados por dichas organizaciones para la fabricación de productos propios capaces de comunicarse entre sí. La desventaja de estos es que por lo general no están disponibles al público.

Existen diversas tecnologías y protocolos que se utilizan actualmente para el desarrollo de sistemas domóticos. Entre los protocolos estándar más importantes y empleados se encuentran EIB, Konnex, y X-10.

### 1.2.1. Protocolo EIB (Bus de Instalación Eléctrica)

La EIBA(*European Installation Bus Association*) creó y registro su sistema llamado EIB, el cual es empleado para controlar servicios eléctricos de residencias, viviendas, edificios de negocios, etc. Las instalaciones eléctricas con este protocolo son casi como las instalaciones tradicionales, sólo que el usuario utiliza interruptores especiales del sistema EIB para encender y apagar las luces de su hogar. En la Figura 1.5 se observa el interruptor superior del sistema, con ocho pulsadores, con la misma dimensión del interruptor convencional.

Cada interruptor del sistema EIB permite controlar 16 actuadores. Al pulsar de diferentes formas los interruptores se puede regular la luminosidad e intensidad de las lámparas, subir o bajar las persianas, etc.



Figura 1.5: Interruptor EIB

En la Figura 1.6 se puede ver el esquema de conexión de un sistema EIB. En ella se observan los diferentes actuadores y sensores que componen el sistema, y como están interconectados entre sí. Se nota claramente que el tipo de enlace de este protocolo es el cableado exclusivo (línea bus). Los componentes conectados al bus intercambian información, la transmisión de dicha información se lleva a cabo en serie y según un protocolo de bus.

Los datos a transmitir se empaquetan en datagramas, que son transportados a través del bus, desde los sensores hacia uno o más actuadores. Cada receptor debe confirmar la correcta recepción del datagrama. Si no hay confirmación se repite la transmisión hasta tres veces. Si aún así no se confirma la recepción, se interrumpe el proceso de emisión y el error se almacena en un *buffer* de la memoria del emisor.

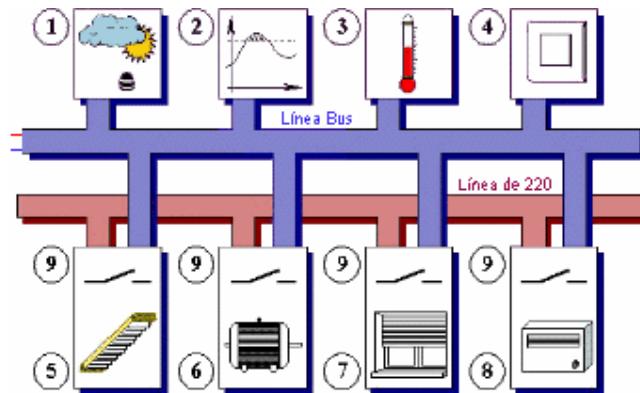


Figura 1.6: Esquema de conexión de un sistema EIB

Los componentes del bus se alimentan con una tensión de 24V. Los datagramas se modulan en base a esta tensión, de modo que un “cero lógico” se transmite como un impulso y la falta de impulso se interpreta como un “uno lógico”. Cada dato del datagrama se transmite asincrónicamente. La transmisión se sincroniza mediante bits de inicio y parada.

Para la correcta recepción de los datagramas dentro del protocolo de transmisión, se ha de asignar a cada componente del bus una dirección propia que lo identifique de manera inequívoca, de modo que desaparezca la posibilidad de confusión con el resto de componentes. Cada dispositivo tiene dos direcciones de 16 bits, la física y la lógica. Ambas son asignadas al preparar la instalación, tienen funciones diferentes y son mutuamente excluyentes (se utiliza una u otra).

El empleo del bus como medio físico colectivo de comunicación para transmisiones asíncronas debe regularse convenientemente. Para ello, el EIB utiliza el protocolo CSMA/CA, que garantiza un funcionamiento del bus libre de colisiones, sin reducir por ello la capacidad de transmisión de los datos del bus. Algunas de las características del protocolo son las siguientes:

- Todos los componentes del bus están sensando el bus en busca de instrucciones, pero solo las ejecutan cuando la dirección con la que están programados coincide con la transmitida.
- Si un componente necesita emitir cierta información, debe comprobar primero que no hay otro elemento emitiendo. Si lo hubiera, debe esperar hasta que finalice. Esta forma de actuar (que, en inglés, se denomina *carrier sense*), da lugar a las dos primeras letras del acrónimo con que se nombra este protocolo de transmisión.
- Si el bus está libre, cualquier componente puede, en principio, comenzar con el proceso de emisión (en inglés, *multiple access*).
- Si dos componentes comienzan a emitir en el mismo instante, se impone el componente de mayor prioridad (*collision avoidance*), mientras que el otro se detiene para retomar su emisión en un instante posterior.
- En el caso de que ambos componentes tengan igual prioridad, se impone el de menor dirección física.

En un sistema EIB pueden convivir módulos de diversos fabricantes y comunicarse entre sí sin ningún tipo de problemas. Esto se debe a que se trata de un protocolo estándar. La normativa de este sistema provee una compatibilidad total entre productos homologados, esto facilita las posteriores actualizaciones. El sistema puede ser ampliado con un máximo de quince áreas. Además trabaja de manera descentralizada y su estructura puede ser lineal, de estrella o con ramificaciones, sin necesidad de un control central.

### 1.2.2. Protocolo Konnex (KNX)

Es una tecnología que resulta de la iniciativa de los sistemas de control europeo Batibus , EIB y EHS, con el objetivo de crear un único estándar europeo para la automatización HBES (*Home and building electronic systems*) basado en sistemas abiertos. Esta decisión se originó porque se necesitaba dar solución a muchos inconvenientes que presentaban individualmente los estándares.

Se buscaba organizar un único estándar para Domótica e Inmótica que cubra todas las exigencias de las instalaciones profesionales del ámbito europeo. Esto ayudó a desarrollar la presencia de estos buses en áreas como la climatización y la seguridad.

Konnex puede ser usado para todas las aplicaciones en el control de casas y edificios desde iluminación, ventanas, control de seguridad y alarmas, calefacción, ventilación, hasta aplicaciones para el hogar, audio y video. El estándar KNX incorpora tres modos de configuración distintos:

- El Modo Sistema (*System mode*): Este mecanismo de configuración está enfocado para realizar funciones de control sofisticadas en edificios. Son instalados y configurados por profesionales con ayuda de software especialmente diseñado para este propósito. La configuración *System mode* ofrece el más

alto grado de flexibilidad en funcionalidad.

- El Modo fácil (*Easy mode*): Los dispositivos están programados en fábrica para realizar una función específica. Aún así pueden ser reconfigurados algunos parámetros y enlaces principalmente de comunicación. Este mecanismo de configuración está destinado a instaladores con una formación básica y provee una rápida evolución del aprendizaje pero con funciones limitadas, comparadas con el S-mode.

- El Modo automático (*Automatic mode*): Este mecanismo de configuración está desarrollado especialmente para aplicaciones de usuario final. Los dispositivos *Automatic mode* disponen de mecanismos de configuración automática, que adaptan sus enlaces de comunicación al resto de dispositivos en la red.

Además de los 3 modos de configuración, el estándar KNX se puede utilizar con los tres tipos de enlace, cableado exclusivo, cableado compartido o inalámbrico. A su vez cada uno de éstos puede ser usado con uno o más modos de configuración. Si se utiliza un cableado exclusivo los elementos de la red domótica deben estar conectados mediante un cable par trenzado. Con este tipo de enlace (que ha sido tomado de EIB) se pueden lograr velocidades de 9600 bits/segundo.

El tipo de enlace de cableado compartido que admite KNX está basado en *Power Line Carrier* (PLC). Con él se logran velocidades de 1200 bits/segundo. Los productos certificados EIB y KNX operan y se comunican los unos con los otros bajo la misma red de distribución eléctrica.

Existen dispositivos KNX RF capaces de funcionar de forma inalámbrica. Estos emplean señales de radio para enviar datagramas KNX. Dichos datagramas son transmitidos en la banda de frecuencia 868 MHz (dispositivos de corto alcance), con una potencia máxima de 25 mW y una velocidad de transmisión de 16.384 kBit/segundo. Los dispositivos KNX RF pueden ser tanto unidireccionales como bidireccionales, y se caracterizan por su bajo nivel de consumo energético. Están destinados a pequeñas y medianas instalaciones que sólo necesitan transmisores en casos excepcionales.

La especificación KNXnet/IP permite que los datagramas KNX puedan ser también encapsulados en datagramas IP. De esta forma, podemos enviar dichos telegramas KNX por redes LAN e Internet. En la actualidad KNX es una de las tecnologías más utilizadas en el mundo.

### 1.2.3. Protocolo X-10

X-10 es uno de los protocolos más antiguos que se están usando en aplicaciones domóticas. Se trata de un protocolo estándar que fue desarrollado entre 1976 y 1978 por los ingenieros de Pico Electronics Ltd, en Escocia. El objetivo principal de la empresa era transmitir datos por las líneas de baja tensión con costos muy bajos. Con este protocolo se puede controlar cualquier dispositivo a través de un cableado compartido, para lo cual se utiliza la red eléctrica doméstica (120V o 220V y 60Hz o 50Hz).

Se puede utilizar más de un controlador para manejar a los actuadores de la red. Por lo tanto, se puede decir que se trata de una arquitectura distribuida. Debido a su madurez, y a la tecnología empleada, los productos X-10 tienen un precio muy competitivo, y esto los convierte en líderes en el mercado norteamericano residencial y de pequeñas empresas.

Con un simple método de direccionamiento, se pueden manejar un total de 256 dispositivos en la red. El protocolo soporta 16 grupos de direcciones denominados códigos de casa (desde la A a la P), y otras 16 direcciones para cada código de casa, denominadas códigos de unidad. Las señales enviadas por el controlador son recibidas en todos los módulos, pero sólo el módulo que posee la dirección indicada en el mensaje realizará alguna operación. El mensaje completo tiene 48 bits.

En su primera versión tan sólo existían seis operaciones: encender, apagar, aumentar, disminuir,

todo apagado y todo encendido. Posteriormente, los códigos de operación fueron extendidos a 256 con una cabecera especial. La cantidad de información que transporta un mensaje puede ser mayor de 48 bits, si es usado el código de datos extendidos en la cabecera de control del mensaje. La transmisión está sincronizada con los cruces por cero de la red eléctrica. El protocolo X-10 utiliza una modulación muy sencilla, comparado con las que usan otros protocolos de control por ondas portadoras.



Figura 1.7: Controlador X-10

En la Figura 1.7 se puede ver un módulo controlador inalámbrico X-10, el mismo posee un transmisor de radio frecuencia en el que se encuentran los comandos que se utilizan para dar las órdenes a los actuadores, y un receptor de radio frecuencia que va conectado a la red eléctrica (*transceiver*). Este decodifica las órdenes recibidas y envía el mensaje X-10.

El *transceiver* X-10 está pendiente de los cruces por cero de la onda sinusoidal de 50 Hz típica de la alimentación eléctrica (60 Hz en EEUU) para insertar un instante después una ráfaga muy corta de señal en una frecuencia fija.

En la Figura 1.8 se puede ver un actuador X10, estos dispositivos vienen preparados para enchufar a un toma corriente de la vivienda. Además poseen un toma corriente hembra en el que se conecta el dispositivo a controlar. Vienen dotados de dos pequeños conmutadores giratorios (uno con 16 letras y el otro con 16 números) que permiten asignarle una dirección de las 256 posibles. Cuando la dirección que recibe por la red eléctrica coincide con la preasignada el dispositivo ejecuta la orden.

En una misma instalación puede haber varios actuadores configurados con la misma dirección, de modo que todos realizarán la función preasignada cuando un transmisor envíe una trama con esa dirección (*multicast*). Los módulos actuadores mas modernos son bidireccionales, por lo tanto, tienen la capacidad de responder y confirmar la correcta realización de una orden. Esta característica puede ser muy útil cuando el sistema X-10 está conectado a un programa de PC que muestre los estados en que se encuentra la instalación domótica de la vivienda.



Figura 1.8: Actuador X-10

### 1.3. Elección del protocolo domótico más adecuado para la realización del proyecto

Para realizar la elección del protocolo domótico más adecuado para este proyecto, se evaluaron todas las opciones teniendo en cuenta los objetivos descritos en la Sección 1. Se debía optar por un protocolo con el cual se pudiera automatizar una vivienda en poco tiempo, de forma económica y sencilla.

Después de estudiar detalladamente los protocolos enumerados en la Subsección 1.2, se decidió utilizar X-10 para realizar el proyecto. La principal ventaja de este protocolo es que utiliza cableado compartido. Por lo tanto, no se debe agregar ninguna red de cables para la automatización de una vivienda. Esto hace a X-10 sumamente versátil y económico. Cualquier toma corriente de la red eléctrica se puede convertir en un toma de la red domótica.

Existe una gran cantidad de información acerca de los principios de funcionamiento de X-10, e infinidad de proyectos realizados. Esto constituye una ventaja por sobre otros protocolos que también utilizan cableado compartido como EIB, Konnex.

Otra de las principales ventajas que ofrece X-10 es su arquitectura abierta. Lo que ha permitido que los precios bajen, y que la oferta de módulos sea enorme. También se puede destacar la facilidad de instalación de los módulos: solo basta con enchufarlos a la red eléctrica y configurarlos con la dirección que se desee. Se puede automatizar cualquier vivienda en un tiempo muy bajo y sin necesidad de un gran conocimiento de electricidad o electrónica. Por otra parte, si el usuario desea cambiar de vivienda este tipo sistema se puede trasladar fácilmente. Esas son las razones por las cuales se concluyó que X-10 es el protocolo más adecuado para este proyecto, y es el que mejor se adapta mejor a los objetivos planteados.

## 1.4. Modelo del sistema a desarrollar

Como se mencionó anteriormente, el objetivo de este proyecto es diseñar un dispositivo para controlar los diferentes artefactos eléctricos de una vivienda mediante una red domótica. También se mencionó que el dispositivo será comandado por intermedio de una página web. Por otra parte, se concluyó que el protocolo más adecuado para la red domótica es X-10. Con esta información se puede plantear el modelo del sistema que se desea desarrollar, que se puede ver en la Figura 1.9.

En primer lugar se necesita un dispositivo que sea capaz de alojar una página web y decodificar las órdenes del usuario, para luego enviarlas a la red domótica. A este dispositivo se lo llamará Módulo Ethernet. En segundo lugar, se necesita otro dispositivo<sup>1</sup> que sea capaz de interactuar con la red eléctrica. Será el encargado de enviar los mensajes en el formato X-10 a los actuadores. Además, este módulo deberá recibir la confirmación que envían dichos actuadores para indicar que una orden fue recibida correctamente.

Por otra parte, el módulo transceptor X-10 podrá utilizarse como un actuador bidireccional X-10. Cuando se configure de esta forma será capaz de recibir una orden mediante la red eléctrica, ejecutarla y enviar la confirmación al transmisor. Se denominará interfaz 802.3/X-10 al conjunto de los dos módulos.

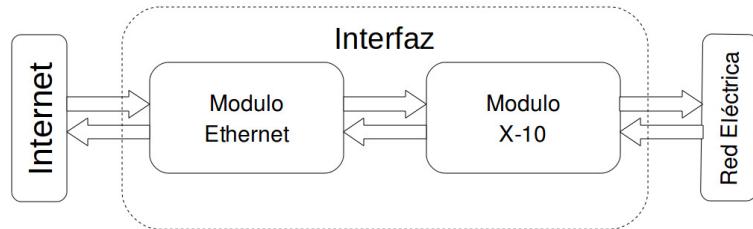


Figura 1.9: Diagrama en bloques del sistema propuesto

## 1.5. Conclusiones

En este capítulo se realizó una introducción en la que se plantearon los objetivos del proyecto, y los requisitos que debe tener el sistema a desarrollar.

Primeramente en la Sección 1.1 se presentaron los conceptos básicos de la domótica, y los tipos de elementos que componen una red domótica (actuadores, sensores y controladores). Se explicaron los tipos de arquitectura con los que se puede relacionar dichos elementos: arquitectura centralizada, distribuida y mixta. También se clasificaron los protocolos teniendo en cuenta el tipo de enlace con el que se conectan sus elementos.

Luego en la Sección 1.2 se describieron los principales protocolos existentes en la actualidad, teniendo en cuenta el tipo de arquitectura, y los tipos de enlace que utilizan:

- **EIB:** Utiliza una arquitectura distribuida y cableado exclusivo.
- **Konnex:** Utiliza una arquitectura mixta, y los tres tipos de enlace, cableado exclusivo, cableado compartido o inalámbrico.

<sup>1</sup>Al que se llamará Módulo transceptor X-10

- **X-10:** Utiliza una arquitectura distribuida, y cableado compartido.

En la Sección 1.3, se justificó la elección del protocolo X-10 describiendo sus principales ventajas. Entre las que se encuentran: la utilización de cableado compartido que lo hace sumamente versátil y económico, la gran cantidad de información acerca de sus principios de funcionamiento, y su arquitectura abierta.

Por último, se planteó el modelo del sistema a desarrollar, que posee dos piezas fundamentales: el módulo X-10 y el módulo Ethernet. El primero Será el encargado de enviar los mensajes en el formato X-10 a los actuadores, y deberá recibir la confirmación que envían dichos actuadores para indicar que una orden fue recibida correctamente. El módulo Ethernet tendrá alojada una página web y decodificará las órdenes del usuario, para luego enviarlas a la red domótica.

En el siguiente capítulo se explicarán en detalle los principios de funcionamiento del módulo X-10 que es el encargado de la interacción con la red eléctrica.



## Capítulo 2

# El módulo transceptor X-10

El módulo transceptor de X-10 puede ser utilizado para cumplir dos funciones distintas. Por un lado se lo puede configurar para formar parte de la interfaz 802.3/X-10. Cuando esto sucede, debe recibir los datos del módulo Ethernet y enviarlos por la red eléctrica. Por último, debe recibir la confirmación (*Hail Acknowledge*), enviada por el receptor de la orden, que indica que la transmisión fue realizada correctamente.

Por otro lado, el módulo puede utilizarse como un actuador bidireccional de X-10. En este caso se encarga de decodificar la orden recibida, realizar la acción, y enviar la confirmación al transmisor. Para explicar el funcionamiento del módulo, se comenzará con una descripción en detalle del protocolo X-10. En la Sección 2.2 se presentará el hardware del transceptor, prestando especial atención al microcontrolador (Sección 2.3) que es la pieza más importante de esta parte de la interfaz. Se finalizará el Capítulo con la Sección 2.4, en la cual se brinda una explicación en detalle del software asociado.

### 2.1. Principios de funcionamiento de X-10

En esta sección se explicará en detalle la forma en la que se transmiten los datos en el protocolo X-10. Una de las principales características de X-10 es que utiliza cableado compartido, las señales de control son enviadas por la línea eléctrica y se suman a la señal de red. Mientras que los módulos se sincronizan con los cruces por cero de la señal de 220V.

Un “uno lógico” es representado por una señal portadora de 120KHz enviada por un lapso de un milisegundo, esta señal debe comenzar a transmitirse luego del cruce por cero de la señal de 220V. Un “cero lógico” se representa por la ausencia de portadora luego del cruce por cero de la señal de 220V. En un sistema de distribución trifásico, el “uno lógico” deberá ser transmitido tres veces para coincidir con los cruces por cero de las tres fases. En la parte superior de la Figura 2.1, se puede ver un gráfico de la señal de línea eléctrica con la portadora sumada. Mientras que en la parte inferior de la figura, se observa a la señal mostrada anteriormente luego de pasar por un filtro pasa altos. En este caso se está enviando un “uno lógico” en un sistema de tres fases.

Un mensaje completo en X-10 está compuesto por un código de comienzo (1110) o *Start Code*, seguido por un código de casa (que se representa con una letra) y un código numérico, con el que se identifica a cada módulo, o a las diferentes funciones (encender, apagar, aumento de intensidad, etc.). En la Figura 2.2 se puede observar la transmisión del *Start Code* (1110).

Tanto el código de casa como el código numérico se envían en formato complementario. Esto significa que para representar un bit de cada una de estas palabras se utilizará un ciclo completo de la

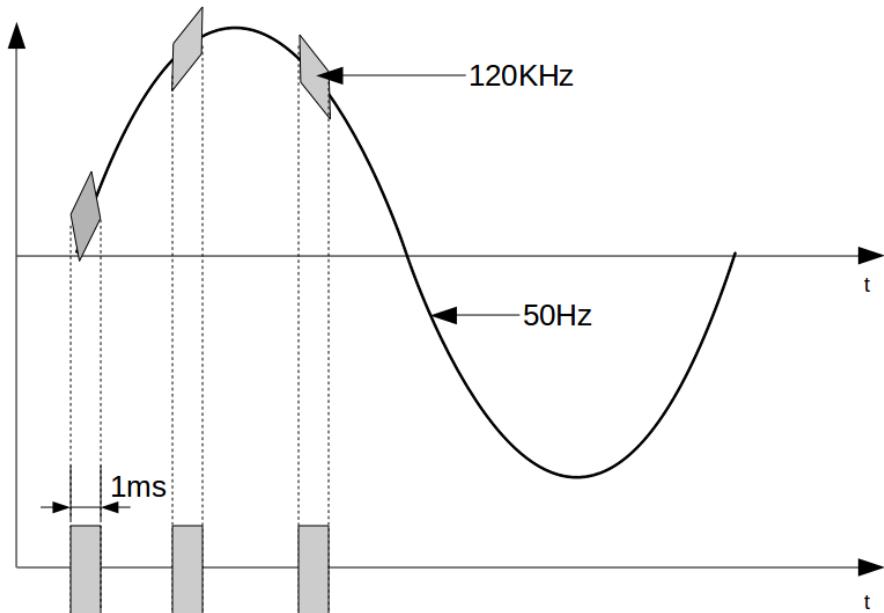


Figura 2.1: Señal de línea eléctrica con portadora X-10, y señal de la portadora filtrada.

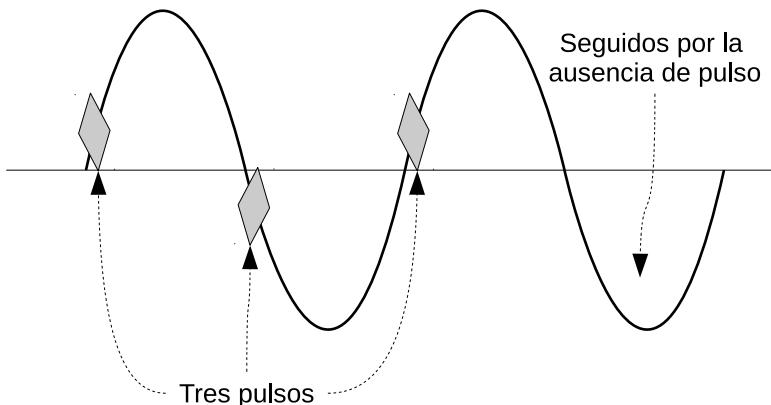


Figura 2.2: Transmisión del código de inicio

señal de red. Si se desea representar un uno, se deberá enviar un “uno lógico” en el primer semi-ciclo y un “cero lógico” en el segundo. Si se desea enviar un cero, se deberá hacer lo contrario, un “cero lógico” en el primer semi-ciclo y un “uno lógico” en el segundo. Para aclarar lo expuesto anteriormente se muestra la Figura 2.3 en la que se representa un código de casa, por otra parte, en la Figura 2.4 se muestran todos los códigos de casa disponibles.

En la Figura 2.5 se muestran los códigos numéricos. Las primeras 15 palabras se utilizan para indicar la dirección de cada uno de los módulos que conforman la red. Las últimas 15 se utilizan para las órdenes. Como se puede apreciar el bit menos significativo (que se denomina sufijo) de las direcciones es siempre “0”, y es “1” en todas las palabras que se utilizan para las órdenes.

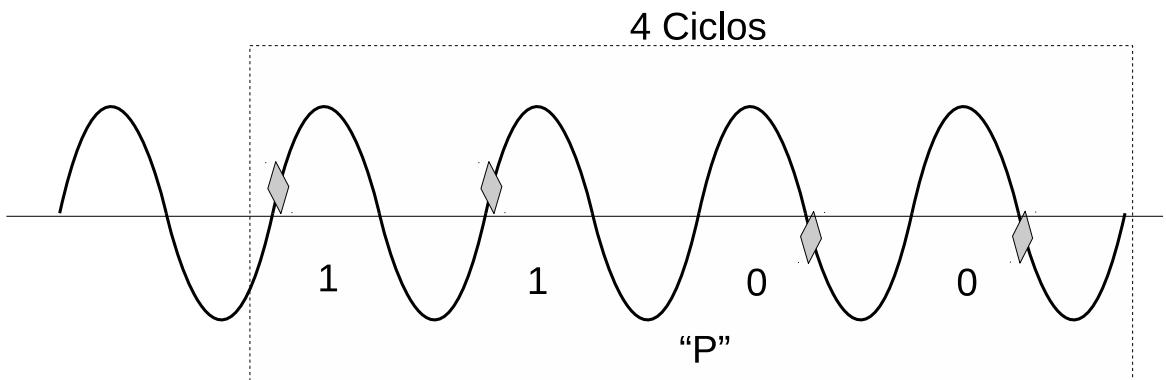


Figura 2.3: Representación de un código de casa

	H1	H2	H4	H8
A	0	1	1	0
B	1	1	1	0
C	0	0	1	0
D	1	0	1	0
E	0	0	0	1
F	1	0	0	1
G	0	1	0	1
H	1	1	0	1
I	0	1	1	1
J	1	1	1	1
K	0	0	1	1
L	1	0	1	0
M	0	0	0	0
N	1	0	0	0
O	0	1	0	0
P	1	1	0	0

Figura 2.4: Códigos de casa disponibles

Para realizar una transmisión, se debe comenzar direccionando al módulo para el cual va dirigida la orden. En primer lugar, se debe enviar el código de comienzo (1110)<sup>1</sup>, luego se debe enviar la dirección de la casa, y por último la dirección de la unidad a la que se le quiere dar la orden.

Debido al medio de transmisión utilizado los diseñadores de X-10 decidieron transmitir dos veces cada uno de estos bloques de información para que el sistema fuese más robusto. En la Figura 2.6 se muestra como están conformados un par de bloques de datos X-10.

Una vez que el receptor ha procesado sus datos de dirección, está listo para recibir una orden de comando. Para esto, se debe enviar un segundo par de bloques de datos, cada uno de estos bloques está compuesto por el código de comienzo, el código de la casa y la orden. Cada par de bloques de

<sup>1</sup>Es la única parte del mensaje que no se envía de forma complementaria.

	D4	D3	D2	D1	D0
1	0	1	1	0	0
2	1	1	1	0	0
3	0	0	1	0	0
4	1	0	1	0	0
5	0	0	0	1	0
6	1	0	0	1	0
7	0	1	0	1	0
8	1	1	0	1	0
9	0	1	1	1	0
10	1	1	1	1	0
11	0	0	1	1	0
12	1	0	1	0	0
13	0	0	0	0	0
14	1	0	0	0	0
15	0	1	0	0	0
16	1	1	0	0	0
All Units Off	0	0	0	1	1
All Lights On	0	0	0	0	1
On	0	0	1	1	1
Off	0	0	1	0	1
Dim	0	1	0	0	1
Bright	0	1	0	1	1
All Lights Off	0	1	1	0	1
Extended Code	0	1	1	1	1
Hail Request	0	0	0	0	1
Hail Acknowledge	1	0	0	1	1
Pre-Set Dim	1	0	1	1	1
Extended Data(analog)	1	1	0	1	1
Status = on	1	1	0	1	1
Status = off	1	1	1	0	1
Status Request	1	1	1	1	1

Figura 2.5: Códigos numéricos

información debe estar separado del anterior por seis cruces por cero<sup>2</sup>. Estos tres ciclos de margen son necesarios para que el receptor mueva los datos de sus registros. En la Figura 2.7 se representa la separación que existe entre los dos pares de bloques de datos.

Haciendo una síntesis de lo explicado anteriormente se puede decir que cada bloque de información está conformado por once ciclos de la línea eléctrica. Los primeros dos ciclos representan un código de comienzo, los siguientes cuatro representan el código de casa, y los últimos cinco ciclos representan una dirección de unidad, o una orden (código numérico).

Este bloque completo (código de comienzo, código de casa, código numérico) debe ser transmitido dos veces. Cada par de bloques de datos tienen que estar separados por tres ciclos de la línea eléctrica. Para asegurar la compatibilidad con cualquier módulo X-10, se debe tener en cuenta que el máximo retraso desde el paso por cero al principio de la transmisión debe ser de  $300\mu s$ . Los receptores al igual que los transmisores detectan cada paso por cero y buscan la señal de 120 KHz durante un período de 1 ms.

<sup>2</sup>Excepto cuando se envían las órdenes Bright y dim que deben ser transmitidas en forma continua.

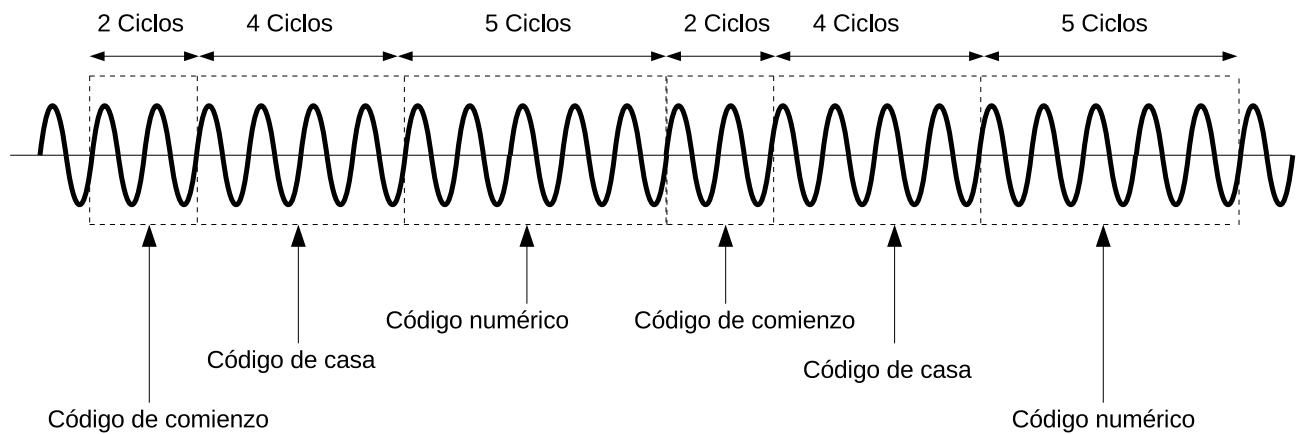


Figura 2.6: Conformación de un par de bloques de datos X-10

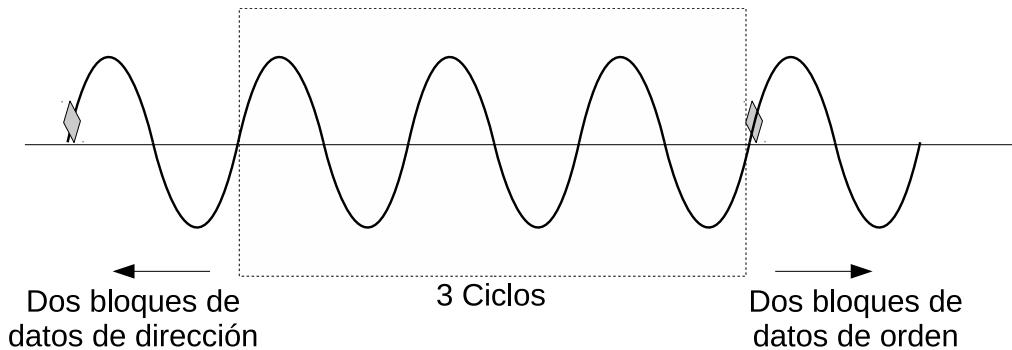


Figura 2.7: Separación existente entre dos pares de bloques de datos

## 2.2. Hardware necesario para la implementación del módulo

El módulo transceptor de X-10 puede ser utilizado para cumplir dos funciones distintas. Se lo puede configurar para formar parte de la interfaz 802.3/X-10, o puede utilizarse como un actuador bidireccional de X-10. Cuando el módulo transceptor de X-10 forma parte de la interfaz 802.3/X-10, se encarga de recibir los datos del módulo Ethernet mediante una comunicación serie. Luego transforma los datos recibidos al formato X-10 y los envía por la red eléctrica. Por último, recibe la confirmación (*Hail Acknowledge*), enviada por el receptor de la orden que indica que los paquetes transmitidos llegaron correctamente, si esto no ocurre debe enviar los datos nuevamente.

Cuando el módulo transceptor de X-10 se configura como actuador bidireccional, se encarga de decodificar la orden recibida, realizar la acción (encender o apagar su salida de 220V), y enviar la confirmación al transmisor.

Para poder recibir las señales provenientes de la línea eléctrica, y transmitir información utilizando dicha línea, el módulo deberá contar con una serie de circuitos. En la Figura 2.8 se muestra un diagrama en bloques del mismo.

La pieza más importante del módulo es el microcontrolador. Este realiza una gran cantidad de

tareas tales como decodificar los mensajes provenientes de la red eléctrica y del módulo Ethernet. Además, se encarga de ensamblar las palabras a transmitir, generar la portadora y los retardos. Para realizar todas estas tareas, el microcontrolador se debe sincronizar con los cruces por cero de la red eléctrica, mediante un detector de cruce por cero.

La etapa de entrada/salida se encarga de interactuar con la red eléctrica recibiendo y transmitiendo las señales de X-10. Debido a que las señales provenientes de la red eléctrica poseen una escasa amplitud y un nivel de ruido muy alto, se las debe amplificar y filtrar. Para esto se utiliza el filtro pasobandas y el amplificador sintonizado.

Luego, se debe demodular la señal X-10 que ingresa al módulo con un detector de envolvente. De esta forma, el microcontrolador puede reconocer los unos y ceros lógicos del mensaje.

Cuando se desea transmitir una señal por la red eléctrica se debe amplificar la misma a un nivel de tensión adecuado. Para ello se utiliza el amplificador de salida. En esta sección se explicará en detalle el funcionamiento de cada uno de los circuitos citados anteriormente.

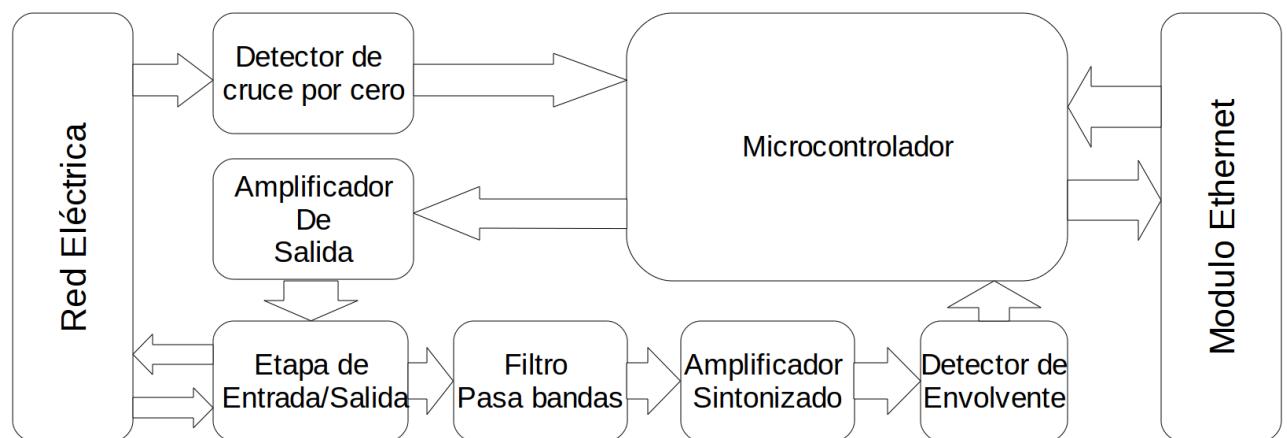


Figura 2.8: Diagrama en bloques del módulo transceptor X-10

### 2.2.1. Detector de cruce por cero

Este es el circuito que se encarga de sincronizar al módulo con los cruces por cero de la señal de 220V. Para lograr esto genera una señal cuadrada cuyos flancos (positivos y negativos) coinciden con los cruces por cero. Como se verá en las secciones siguientes, se utilizará la señal de salida de este detector para llamar a una subrutina del microcontrolador usando una interrupción externa cada vez que se produzca un cruce por cero de la señal de red.

En la nota de aplicación AN236 de Microchip [Bur2] se propone el circuito que se muestra en la Figura 2.9. Como se puede ver, el microcontrolador es conectado a la red eléctrica mediante un resistor, cuya función es la de limitar la corriente entrante. Cada entrada del microcontrolador posee dos diodos internos de protección, la señal que ingresa al circuito integrado es recortada por los diodos si supera a la tensión de alimentación ( $V_{dd}$ ) de los mismos, o es inferior a la tensión de referencia ( $V_{ss}$ ).

Cuando la señal de la red eléctrica ingresa al microcontrolador es recortada, y se genera una señal cuadrada de amplitud  $V_{dd}-V_{ss}$ . Los flancos de la señal cuadrada coinciden con los cruces por cero de

la señal de 220V<sup>3</sup>, y con cada uno de estos flancos se provoca una interrupción del programa principal del microcontrolador y se atiende a una subrutina. En la Figura 2.10 se puede ver una representación de la señal sinusoidal de la red eléctrica y la señal cuadrada que ingresa al microcontrolador luego de pasar por los diodos. Esta forma de realizar el detector de cruce por cero es muy poco recomendable, debido a que no existe una aislación entre el módulo y la red eléctrica. No se lo debe utilizar, porque pone en peligro a los componentes del módulo y a las personas que lo utilizan.

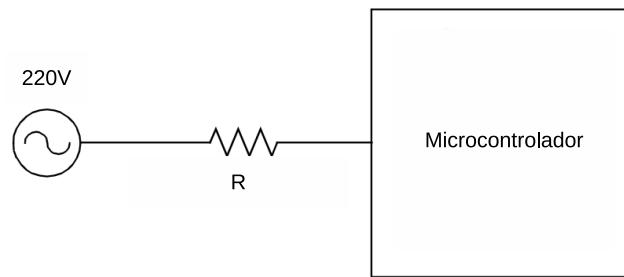


Figura 2.9: Circuito propuesto en la Nota de aplicación AN236 de Microchip

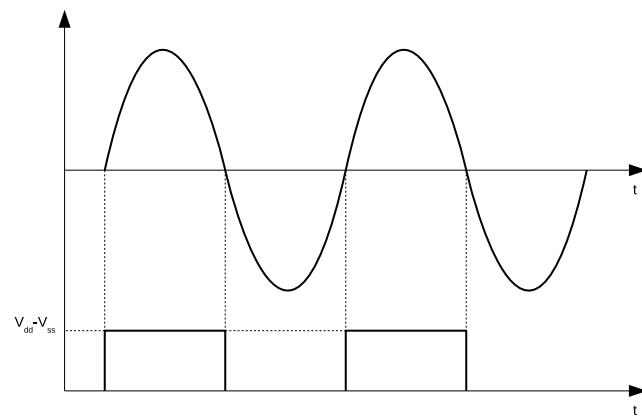


Figura 2.10: Señal sinusoidal de red y señal cuadrada que ingresa al microcontrolador

Para proteger a los componentes sensibles del módulo (microcontrolador y otros componentes ), y principalmente para garantizar la seguridad del usuario el detector de cruce por cero debe poder sensar la señal de red aislado de la misma. Para ello se utilizó un opto-acoplador 4N26[Sem99]. La Figura 2.11 muestra el circuito del detector.

El opto-acoplador 4N26 está compuesto por un diodo emisor de luz y un foto-transistor. Cuando circula una corriente por el foto-diodo este emite luz, que a su vez excita la base del foto-transistor haciendo circular una corriente por la misma.

De esta manera, se logra un aislamiento casi perfecto (aproximadamente de 3750 Volts) entre la

<sup>3</sup>Se puede suponer que esto se produce cuando hay un cruce por cero de la señal de 220V debido a que  $V_{dd} - V_{ss} \ll 220V$

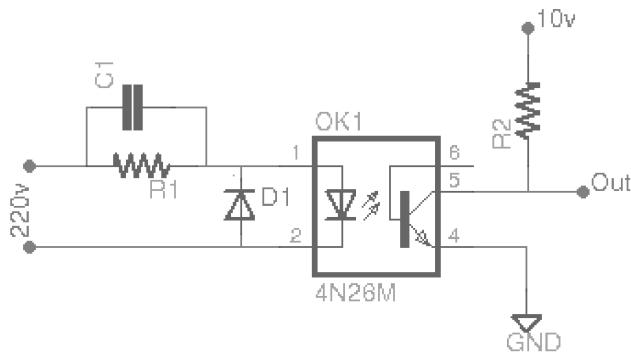


Figura 2.11: Circuito del detector de cruce por ceros

entrada y la salida del opto-acoplador debido a que no existe ninguna unión física entre el foto-diodo y el foto-transistor.

En este caso el foto-diodo es alimentado con la tensión de red, mediante  $C_1$  y  $R_1$ . El capacitor se utiliza para limitar la corriente entrante, (cuyo pico debe ser menor de 60mA) y está calculado para que (en conducción) dicha corriente sea la adecuada para generar la luz suficiente que permita saturar al transistor.

Debido a que  $C_1$  se carga con tensiones muy altas se debe colocar la resistencia  $R_1$  en paralelo para que el capacitor se descargue cuando el módulo es desconectado de la línea. El valor de  $R_1$  debe ser por lo menos 10 veces mayor que la reactancia de  $C_1$  a la frecuencia de red, para que cuando el módulo este funcionando la corriente que circula por ella sea despreciable y no disipe potencia.

Se comenzó calculando a  $C_1$  para obtener un pico de corriente de 10mA. Con este valor se logró la señal de salida deseada, evitando colocar un capacitor demasiado grande y voluminoso. La expresión que permite calcular el valor pico de la corriente sin tener en cuenta el efecto de  $R_1$  es:

$$I_{in} = \frac{220V\sqrt{2}}{X_c}$$

En donde:

$$|X_c| = \frac{1}{2\pi 50Hz C_1}$$

Despejando  $X_c$  para obtener la corriente deseada tenemos:

$$|X_c| = \frac{220V\sqrt{2}}{10mA} \rightarrow |X_c| = \frac{311,12V}{10mA} = 31K\Omega$$

Luego se despeja  $C_1$ :

$$|X_c| = \frac{1}{2\pi 50Hz C_1} \rightarrow C_1 = \frac{1}{2\pi 50Hz |X_c|} = 102nF$$

Se adoptó  $C_1=100nF$ , y  $R_1=470K\Omega$ . El diodo  $D_1$  colocado en contraposición se utiliza para que el opto-acoplador no tenga que soportar 220V cuando no conduce corriente. Si ocurriera esto se dañaría ya que la tensión de ruptura en inversa del mismo es de 5V. Por otra parte, la resistencia  $R_2$  se utiliza para polarizar el foto-transistor que se alimenta con 10V.

Cuando hay un ciclo positivo de la señal de red el foto-diodo comienza a conducir, se satura el transistor y en el colector tenemos la tensión  $V_{ce}$  de saturación (aproximadamente 0 Volts) que el microcontrolador interpreta como un “cero lógico”.

En el momento en el que la tensión en los terminales del foto-diodo es menor a  $V_f$ <sup>4</sup>, este deja de conducir, esto produce que el foto-transistor se corte y que la tensión de salida se eleve a 10V produciéndose así un flanco positivo. Cuando la señal de línea supera la tensión  $V_f$  (se puede suponer nuevamente que es en el cruce por cero de la señal de red) el foto-diodo comienza a conducir nuevamente y esto hace que el foto-transistor se sature y produzca un flanco negativo en la salida. El resultado es una señal cuadrada de 10Vpp y una frecuencia de 50Hz cuyos flancos coinciden con los cruces por cero de la red eléctrica. En la Figura 2.12 se puede ver una captura de la pantalla de un osciloscopio en donde se observa la señal de salida del detector de envolvente.

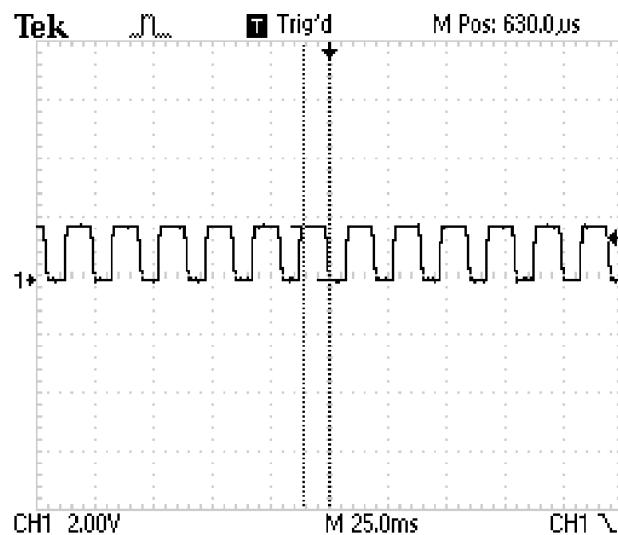


Figura 2.12: Señal de salida del detector de cruce por cero

### 2.2.2. Etapa de entrada/salida

La etapa de entrada/salida es el circuito que interactúa con la red eléctrica. Se utiliza para inyectar la señal de X-10 cuando se transmite un mensaje, y para ingresar al módulo la portadora proveniente de la línea. También posee un transistor que se utiliza para desconectar el circuito de recepción, y así proteger los componentes del mismo en el momento en el que se transmite. Se diseñó siguiendo la premisa de aislar al circuito de la línea eléctrica, la misma que se tuvo con el detector de cruce por cero.

En primer lugar, se decidió implementar el circuito con opto-acopladores, por su bajo costo y su reducido tamaño. Cuando se intentaba recibir las señales de 120KHz se deformaban, debido a que el ancho de banda del dispositivo era insuficiente. Por otra parte, cuando se utiliza un opto-acoplador para transmitir se necesita una fuente del lado de la red eléctrica para polarizar el opto-

<sup>4</sup>Se puede suponer que esto se produce cuando hay un cruce por cero de la señal de 220V debido a que  $V_f = 1,5V \ll 220V$

transistor del mismo<sup>5</sup>. Lo cual trae aparejado un aumento de la cantidad de hardware necesario para la implementación. Después de infinidad de pruebas infructuosas se concluyó que este no era el dispositivo más adecuado para realizar esta tarea. Luego de evaluar otras opciones se decidió utilizar un transformador de núcleo de ferrite. La Figura 2.13 muestra el circuito de la etapa de entrada/salida.

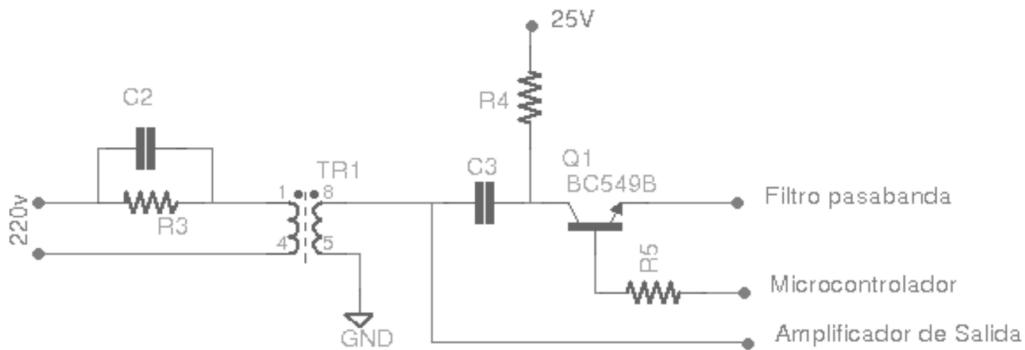


Figura 2.13: Circuito de la Etapa de entrada/salida

Los transformadores de núcleo de ferrite presentan varias ventajas:

- **Confiabilidad:** El transformador es uno de los dispositivos mas confiables y durables que existen en electrónica, además no se ve afectado por los sobre-picos de tensión que se producen en la red eléctrica .
- **Menor cantidad de hardware:** Con su utilización se elimina el problema de las fuentes que se requerían con los opto-acopladores, y se puede utilizar el mismo dispositivo para transmisión y recepción.
- **Permiten adaptar impedancias:** Uno de los problemas que se presentan a la hora de transmitir señales por la red eléctrica es su baja impedancia. Esto dificulta el diseño de las etapas de salida de los amplificadores (como se vera luego). Modificando la relación de espiras del transformador se pueden resolver los problemas de adaptación.

Para el diseño de esta parte del módulo se realizó una medición de la impedancia que presentaba la línea eléctrica a la frecuencia de la portadora (120KHz). Para esto se utilizó el circuito de la Figura 2.14. Se colocó un transformador con una relación de espiras de 1:1 en serie con un capacitor de  $0,1 \mu\text{F}$ , que se utilizó para filtrar la componente de 50Hz. A la frecuencia de portadora presenta una impedancia de  $13,2\Omega$ , mientras que a 50Hz su impedancia es de  $30\text{K}\Omega$ . En el secundario del transformador se colocó un potenciómetro de  $100\Omega$  en serie. Luego se inyectó una señal de  $10\text{Vpp}$  y  $120\text{KHz}$  en el generador y se conectó un osciloscopio como indica la figura.

Para realizar la medición se varió el potenciómetro hasta lograr que la amplitud pico a pico de la señal medida fuese la mitad de la señal de entrada ( $5\text{Vpp}$ ). En este punto la impedancia del secundario del transformador es igual a la resistencia del potenciómetro (que a su vez es igual a la impedancia reflejada por el primario debido la relación de espiras). Luego se desmontó el mismo y se midió su

<sup>5</sup> Esto se debe a que no se puede utilizar la fuente del módulo porque esto eliminaría la aislación buscada.

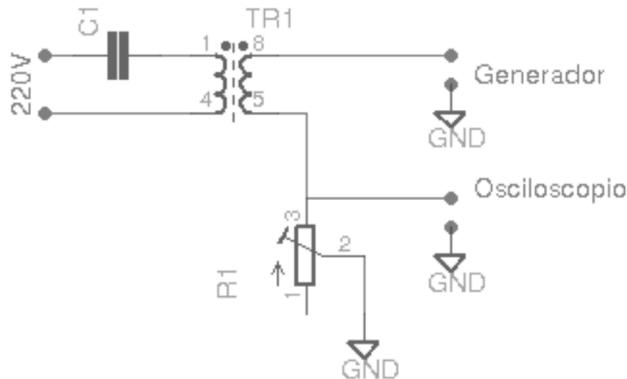


Figura 2.14: Circuito para la medición de impedancia de la red eléctrica

valor, aproximadamente  $21\Omega$ . A la impedancia obtenida en el procedimiento anterior se le debe restar la de  $C_1$  que está en serie con la línea. Haciendo el cálculo se obtiene un valor de  $7,8\Omega$  que es la impedancia de la red eléctrica a 120KHz.

Otra de las mediciones que se realizaron fue la del ruido que presentaba la línea. Para esto se armó un circuito similar al que se utilizó para realizar la medición anterior, pero se conectó el osciloscopio directamente en el secundario del transformador. Se midió un nivel de ruido de 1Vpp.

El lugar en donde se realizaron las pruebas presenta un elevado nivel de ruido, debido a la gran cantidad de computadoras y máquinas eléctricas conectadas. Es poco probable que en una vivienda exista un nivel de ruido superior al medido. Se puede afirmar esto último, porque en una vivienda promedio la cantidad de artefactos conectados a la red eléctrica es mucho menor que en el laboratorio en donde se realizó la medición, y no existe maquinaria eléctrica. Por lo tanto, el diseño fue hecho suponiendo que el valor medido (1Vpp) es el máximo posible.

Con los resultados de las mediciones anteriores se diseñó el transformador de ferrite. Se optó por una relación de vueltas que permitiera adaptar impedancias, para lograr una carga aceptable para el amplificador de salida, y que por otra parte permitiera transmitir una señal que superara el nivel de ruido presente en la línea.

A medida que se aumentan las vueltas del secundario aumenta la cantidad de tensión que se debe aplicar a dicho bobinado (en la transmisión) para obtener una amplitud aceptable en el primario. Considerando lo dicho anteriormente, y luego de realizar pruebas se llegó a la conclusión de que una relación de vueltas 1:20 era la más apropiada. La impedancia reflejada por la línea en el secundario del transformador es:

$$| Z_s | = Z_p \left( \frac{N_2}{N_1} \right)^2 \rightarrow | Z_s | = 21\Omega (20)^2 = 8,40K\Omega$$

Teniendo en cuenta esto se decidió que la tensión de salida del amplificador debía ser de 25Vpp. De esta forma, se logra una señal de 1,2Vpp en el primario del transformador, que es suficiente para superar el piso de ruido, y para que el receptor pueda recibir los pulsos luego del filtrado.

Si se observa el circuito de la Figura 2.13 se puede ver que el transformador está conectado a la línea a través de  $C_2$  y  $R_3$ . Este es un filtro igual al que se utilizó en el detector de cruce por cero, como se explicó anteriormente se utiliza para atenuar la señal de 50Hz y que la misma no se encuentre presente en el primario del transformador.

El transistor Q1 se utiliza para desconectar la entrada del circuito de recepción en el momento en el que un dato es transmitido, debido a que los componentes del amplificador sintonizado pueden soportar una señal de entrada de hasta 10V y la señal que se aplica al secundario del transformador durante la transmisión es de 25Vpp.

La base de Q1 es comandada por el microcontrolador (conectado mediante R5) que coloca un “uno lógico” (3,3V) en el momento de la recepción. De esta forma, Q1 deja pasar a la señal que proviene de la red eléctrica para que ingrese al filtro pasobanda. Por otra parte, C3 se utiliza como capacitor de desacople y la resistencia R4 se utiliza para polarizar al transistor que se alimenta con 25V. Cuando el circuito se utiliza para transmitir se aplica un “cero lógico” a la base de Q1 lo que provoca que este se corte y no deje pasar ninguna señal de entrada al filtro.

### 2.2.3. Amplificador de salida

Este circuito se utiliza para amplificar la portadora proveniente del microcontrolador. El mismo alimenta el transformador de la etapa de entrada/salida con el cual se conecta el módulo a la red de 220V.

En la Figura 2.15 se puede ver parte de la etapa de entrada/salida descripta anteriormente y el amplificador de salida. Este está compuesto por dos etapas transistorizadas. La portadora, una señal cuadrada de 3,3Vpp y 120KHz que sale del microcontrolador, ingresa a la base de Q2 (BC548). Esta etapa está configurada como emisor común, y está calculada para que el transistor Q2 trabaje fuera de su zona lineal. El mismo se satura cuando la salida del microcontrolador está en alto, y se corta cuando se lo excita con un “cero lógico”. Esto hace que su ganancia de tensión sea elevada.

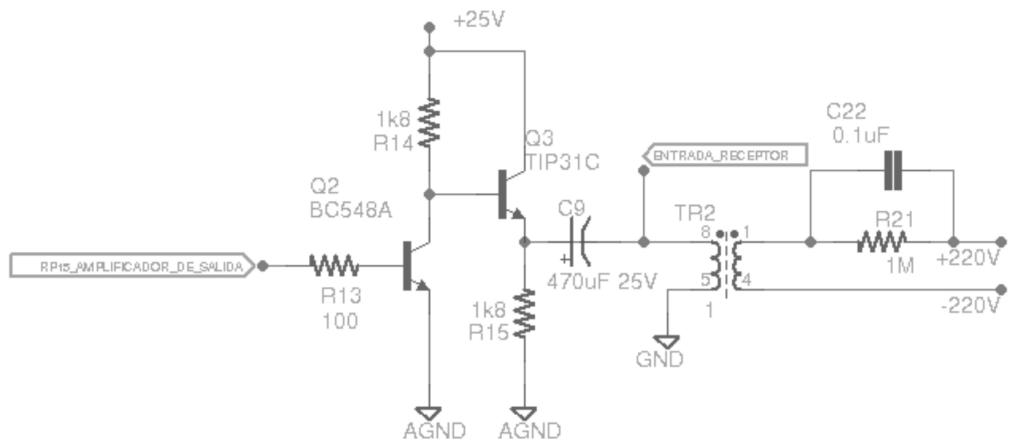


Figura 2.15: Amplificador de salida y Etapa de entrada/salida

En la salida de esta etapa la señal tiene aproximadamente 25V de amplitud pico a pico con un valor medio de 12,5V (durante los pulsos de 1ms), y además esta invertida. Esto es irrelevante debido a que solo es necesario que la señal de 120KHz esté presente durante 1ms sin importar su fase.

La resistencia R14 está calculada para limitar la corriente de colector de Q1 a aproximadamente 13,8mA cuando se encuentra saturado. Si se desea inyectar una portadora con amplitud razonable en la red se necesita que la señal que ingresa al transformador posea una potencia aceptable. Para lograr esto se colocó la segunda etapa del amplificador, en la que se utiliza un TIP31[Cor00] en una

configuración de colector común. Esta le brinda al amplificador la corriente necesaria para alimentar al transformador. Como ocurre con cualquier etapa de este tipo la ganancia de tensión es prácticamente unitaria.

Cuando la interfaz no se encuentra transmitiendo se debe colocar un nivel alto en la base de Q2 para que este transistor se sature, haciendo que la base de Q3 quede con una tensión de aproximadamente 0V de manera que quede cortado. Esto se debe hacer para evitar que cuando se reciba una señal por la línea eléctrica el transformador quede conectado a masa haciendo que dicha señal se pierda.

#### 2.2.4. El filtro pasobandas

Debido al ruido presente en la línea eléctrica, se debe filtrar la señal recibida antes de que acceda al amplificador sintonizado del receptor. En la nota de aplicación AN236 de Microchip[Bur2], se utilizó un filtro pasa altos de segundo orden con una frecuencia de corte de 30KHz como el que se muestra en la Figura 2.16.

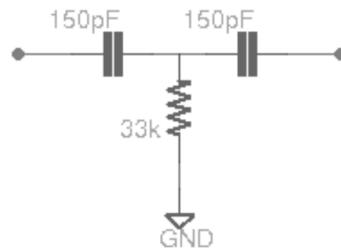


Figura 2.16: Filtro pasa altos de segundo orden

Después de realizar pruebas se llegó a la conclusión de que este filtro no era apropiado debido a que en la red de 220V existen muchas componentes de ruido de frecuencias mayores a 30KHz. Cuando se midió la salida de dicho filtro con el osciloscopio se observó que muchas de estas componentes pasaban el filtro ingresando en las etapas posteriores al mismo, lo que originaba una deformación en el techo de los pulsos recibidos. A su vez esta deformación causaba problemas en la decodificación del mensaje.

Por este motivo se modificó el circuito y se procedió a diseñar un filtro más selectivo. Se estudiaron varias opciones, y se concluyó que un pasobandas del tipo Chebyshev era lo más adecuado. Con este tipo de filtros se puede lograr la respuesta en frecuencia deseada con un orden relativamente bajo, y el comportamiento del sistema no se ve afectado por el ripple en la banda de paso.

El filtro debía tener una frecuencia central de 120KHz y un ancho de banda lo suficientemente grande para que los pulsos pudieran pasar sin sufrir deformaciones en sus flancos. En este caso se comenzó con un filtro de tercer orden con un ancho de banda de 3dB de aproximadamente 90KHz. En pruebas posteriores se pudo observar que cumplía perfectamente con el objetivo planteado.

Para el cálculo de los componentes del circuito se utilizó la versión para estudiantes de “Elsie” [Ton11], un software para el diseño de filtros pasivos. Su uso es muy intuitivo, ya que se colocan parámetros tales como: topología, ancho de banda, orden, etc. y luego se puede visualizar tanto el circuito resultante como el gráfico de respuesta en frecuencia. Además, se pueden modificar los componentes del circuito manualmente para hacer coincidir sus valores con los disponibles comercialmente y ver qué efecto causa esto en la respuesta del filtro. En la Figura 2.17 se muestra la pantalla de configuración de “Elsie”, en donde se colocan los parámetros del filtro que se desea diseñar.

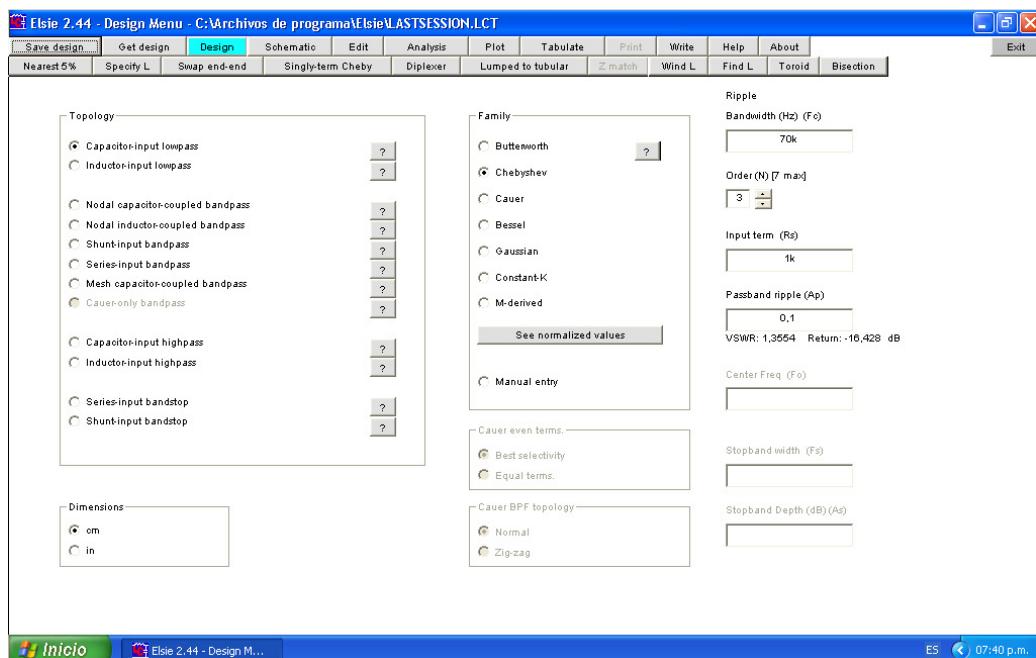


Figura 2.17: Pantalla de configuración de “Elsie”

Luego de ingresar las características del filtro en el programa se puede acceder a la solapa “schematic” que se muestra en la Figura 2.18. En ella se observa la primera versión del circuito a diseñar. El software calcula valores de componentes exactos. Dichos valores se pueden modificar utilizando la solapa “Edit”. Esta se utilizó para modificar el valor de los inductores, debido a que los presentes en el circuito propuesto por “Elsie” eran demasiado grandes. Se adoptaron valores de 1mH y 3mH. Cualquier modificación en alguna parte del circuito hace que a su vez el programa modifique otros componentes para que la transferencia siga siendo lo más aproximada posible a la deseada.

Luego de modificar los componentes para adoptar valores comerciales se llegó al diseño definitivo, el cual se muestra en la Figura 2.19. En la Figura 2.20 se muestra el gráfico de la respuesta en frecuencia del filtro extraído de “Elise”. Aquí se puede observar que tiene una ganancia de 0dB en un rango de frecuencia que va desde 90KHz hasta los 100KHz y las frecuencias de corte se encuentran en 73KHz y 164KHz. En la Figura 2.21 se puede ver una captura de la pantalla de un osciloscopio, en donde se observan pulsos de 1ms de duración luego de pasar por el filtro pasobandas. Estos pulsos forman parte de un mensaje X-10, y dentro de ellos se encuentra la señal portadora de 120KHz.

### 2.2.5. Amplificador sintonizado y detector de envolvente

Una vez que la señal es recibida y filtrada, se la debe amplificar. A su vez se debe acondicionar la misma con un detector de envolvente para lograr que el microcontrolador pueda detectar la presencia de los unos o ceros lógicos presentes en el mensaje recibido.

El circuito que se utilizó para realizar esta tarea es uno de los mas frecuentemente utilizados en aplicaciones de X-10, debido a que es sencillo y económico. En el mismo se utiliza un séxtuple inversor CD4069 [Cor99] para obtener un amplificador y un detector de envolvente.

Si se realimenta negativamente a cualquiera de los inversores que posee el integrado para utilizarlo

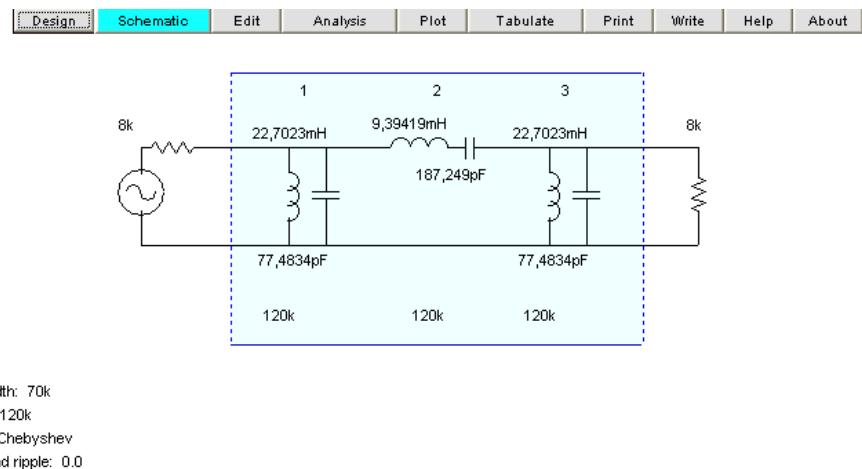


Figura 2.18: Primera versión del filtro

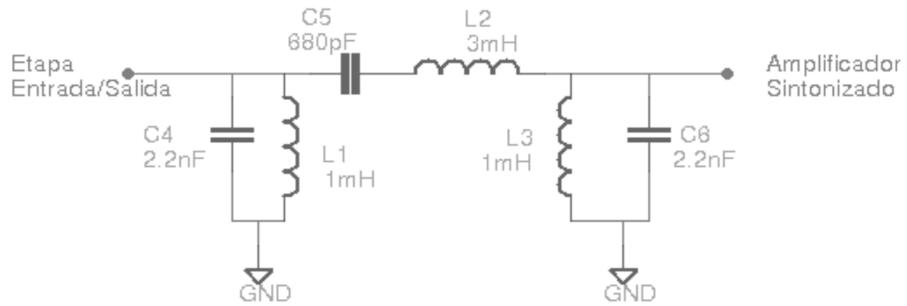


Figura 2.19: Diseño definitivo del filtro pasobanda

en su zona lineal se obtiene un amplificador. Si a su vez se utilizan cuatro inversores de esta forma y se los conecta en cascada se puede obtener una considerable ganancia. En la Figura 2.22 se muestra el circuito del amplificador.

Para estar completamente seguros de que se amplificará la portadora de X-10 y no cualquier otra señal indeseada se debe comenzar realizando la tarea con dos etapas sintonizadas a una frecuencia de 120KHz. Luego, para finalizar con el proceso de amplificación, se agregan dos etapas mas no sintonizadas para llegar a un nivel de señal adecuado.

Con la salida del amplificador se excita al detector de envolvente (Figura 2.23) que está realizado con un negador del CD4069, un diodo 1N4148 [Sem04], el capacitor C14 y el resistor R8. Los resistores R9 y R12 que se encuentran conectados a la salida del detector se utilizan para reducir la tensión pico de la salida de 10V a 2.5V debido a que este punto se conecta a la entrada RB5 del microcontrolador, que soporta hasta 3,3V y es la encargada de sensar la señal entrante. En la Figura 2.24 se muestra una captura de la pantalla del osciloscopio. En la parte superior de la pantalla se observa la señal de entrada del amplificador, se trata de los pulsos de 1ms que contienen la portadora de 120KHz, estos tienen una amplitud de aproximadamente 1,2Vpp. En la parte inferior de la pantalla se puede ver la señal de salida del detector de envolvente, su amplitud es de casi 9Vpp. Aquí se puede ver claramente

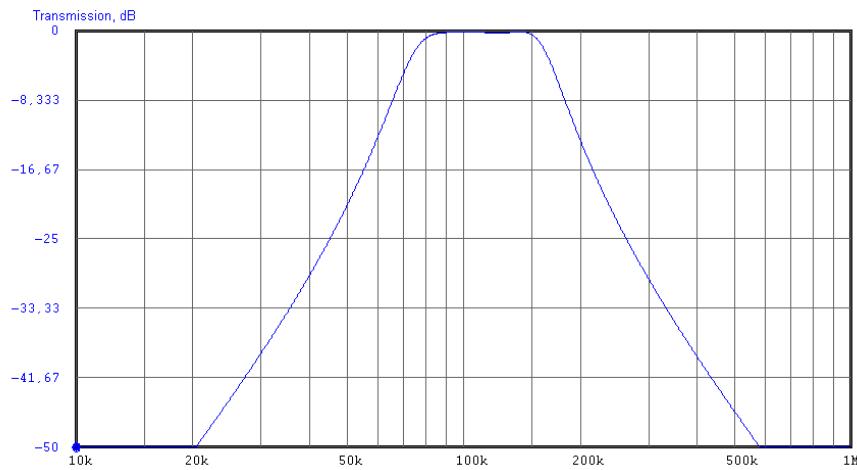


Figura 2.20: Respuesta en frecuencia del filtro pasobanda

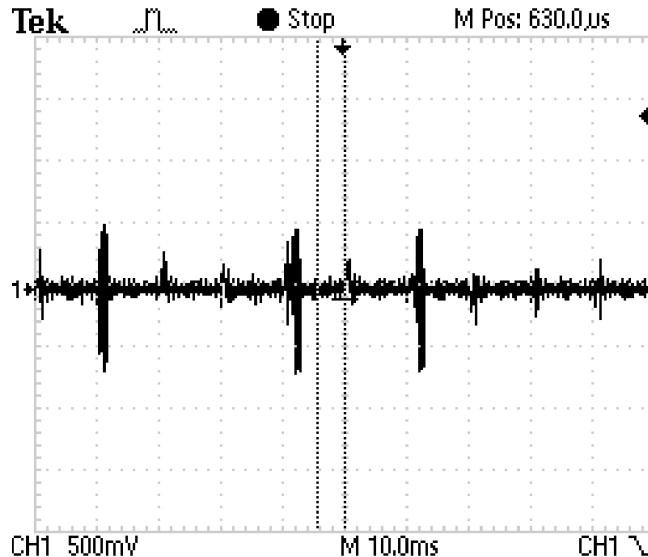


Figura 2.21: Pulsos de un mensaje X-10 luego de pasar por el filtro pasobandas

como se demodula la señal de entrada.

### 2.2.6. Fuente de alimentación

El módulo X-10 necesita tres tensiones de alimentación: 25V para el amplificador de salida, 10V para el amplificador de entrada y el detector de envolvente, y 3,3V para el microcontrolador. Por otra parte, el módulo Ethernet funciona con una tensión de alimentación de 3,3V.

Se decidió instalar la fuente de alimentación de la interfaz en el módulo X-10, debido a que dicho módulo debe estar conectado a la red eléctrica en todo momento. El módulo Ethernet no posee fuente de alimentación, solo un regulador de 3,3V. Cuando se utiliza en la interfaz 802.3/X-10, se alimenta

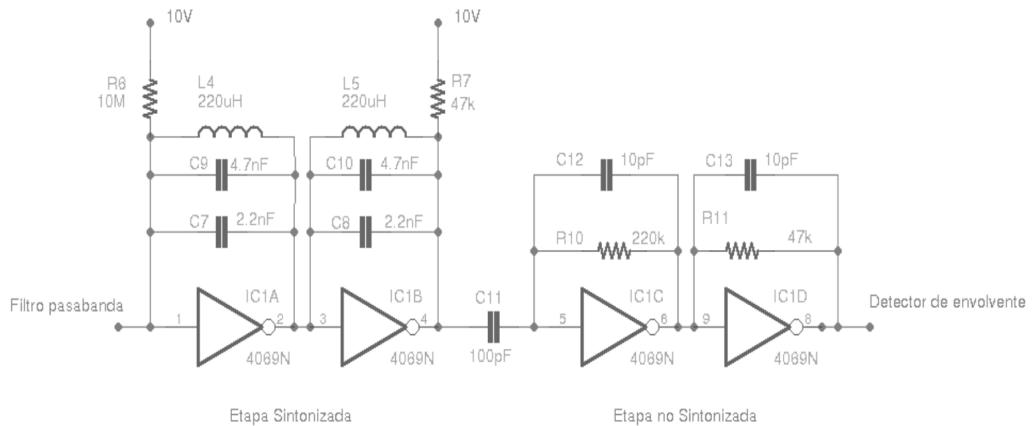


Figura 2.22: Amplificador sintonizado

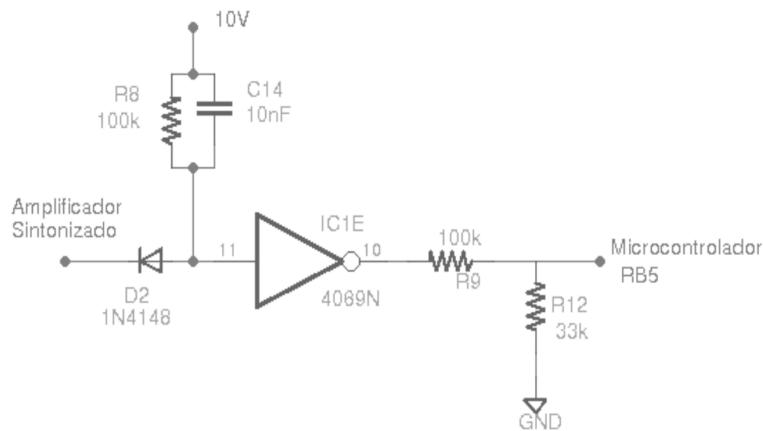


Figura 2.23: Detector de envolvente

con una salida de 5V que posee el módulo X-10.

En la nota de aplicación AN236 de Microchip [Bur2], se propone una fuente de alimentación sin transformador como la que se muestra en la Figura 2.25. Esta fuente utiliza dos capacitores (C1, C2) en paralelo para reducir la tensión de la red eléctrica. La resistencia R1 es utilizada para que los capacitores C1 y C2 se descarguen cuando el circuito está desconectado. Los diodos 1N4005 forman un rectificador de media onda. El capacitor C3 se utiliza como filtro, y el diodo zener regula la tensión de salida en un nivel de 5V.

La principal ventaja de esta fuente de alimentación es su volumen reducido y su bajo costo, por eso fue una de las opciones evaluadas a la hora de realizar el diseño. Por otra parte, este tipo de circuitos tiene dos grandes desventajas.

La primer desventaja, es que se trata de fuentes de alimentación **extremadamente inseguras**, debido a que no existe ninguna aislación entre la carga y la línea eléctrica. La segunda desventaja es que si el consumo de corriente de la carga no es constante, la tensión de entrada al rectificador sufre grandes variaciones.

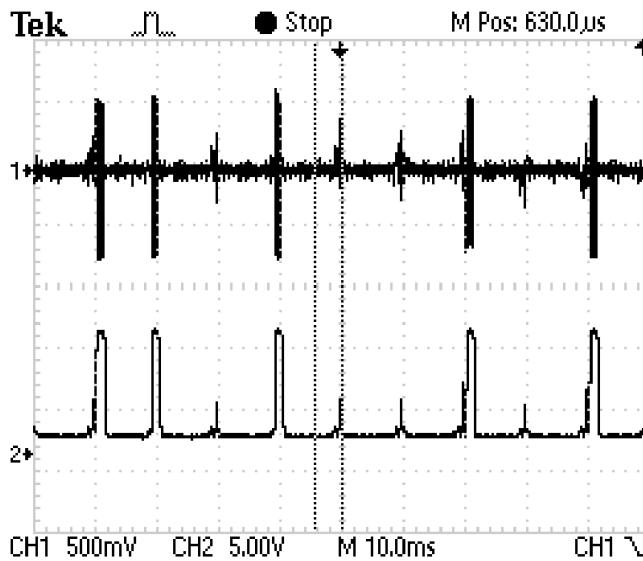


Figura 2.24: Señal de entrada del amplificador y salida del detector de envolvente

Este tipo de fuente fue descartada, debido a que en todo momento se **priorizó la seguridad** del usuario. El módulo X-10 tiene un consumo de corriente de 100mA, esta es la corriente que debe suministrar la fuente de alimentación cuando es utilizado como actuador bidireccional. Por otra parte, cuando se utiliza al módulo X-10 dentro de la interfaz 802.3/X-10 la fuente de alimentación debe entregar una corriente de 300mA, debido a que se debe alimentar al módulo Ethernet que presenta un consumo de 200mA. Esta gran variación en la corriente de carga hace que sea muy difícil diseñar una fuente sin transformador capaz de alimentar al circuito.

El problema fue resuelto utilizando una fuente commutada preeensamblada de 25V 400 mA. Estas fuentes de alimentación presentan muchas ventajas, en primer lugar, su tamaño es muy reducido, son muy económicas, y su salida se encuentra aislada de la red eléctrica. Por otra parte, no se ven afectadas por la fluctuación de corriente en la carga.

Para obtener la tensión de 10V que alimenta al amplificador sintonizado y al detector de envolvente se utilizó un regulador UA78M10 [Inc10b]. Además, se utiliza la salida de 10V para alimentar dos reguladores<sup>6</sup> UA78M33C [Inc10b] con los que se obtienen las tensiones de 3,3V para alimentar a los microcontroladores. Los diagramas esquemáticos de esta parte del circuito se pueden ver en el apéndice A.

### 2.3. Microcontrolador

Es el componente más importante del módulo. En esta sección se justificará la elección del microcontrolador utilizado para el proyecto, y luego se describirán los principales aspectos relacionados con el hardware del mismo. En secciones posteriores se explicarán los aspectos relacionados con el software asociado.

Las tareas que debe realizar varían de acuerdo con el uso que se le dé al módulo. Si el módulo se configura para formar parte de la interfaz 802.3/X-10, el microcontrolador debe realizar las siguientes

<sup>6</sup> Uno de estos reguladores se encuentra en el módulo X-10 y otro en el módulo Ethernet

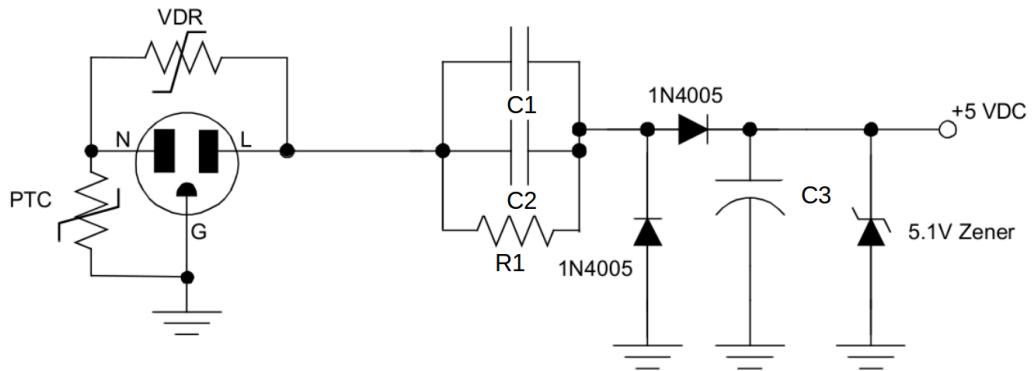


Figura 2.25: Fuente de alimentación sin transformador

tareas:

- Utilizar la salida del detector de cruce por cero para sincronizarse con la red eléctrica.
- Recibir y decodificar la orden proveniente del módulo 802.3.
- Ensamblar los bloques de datos X-10.
- Generar la portadora de 120KHz durante 1ms para transmitir.
- Transmitir la orden.
- Sensar la red eléctrica en busca de la confirmación del receptor.

Si el módulo se configura como actuador bidireccional de X-10, el microcontrolador debe realizar las siguientes tareas:

- Utilizar la salida del detector de cruce por cero para sincronizarse con la red eléctrica.
- Sensar la salida del detector de envolvente en busca de un mensaje X-10.
- Decodificar la señal entrante y realizar la acción solicitada.
- Generar la portadora de 120KHz durante 1ms para transmitir.
- Ensamblar los bloques de datos X-10 para transmitir la confirmación al receptor.
- Transmitir la confirmación al receptor.

Luego de evaluar la gran cantidad de microcontroladores presentes en el mercado se optó por uno de la familia de Microchip. La mayor ventaja por sobre otras empresas fabricantes de circuitos integrados es la gran cantidad de información en la red sobre todos sus productos y sus herramientas de diseño. Esto facilita enormemente la tarea del desarrollador debido a que se puede llegar a obtener el máximo potencial de los dispositivos utilizados en un tiempo relativamente corto.

Se optó por un microcontrolador de 16 bits, debido a que estos poseen mucha mayor resolución que los de 8 bits. Esto es una gran ventaja cuando se desea generar señales con el microcontrolador, debido a que se logra una mejor exactitud en el período de las mismas. Otra de las ventajas de los circuitos integrados de 16 bits es que pueden ser programados con el compilador C30, este ofrece muchas ventajas y esta mucho más optimizado que el C18 que se utiliza para las familias de 8 bits.

Microchip ofrece dos familias de microcontroladores de 16 bits, la familia PIC24F de bajo costo y *performance* media y la familia PIC24H de alta *performance*. Entre las familias hay muchos atributos

en común, entre ellos son la compatibilidad de pines, compatibilidad de periféricos, misma herramienta de desarrollo, entre otras.

El circuito integrado elegido tanto para la realización de este módulo como para la del módulo 802.3 es el PIC24HJ128GP502 [Inc09], su gran velocidad (80MHz, 40 millones de instrucciones por segundo), su encapsulado (28-Pin SDIP) que permite realizar prototipos de forma sencilla, la gran cantidad de temporizadores que posee, sus 4 módulos de Comparación/Captura/PWM, su gran capacidad de memoria 128Kb, y sus 16 pines remapeables son algunas de las particularidades que lo convierten en el más adecuado para el proyecto. Sus principales características son:

- El CPU puede operar hasta 40 millones de instrucciones por segundo(MIPS).
- Oscilador interno de 8MHz.
- PLL interno.
- Memoria de programa de 128kB.
- Memoria RAM 8kB.
- Direccionamiento lineal de memoria de programa de hasta 4MB.
- Direccionamiento lineal de la memoria de datos de hasta 64kB.
- *Stack* por software.
- Hardware para multiplicación 16x16, división 32x16 y 16x16.
- 8 canales DMA (Acceso a memoria directo).
- 2 UART - 2 SPI - 1 I2C.
- Módulo AD. 10-bit 1.1 Msps o 12-bit a 500ksps.
- 4 Módulos Comparación/Captura/PWM. 16-bits PWM.
- Timers, 5 x 16-bit 2 x 32-bit.
- RTC (Reloj de tiempo real).
- PPS (Peripheral Pin Select).

En la Figura 2.26 se puede ver un esquema del PIC24HJ128GP502 y la distribución de pines del mismo.

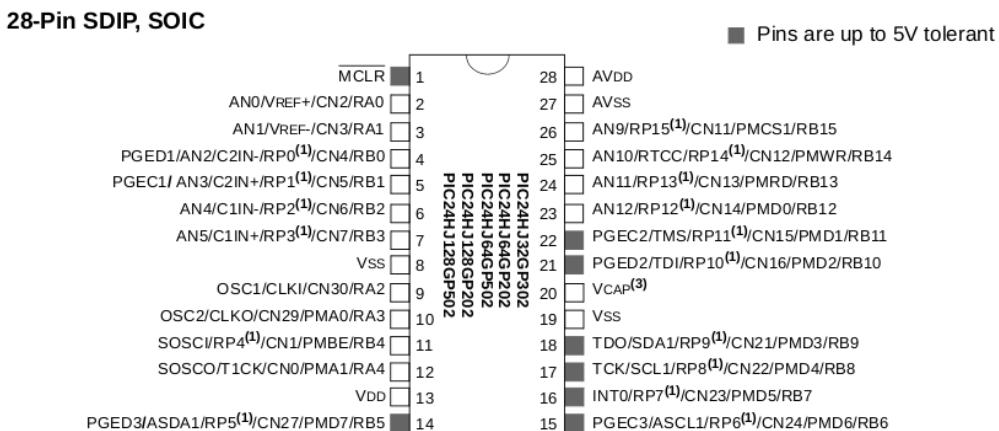


Figura 2.26: Diagrama de pines del PIC24HJ128GP502

### 2.3.1. Configuración del Oscilador

Como se mencionó anteriormente una de las grandes ventajas de este microcontrolador es su elevada velocidad. Este puede trabajar a 40 MIPS para lo cual necesita de una frecuencia de reloj de 80MHz. Para lograr esta elevada frecuencia se debe utilizar el lazo de enganche de fase que proporciona el circuito integrado. En este caso se necesita una señal de reloj exacta y estable debido a que (como se verá después) se utilizan los *timers* y el módulo *Output Compare* para generar la señal de portadora y los retardos necesarios. Por este motivo se utilizó el oscilador primario con PLL en la configuración HS con un cristal de 10MHz.

En la Figura 2.27 se puede ver el diagrama en bloques del PLL. Para configurarlo se deben programar tres registros PLLPRE, PLLDIV, y PLLPOST. Con ellos se configuran los valores de N1, M, y N2 respectivamente. El valor N1 se utiliza para dividir la frecuencia de la señal que proviene del oscilador primario para que la señal que ingrese al PLL esté dentro de un rango de 0.8 a 8 MHz. N1 puede variar entre 2 y 33, y en este caso se eligió el valor 2 para tener una frecuencia de entrada de 5MHz.

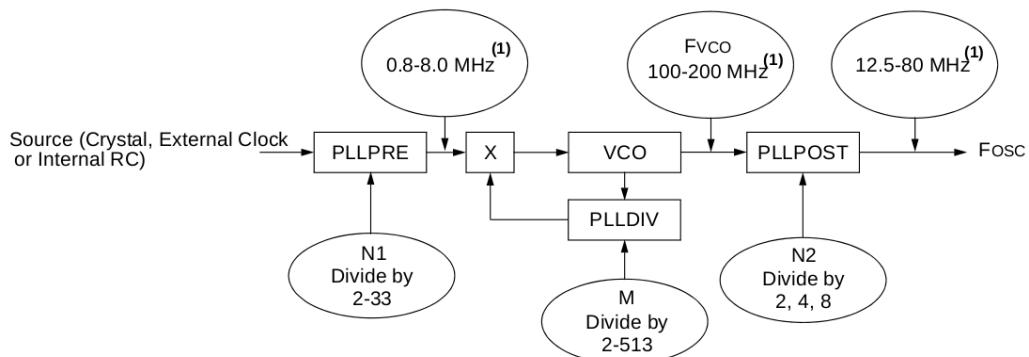


Figura 2.27: Diagrama en bloques del PLL

El divisor de la realimentación del PLL se configura con el valor M. Con él se puede configurar la frecuencia de salida del VCO que debe estar dentro de un rango de 100 a 200MHz. El valor de M puede variar entre 2 y 513, en este caso se eligió un M de 32 para tener una frecuencia de salida de 160MHz.

Luego se divide a esta frecuencia por un factor N2 (que puede valer 2,4 u 8) y así se obtiene la señal de reloj que usará el integrado. En este caso N2 configuró con el valor 2 para obtener una frecuencia de 80MHz, que es la máxima con la que puede funcionar el PIC24HJ128GP502.

Luego de configurar estos valores se debe realizar un cambio de *clock* para pasar del oscilador interno (con el que comienza a funcionar el integrado) al oscilador primario en el modo HS con PLL. Esto se realiza mediante el software y será explicado en la Sección 2.4. Es muy importante que se respeten los rangos de frecuencias de la Figura 2.27 debido a que si cualquier señal está fuera de dichos rangos el PLL no se enganchará y el cambio de clock no se podrá efectuar, provocando que el integrado funcione a una velocidad menor o que el programa quede detenido esperando a que dicho cambio se realice.

El diagrama de conexión del oscilador a cristal se muestra en la Figura 2.28, el fabricante recomienda conectar el cristal con dos capacitores de 33pF y una resistencia de 10MΩ para darle estabilidad al reloj.

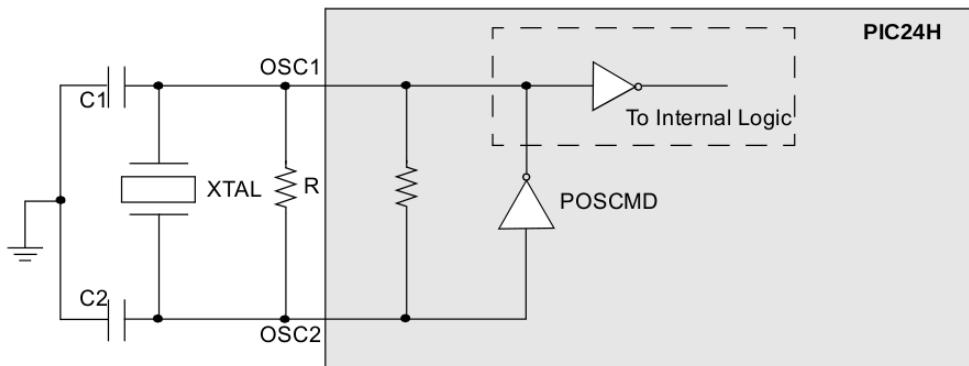


Figura 2.28: El diagrama de conexión del oscilador a cristal

### 2.3.2. Mapeo de pines

En la Figura 2.29 se puede ver el esquema de conexión del microcontrolador, mientras que en la Figura 2.30 se observan las conexiones básicas para su funcionamiento. El circuito del *master clear*, el del *clock*, y C2 que es un capacitor de  $10\mu F$  que debe ser colocado entre el pin 20 y la referencia.

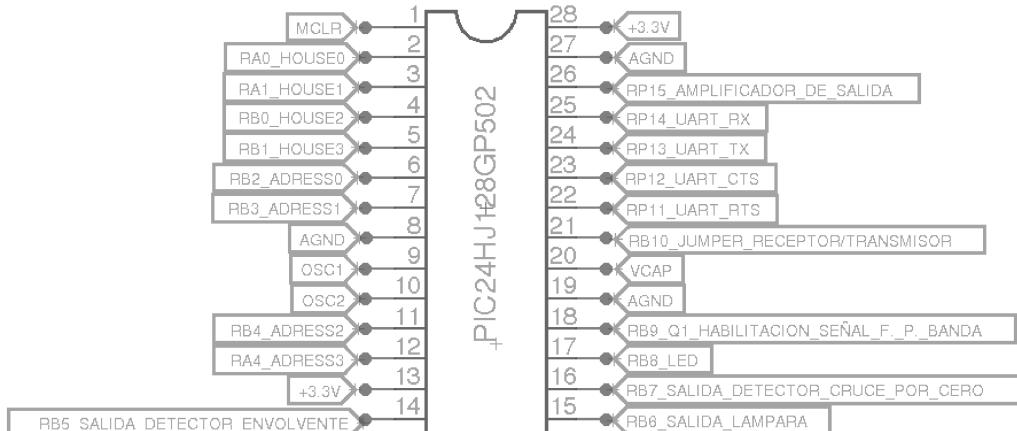


Figura 2.29: Esquema de conexión del microcontrolador

En la Figura 2.30 también se puede ver el circuito del *jumper* que se colocó en la pata RB10, que sirve para configurar al módulo como transmisor (si está colocado) o como receptor (si no se encuentra colocado). La salida del módulo *Output Compare* (que se utilizó para generar la portadora) se puede mapear en cualquiera de los pines disponibles para este propósito (RP0 a RP15) mediante software. En este caso se utilizó la pata RP15. Por este terminal se extrae la señal de X-10 para transmitir a la red eléctrica. Por lo tanto, es la que se conecta al amplificador de salida.

Para realizar la sincronización con la red eléctrica, se conecta la salida del detector de cruce por cero al pin RB7 en el que esta asignada la interrupción externa INT0. Esta se utiliza para, como su nombre lo indica, interrumpir el programa principal cada vez que se produce un flanco y ejecutar una

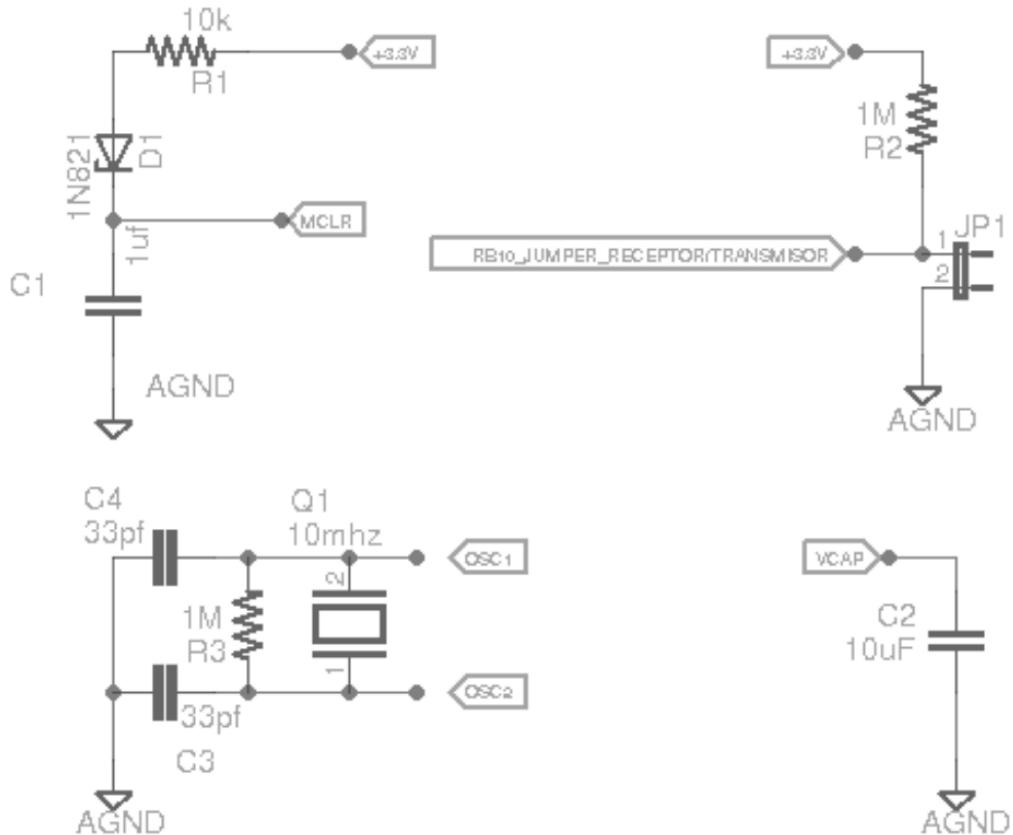


Figura 2.30: Circuitos básicos para el funcionamiento del microcontrolador

subrutina. Dicho flanco puede ser negativo o positivo, y esto puede ser configurado por software. En este caso el tipo de flanco con el que se activa INT0 se alterna (esto será explicado en Sección 2.4). De esta forma, se activa la subrutina con cada variación de la señal de salida del detector, que a su vez coincide con los cruces por cero de la señal de 220V.

La entrada por la cual se recibe la señal de X-10 proveniente de la línea se mapeó en RB5, esta va conectada al detector de envolvente. Como se explicó en la descripción de la etapa de entrada/salida se debe evitar que la señal transmitida ingrese al circuito de recepción. Esto se logra con el transistor Q1 cuya base se encuentra conectada al terminal RB9, con ella se controla por software la habilitación de dicho transistor.

La interconexión con el módulo 802.3 se realiza a través de la UART1. Los terminales de recepción (Rx) y transmisión (Tx) también se pueden mapear en cualquiera de los pines disponibles para este propósito (RP0 a RP15) mediante software. En este caso Rx se mapeó en el pin RP14 y Tx se mapeó en el pin RP13. También se utilizó el control de flujo (esto se explicará en la Sección 2.4) para lo cual son necesarios dos pines mas: *Clear to Send* (U1CTS) que se mapeó en RP12 y *Request to Send* (U1RTS) mapeada en RP11.

Cuando el módulo se utiliza como actuador bidireccional, se encarga de comandar una salida de 220V a la que se conecta un artefacto eléctrico. Esta salida es comandada por el microcontrolador mediante un micro relé, la base del transistor que controla la bobina de dicho relé es conectada al pin

RB8.

Por último se le debe indicar al microcontrolador cual es su dirección de casa y su número de módulo, esto se logra con *jumpers* conectados como el que se ve en la Figura 2.30, los mismos están conectados en los pines RA0, RA1, RB0, RB1 (dirección de casa) y RB2, RB3, RB4, RA4 (dirección de módulo).

### 2.3.3. Módulos temporizadores

Un temporizador interno o *timer* es un módulo de hardware incluido en el mismo microcontrolador que está especialmente diseñado para incrementar automáticamente el valor de un registro asociado (TMRx) cada vez que recibe un pulso. A este pulso se lo llama “señal de reloj”.

El módulo siempre incrementa el valor del registro asociado TMRx. Cuando el valor de la cuenta es igual al valor del registro en el que se fija el período del *timer* (PRx), el registro asociado TMRx se borra y el módulo comienza su cuenta desde cero.

La señal de reloj de cada uno de éstos módulos puede ser de origen interno o externo. Si el origen de la señal de reloj está configurado como externo, el módulo temporizador puede ser utilizado como un contador de eventos externos, incrementando el temporizador con cada pulso recibido mediante el pin correspondiente. Si el origen de la señal de reloj es interno, el temporizador se incrementa con cada ciclo del oscilador. Esto permite utilizar el temporizador como “contador de ciclos de programa”, en donde un ciclo corresponde al tiempo de ejecución de una instrucción.

Estos microcontroladores disponen de *timers* de 16-bits que pueden dividirse en tres tipos de acuerdo a sus funcionalidades:

- *Timer tipo A (timer 1)*
- *Timer tipo B (timer 2, 4, 6 y 8 )*
- *Timer tipo C (timer 3, 5, 7 y 9)*

Los *timers* de tipo A, a diferencia de los demás, pueden operar desde un oscilador de baja potencia de 32KHz y poseen el modo de contador asincrónico desde fuente de *clock* externa. En cambio los *timers* de tipo B y C pueden combinarse para formar uno de 32-bits. Además, aquellos del tipo C pueden activar la conversión AD.

En la Figuras 2.31, 2.32, 2.33 se pueden ver los diagramas en bloques de los tres tipos. Cada módulo *timer* es un contador/temporizador de 16-bits que tiene los siguientes registros de control que pueden ser leídos y escritos:

**-TMRx:** Registro de 16-bits que lleva la cuenta del *timer*.

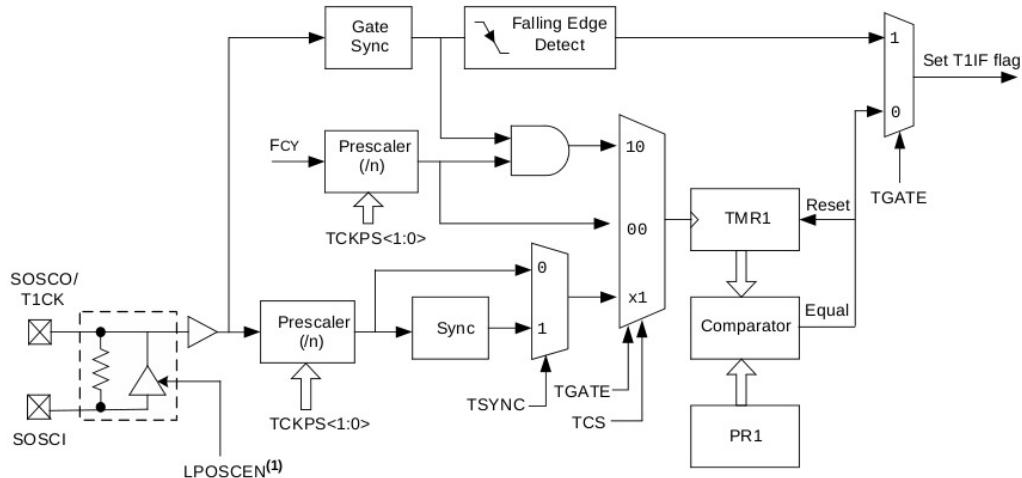
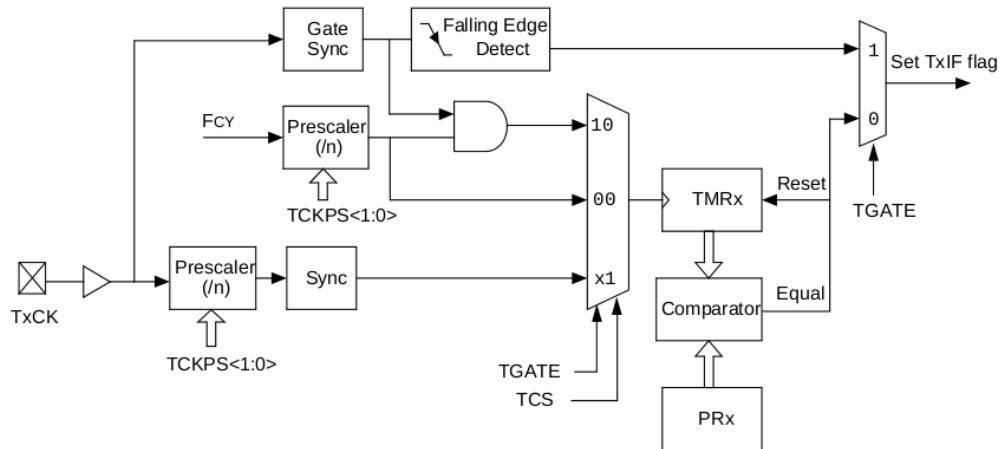
**-PRx:** Registro de 16-bits que fija período al *timer*.

**-TxCON:** Registro de control del *timer*.

También cada módulo tiene asociado sus bits para el control de interrupción, para habilitación (TxIE), para control de estado (TxIF) y para fijar prioridad (TxIP).

Los *timers* pueden trabajar de los siguientes modos:

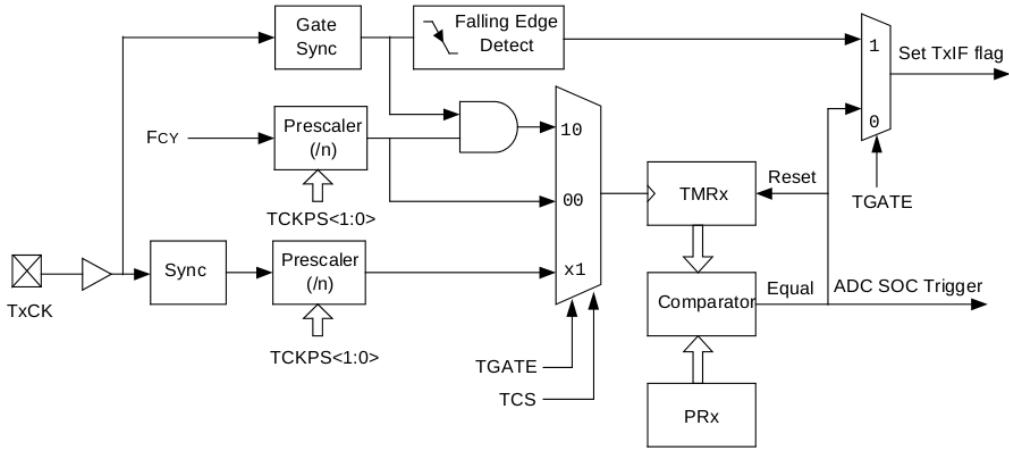
- 1- Modo temporizador.**
- 2- Modo temporizador controlado por pin externo (*Gated timer*).**
- 3- Modo contador asincrónico (únicamente *timers* tipo A).**

Figura 2.31: Diagrama en bloques del *timer* tipo AFigura 2.32: Diagrama en bloques del *timer* tipo B

#### 4- Modo contador sincrónico.

En los modos 1 y 2 se utiliza como *clock* el ciclo de instrucción interna (Fcy). En cambio, en los modos 3 y 4 se utiliza el *clock* externo derivado del pin TxCK. Para el control del modo de operación se utilizan los siguientes bits:

- **TCS (TxCON<1>):** Bit que determina fuente de *clock*.
- **TSYC (TxCOM<2>):** Bit para determinar si el *clock* externo se sincroniza con el ciclo de instrucciones (únicamente *timers* tipo A).
- **TGATE (TxCON<6>):** Bit que determina modo *gated*.

Figura 2.33: Diagrama en bloques del *timer* tipo C

El reloj de entrada (FCY o pin TxCK) de todos los *timers* posee la opción de ser “preescalado” entre 1:1, 1:8, 1:64 o 1:256. Esta configuración de *preescalador* se realiza mediante los bits TCKPS<1:0> del registro TxCON.

La cuenta de los *preescalers* se borra cuando se efectúa una escritura sobre los registros TMRx y TxCON, cuando se deshabilita el *timer* (TON=0), y al ocurrir un *reset*.

La gran ventaja de los temporizadores es que utilizándolos se pueden realizar retardos de cualquier valor y extremadamente precisos, sin la necesidad de librerías, como se explicará en secciones posteriores. Si además se usan en conjunto con los módulos *Output Compare* del microcontrolador se pueden generar pulsos de prácticamente cualquier frecuencia con ciclo de trabajo variable.

Durante la recepción, el módulo X-10 emplea un temporizador para generar un retardo de  $500\mu s$  luego de cada cruce por cero de la señal de 220V. Este retardo se utiliza para demorar el sensado de la salida del detector de envolvente. En la Figura 2.34 se muestra un gráfico en el que se puede ver el cruce por cero de la señal de 220V y un pulso recibido luego de pasar por el detector de envolvente. También se puede ver el momento en el que se sensa la salida del detector de envolvente. Como se explicó en la Sección 2.1, el máximo retardo que puede existir entre el cruce por cero de la señal de línea y el pulso de X-10 recibido es de  $300\mu s$ . Por otro lado, la duración de cada pulso es de 1ms. Por lo tanto, si la salida del detector de envolvente es sensada después de  $500\mu s$  se puede asegurar que la muestra será tomada justo en el momento en el que el pulso es recibido.

Cuando el módulo X-10 efectúa una transmisión, se utiliza un módulo temporizador para generar un retardo de 1ms. Durante este intervalo de tiempo de 1ms se mantiene activado el módulo *Output Compare*, con el cual se genera la portadora de 120KHz. De esta forma, se representa un “uno lógico” en el formato X-10. En la Figura 2.35 se puede ver una captura de la pantalla de un osciloscopio en la que se muestra la señal portadora de 120KHz transmitida durante 1ms.

### 2.3.4. Módulo Output Compare

El módulo *Output Compare* utiliza uno de los módulos temporizadores como base de tiempo y compara su valor con un registro de comparación<sup>7</sup>. Cuando los dos valores son iguales el estado de su

<sup>7</sup>Pueden ser dos registros en algunos modos de operación

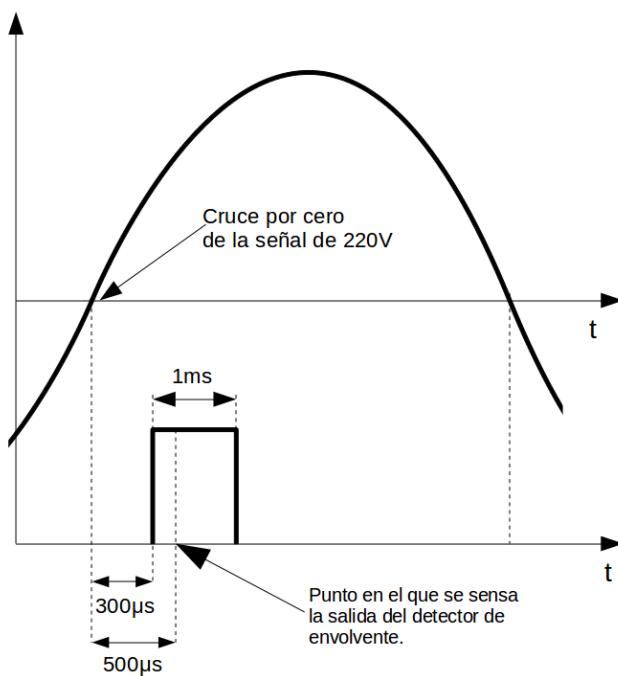


Figura 2.34: Sensado de la salida del detector de envolvente

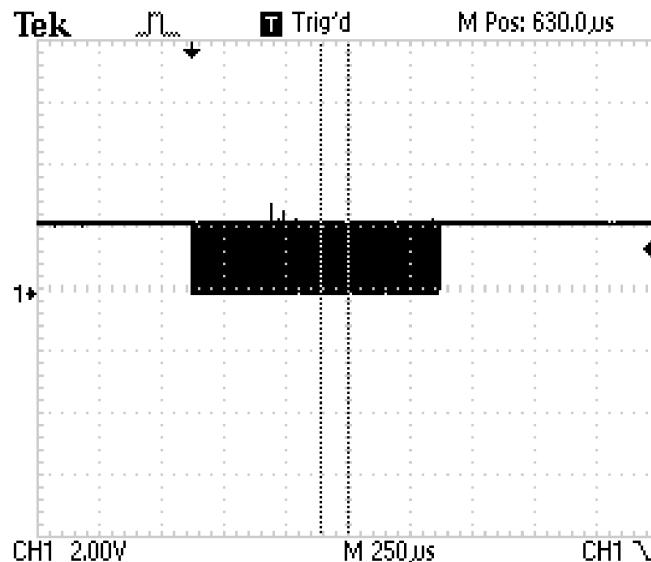


Figura 2.35: Señal portadora de 120KHz transmitida durante 1ms

salida varía. Por lo tanto, el módulo se puede utilizar para generar un pulso único, una secuencia de pulsos de salida, o bien para generar una interrupción cada vez que el valor del *timer* coincide con el de sus registros. Este posee múltiples modos de operación:

- **Modo de disparo único activo bajo** (*Active-Low One-Shot mode*).
- **Modo de disparo único activo alto** (*Active-High One-Shot mode*).
- **Modo Toggle.**
- **Modo de disparo único retardado** (*Delayed One-Shot mode*).
- **Modo de pulso continuo** (*Continuous Pulse mode*).
- **Modo PWM sin protección contra errores** (*PWM mode without fault protection*).
- **Modo PWM con protección contra errores** (*PWM mode with fault protection*).

En la Figura 2.36 se muestra un diagrama en bloques de uno de los módulos. En él se pueden ver los diferentes registros, la forma en la que están conectados entre si, y también la lógica de salida del módulo.

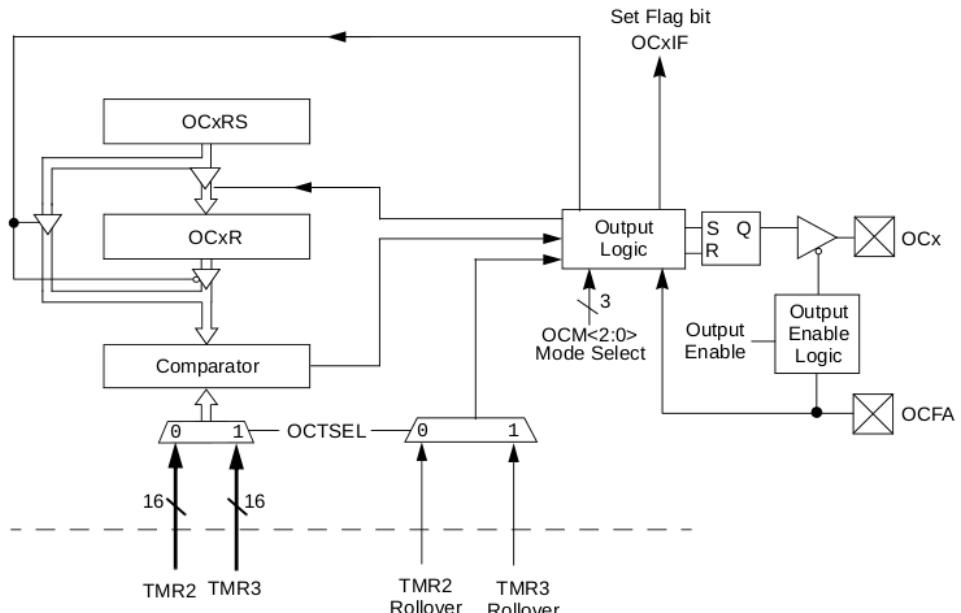


Figura 2.36: Diagrama en bloques del módulo Output compare

Para seleccionar el modo que se usará se debe modificar el valor de los bits OCM<2:0> del registro OCxCON<2:0>. En la Figura 2.37 se muestra una tabla con las posibles configuraciones de los bits OCM<2:0>, en ella también se puede ver el estado inicial del pin de salida en cada uno de los modos, y el tipo de flanco con el que se genera la interrupción en cada uno de ellos.

El módulo puede utilizar el *timer* 2 o el *timer* 3 para su base de tiempo. Esto se puede configurar modificando el bit OCTSEL del registro OCxCON<3>. El *timer* seleccionado tiene un valor inicial de cero y se incrementa con cada ciclo de reloj hasta llegar al valor que se encuentra almacenado en el registro PRy(*Period Register*). Cuando esto sucede se “resetea” a cero y comienza su cuenta nuevamente.

En la Figura 2.38 se pueden observar las formas de onda asociadas a cada uno de estas configuraciones. En la parte superior de la figura se ve como el *timer* va aumentando su valor hasta que es igual al registro OCxR. En ese momento se produce el primer evento, y aquí la salida cambia su valor en todos los modos de operación. En este instante finalizan los pulsos generados por *Active-Low One-Shot*.

OCM<2:0>	Mode	OCx Pin Initial State	OCx Interrupt Generation
000	Module Disabled	Controlled by GPIO register	—
001	Active Low One-Shot mode	0	OCx Rising edge
010	Active High One-Shot mode	1	OCx Falling edge
011	Toggle mode	Current output is maintained	OCx Rising and Falling edge
100	Delayed One-Shot mode	0	OCx Falling edge
101	Continuous Pulse mode	0	OCx Falling edge
110	PWM mode without fault protection	0, if OCxR is zero 1, if OCxR is non-zero	No interrupt
111	PWM mode with fault protection	0, if OCxR is zero 1, if OCxR is non-zero	OCFA <sup>(1)</sup> Falling edge for OC1 to OC4 OCFB <sup>(1)</sup> Falling edge for OC5 to OC8

Figura 2.37: Posibles configuraciones de OCM&lt;2:0&gt;

mode y Active-High One-Shot mode que se utilizan cuando se desea obtener un pulso único (activo bajo o activo alto) de longitud variable.

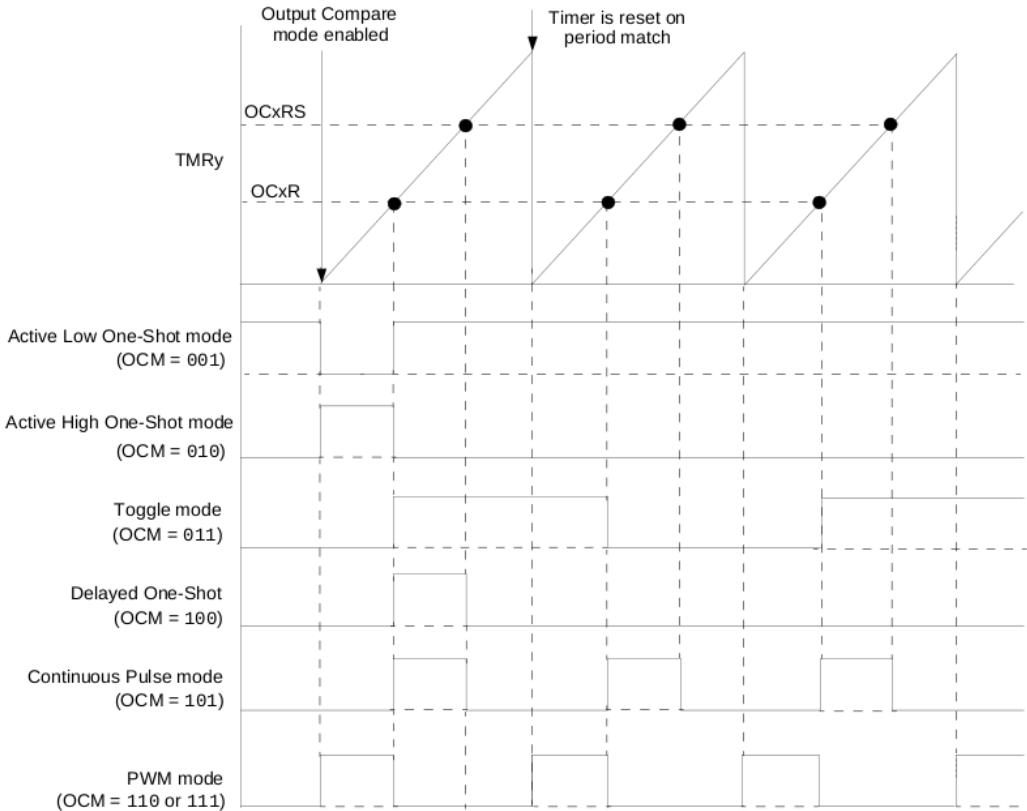


Figura 2.38: Formas de onda asociadas a cada una de las configuraciones

Luego el *timer* llega al valor de OCxRS y la salida solo se modifica en los modos *Delayed One-Shot* y *Continuous Pulse mode*. El primero es útil cuando se desea un pulso único de longitud variable que comience con un retardo programable. El segundo se utiliza cuando se desea una señal periódica con

un ciclo de trabajo programable. Este se configura en el momento de programar el módulo y no puede ser modificado luego.

Luego el *timer* continua con su cuenta hasta que llega al valor del registro PRx. En este momento se reinicia y comienza a contar desde cero, y la salida solo cambia de valor en los modos PWM que se utilizan para generar una señal cuadrada de ciclo de trabajo variable. Este puede ser variado en cualquier momento lo que lo diferencia del modo *Continuous Pulse mode*. También se debe aclarar que si el registro OCxR es configurado con el valor cero las salidas de los modos *Delayed One-Shot*, *Continuous Pulse mode* y *Toggle* cambiarían de valor.

En este proyecto se utilizó el módulo *Ouput Compare 1* en el modo *Toggle* para generar la señal de 120KHz de la portadora. En esta configuración la salida cambia de estado cuando el valor del *timer* coincide con el del registro OCxR. También se puede ver claramente que para lograr un ciclo de la señal de salida se necesita que el reloj llegue dos veces al valor de PRx.

En este caso se utilizó el *timer 2* como fuente de reloj. En primer lugar, se calculó el valor del registro PRx para obtener el período de la señal ( $8.33\mu s$ ), para lo cual se utilizó la siguiente ecuación:

$$2PRx = \frac{Fcy}{F_{Portadora}} = \frac{40MHz}{120KHz} = 333,333$$

De esta forma, se obtiene la cantidad de ciclos de máquina que entran en un ciclo de la portadora, pero como se dijo antes en el modo *toggle*, el reloj se debe “resetear” dos veces por cada ciclo de la señal de salida. Por lo tanto:

$$PRx = \frac{333,333}{2} = 166,5$$

Se utilizó un valor de 167. Luego se configuró el registro OCxR con el valor cero para que cuando se activa el módulo se envíe la señal sin ninguna demora. Como se explicó en secciones anteriores el máximo retardo entre el cruce por cero de la red y la transmisión de un “uno lógico” debe ser menor a  $300\mu s$ . En la Figura 2.39 se puede ver una captura de la pantalla de un osciloscopio en donde se muestra la señal de la portadora generada.

## 2.4. Descripción del software asociado

El software de este proyecto fue programado íntegramente con MPLAB V8.10 y la versión para estudiantes del compilador C30 V3.01. El software esta compuesto por dos archivos X-10.h y X-10.c. El primero es un archivo del tipo *header* en donde están almacenados los códigos de casa y los códigos numéricos sin el sufijo. El segundo es el archivo que contiene el código fuente en C30.

Este es un módulo transceptor, por lo tanto puede utilizarse tanto para transmitir señales a la línea y dar órdenes a otros módulos, como para recibir dichas señales y ejecutar órdenes. Como se explicó en la Sección 2.3, las funciones del microcontrolador son distintas en los dos casos, por lo tanto el programa que ejecuta el microcontrolador varía de acuerdo a la configuración del módulo.

### 2.4.1. Programa principal

En la Figura 2.40 se puede ver el diagrama de flujo del programa principal (X-10.c). En primer lugar se debe configurar las funciones principales del microcontrolador. Para esto se utilizan los bits de configuración:

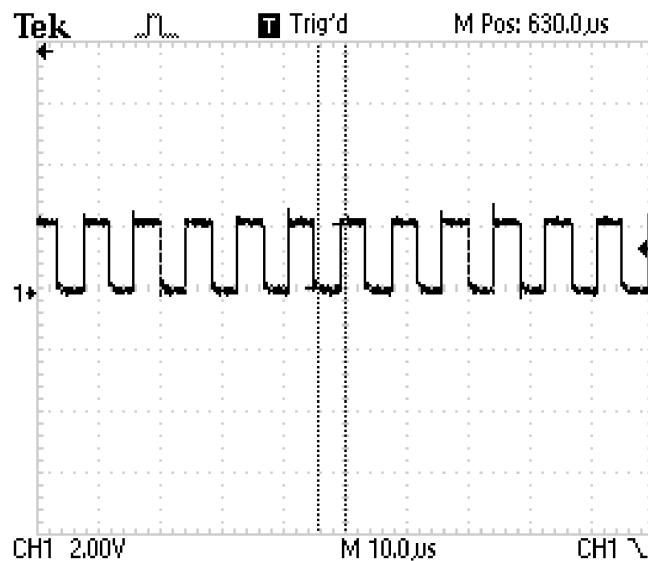


Figura 2.39: Señal de portadora generada con el módulo

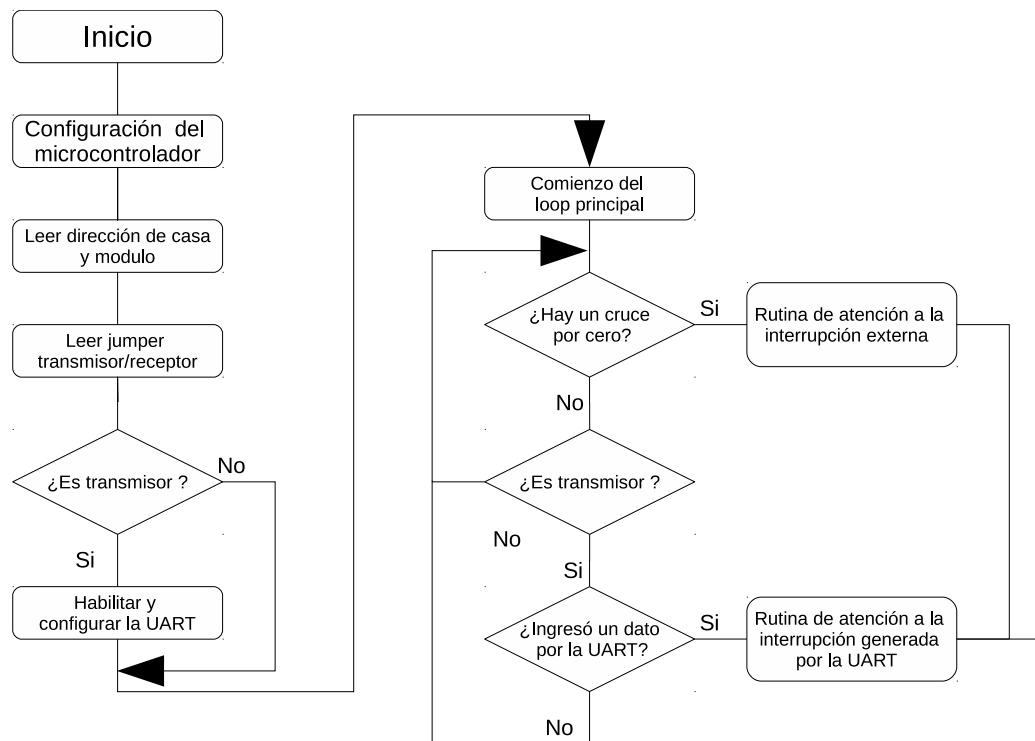


Figura 2.40: Diagrama de flujo del programa principal

```
_FOSCSEL(FNOSC_PRIPLL)
_FOSC(POSCMD_XT & FCKSM_CSECMD & OSCIOFNC_OFF)
_FWDT(FWDTEN_OFF)
```

En la primera línea se habilitó el oscilador primario con PLL, y luego en la segunda línea se habilitó el cambio de *clock*. Por último se deshabilitó el *watchdog timer*. Luego se debe realizar la configuración del lazo de enganche de fase y la operación de cambio de *clock*:

```
PLLFBD = 0x20;
CLKDIV = 0x0000;
-- builtin_write_OSCCONH(0b011);
-- builtin_write_OSCCONL(0x01);
while(OSCCONbits.OSWEN == 1);
```

En las dos primeras líneas se definen los valores de N1, N2, y M ( ver Sección 2.3.1) que se cargan en los registros PLLFDB (M) y CLKDIV (N1 y N2), y luego se efectúa el cambio de *clock*, para lo cual se debe realizar la secuencia de desbloqueo para permitir la escritura del byte alto del registro OSCCON. Esto se realiza con el comando “*\_builtin\_write\_OSCCONH*”. Luego se debe escribir el valor apropiado en los bits de control NOSC (OSCCON<10:8>) para la nueva fuente de *clock* en este caso “0b011” (oscilador primario con PLL).

En segundo lugar, se debe desbloquear la parte baja del registro OSCON para fijar el bit OSWEN (OSCON<0>) y así iniciar el cambio de oscilador. Por último, se debe esperar a que el microcontrolador coloque un cero en el bit OSWEN lo que indica que el cambio fue realizado satisfactoriamente. Esto se hizo con la sentencia *while* que se ve en la última línea del código mostrado anteriormente.

El paso siguiente es la configuración de los módulos temporizadores. Para ilustrar como se realizó, se tomará como ejemplo la configuración del *timer2* que se utiliza como base de tiempo del módulo *Output Compare*:

```
T2CONbits.T32=0;
T2CONbits.TCKPS=0b00;
T2CONbits.TCS=0;
T2CONbits.TGATE=0;
PR2=167;
```

Para realizar esta tarea se deben utilizar dos registros. En primer lugar el TxCON ( en este caso T2CON) que es el registro de control del *timer* 2, y en segundo lugar el registro PRx (PR2) en donde se almacena el valor del período del *timer* (calculado en la Subsección 2.3.3).

En la primera línea del código se coloca un cero en el bit T32 de T2CON. Con esto se hace que el *timer* trabaje en el modo de 16 bits, debido a que la cuenta a la que tiene que llegar es 167, lo que hace que utilizar 16 bits sea mas que suficiente.

En la segunda línea se configura a los bits TKPS de T2CON en cero. Esto hace que el *prescaler* pase a tener el valor 1:1, y de esta forma no se divide a la frecuencia del reloj. El objetivo de esto es lograr una mayor resolución en el período de la señal de la portadora generada por el módulo *Output Compare*. Luego, se coloca un cero en el bit TCS y el bit TGATE de T2CON para hacer que la fuente de *clock* del *timer* sea el oscilador interno (Fcy). Por último, se carga el valor del período del *timer* en PR2.

Una vez configurado el módulo temporizador se puede pasar a la configuración del módulo *Output Compare*. Esto se realizó con el siguiente código:

```
OC1CONbits.OCTSEL=0;
OC1R=0;
OC1CONbits.OCM=0b011;
```

Aquí se cargan los registros de la forma explicada en la Sección 2.3.4. En primer lugar se selecciona la fuente de reloj (*timer2*), y luego se ingresa el valor cero en el registro OC1R. Por último se configura al módulo en el modo toggle.

A continuación se realiza el mapeo de pines del integrado, para que la distribución de pines del PIC se ajuste a lo explicado en la Subsección 2.3.2. La selección de pines se divide en pines que tienen la función de entrada y pines de salida.

Para el mapeo de pines con la función de entrada se utiliza el registro RPINRx, que contiene un campo de 5-bits para seleccionar el pin RPx asociado al periférico. En cambio, la selección de pines con la función de salida es inversa, con la utilización del registro RPORx se asigna un periférico a un pin determinado. Además tiene la opción NULL (00000) que permite dejar desconectado el pin (sin selección).

Por otro lado se debe tener en cuenta algo muy importante, durante la operación normal del dispositivo, la operación de escritura de los registros RPINRx y RPORX son deshabilitadas. Para desbloquear esta protección es necesario actuar sobre el bit IOLOCK (OSCCON<6>). Esto se realiza con la primera línea del código que se ve a continuación, y debe hacerse antes de realizar cualquier operación de asignación de pines:

```
--builtin_write_OSCCONL(OSCCON & ~(1<<6));

//salidas:

RPOR7bits.RP15R=0b10010;
RPOR6bits.RP13R=0b00011;
RPOR5bits.RP11R=0b00100;

//entradas:

RPINR18bits.U1RXR=14;
RPINR18bits.U1CTSR=12;

--builtin_write_OSCCONL(OSCCON | (1<<6));
```

Después de desbloquear los registros, se asignó la salida del módulo *Output Compare 1* al pin RP15, y las salidas *Transmit* y *Request To Send* de la UART1 a los pines RP13 y RP11 respectivamente. Esto se realizó asignando el código binario correspondiente a cada salida en los registro RPORx como fue explicado anteriormente.

El paso siguiente fue mapear las entradas *Receive* y *Clear To Send* en los pines RP14 y RP12 utilizando los registros RPINRx. Por último, se procedió a bloquear la escritura de los registros nuevamente.

Luego de configurar al microcontrolador, se procede a leer los ocho *jumpers* que se utilizan para que el usuario pueda asignar una dirección de casa y un número de unidad al módulo. Estas direcciones serán utilizadas solo cuando el módulo se configure como actuador bidireccional. Para realizar esta acción se leen los valores de las entradas conectadas a los *jumpers* secuencialmente. Se utilizan dos

variables enteras de 8 bits para guardar los datos leídos en formato complementario. Si el valor presente en la entrada es un “uno lógico”, se almacena un valor de “10” en las posiciones correspondiente a ese bit de la dirección. Por otra parte, si el valor leído en la entrada es un “cero lógico” se almacena el valor “01” en la posiciones correspondientes a ese bit. De esta forma se convierten las direcciones de 4 bits al formato complementario que utiliza X-10.

El último *jumper* que se lee es el que se utiliza para indicarle al microcontrolador si el módulo formará parte de la interfaz 802.3/X-10 (transmisor), o si se utilizará como actuador bidireccional (receptor). Este valor se empleará para decidir que partes del programa se ejecutarán.

El próximo paso es la configuración de la UART número 1. En el diagrama de flujo del programa principal (Figura 2.40) se puede ver que la UART es habilitada y configurada solo si el módulo es utilizado como transmisor. A continuación se muestran los pasos para realizar dicha configuración:

```
U1BRG=259;  
U1MODEbits.PDSEL=0b00;  
U1MODEbits.STSEL=0;  
U1MODEbits.RTSMD=0;  
IEC0bits.U1RXIE=1;  
U1STAbits.URXISEL=0b00;  
U1MODEbits.UARTEN=1;  
U1STAbits.OERR= 0;  
_U1RXIF= 0;
```

En primer lugar se configuró la tasa de transmisión con el registro U1BRG, se optó por un valor de 9600 bps. Luego se emplearon los bits PDSEL y STSEL del registro U1MODE para utilizar palabras de 8 bit sin paridad y con un solo bit de *stop*. Con este mismo registro se habilitó el control de flujo, para ello se utilizó el bit RTSMD. Esto hace que se puedan utilizar los terminales *Clear To Send* y *Request To Send* para controlar el ingreso y egreso de datos de la UART.

Luego se colocó un “uno lógico” en el bit U1RXIE del registro IEC0 para habilitar la interrupción. Con los bits URXISEL del registro U1STA se le ordena a la UART que interrumpa al programa principal cada vez que ingresa un dato. Por último, se habilita la UART colocando un uno en el bit UARTEN, se borra el *flag* de *Overrun error* y el de la interrupción para que la misma pueda producirse.

En este punto del programa también se realiza una acción muy importante. Si el módulo se configuró como transmisor se deshabilita la señal de entrada al filtro pasobanda colocando un cero en la salida RB9. Se concluye con la configuración del microcontrolador inicializando las variables que se utilizarán.

Cuando el microcontrolador entra en el *loop* principal permanece destellando un led hasta que el programa es interrumpido. Esto puede ser originado por un cruce por cero, o (en el caso del transmisor) porque ingresa un dato a la UART. En las siguientes subsecciones se explicarán las rutinas que se ejecutan en ambos casos.

#### 2.4.2. Rutina de atención a la interrupción producida por la UART

Como se mencionó en secciones anteriores, cuando el módulo es configurado para formar parte de la interfaz 802.3/X-10 se comunica con el módulo Ethernet mediante la UART. Cuando el usuario de la red domótica envía una orden por medio de la página web, el módulo Ethernet envía dos palabras de 8 bits al módulo X-10. La primera contiene la dirección del módulo actuador que recibirá la orden y la ejecutará. Los cuatro bits más significativos de la palabra se utilizan para enviar el código de casa

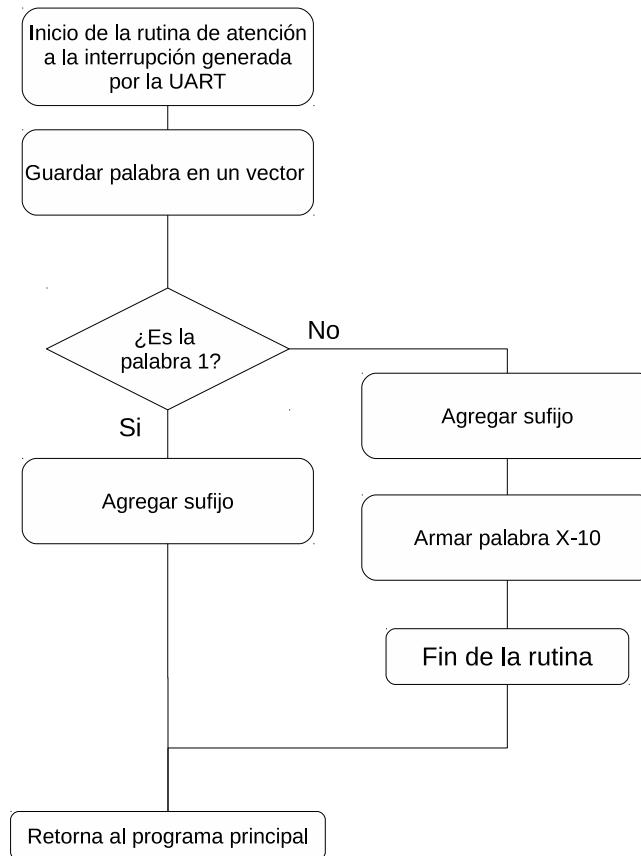


Figura 2.41: Diagrama de flujo de la rutina de interrupción de la UART.

del actuador. Mientras que los cuatro bits menos significativos son utilizados para transmitir el código numérico.

En la segunda palabra de 8 bits se envía nuevamente el código de casa, y los cuatro bits menos significativos se utilizan para enviar el código numérico de la orden que se desea ejecutar. Los códigos numéricos de ambas palabras se envían sin el sufijo (bit menos significativo ver Sección 2.1). Se realiza de esta forma para que las palabras sean de 8 bits, y porque el bit del sufijo se puede agregar en el momento de la recepción.

En la Figura 2.41 se puede ver el diagrama de flujo de la rutina de atención a la interrupción producida por la UART, y a continuación se muestra el código de la rutina:

```

void __ISR __attribute__((interrupt)) _U1RXInterrupt (void){
    int i,mascara,dato;
    dato=U1RXREG;
    if (parte==0){
        mascara=128;
        for (i=0;i<8;i++){
            palabra1[i]=(dato&mascara)?1:0;
            mascara=mascara/2;
        }
    }
}
  
```

```
palabra1[8]=0;
parte=1;
}
else{
mascara=128;
for (i=0;i<8;i++){
    palabra2[i]=(dato&mascara)?1:0;
    mascara=mascara/2;
}
palabra2[8]=1;
parte=0;
recibido=1;
PORTBbits.RB11=1;
}
U1STAbits.OERR= 0;
_U1RXIF= 0;
}
```

En la primera línea de la rutina se declaran las variables locales. Luego, se coloca el dato recibido en la variable llamada <dato>. Si no se recibió ninguna palabra la variable <parte> tiene el valor cero, por lo tanto se cumple la condición de la sentencia *if*, y se ejecuta el primer bloque de código.

En primer lugar, se guarda cada bit de la variable <dato> en un vector de nueve elementos llamado <palabra1>. Luego se agrega el sufijo para la primera palabra en el octavo elemento de <palabra1>. El valor del sufijo es siempre cero debido a que se trata de una dirección.

Cuando se recibe la segunda palabra se interrumpe por segunda vez el programa principal, el valor de la variable <parte> es uno, y se ejecuta el segundo bloque de código. Se guarda nuevamente la palabra en un vector llamado <palabra2>, y se agrega el sufijo. En este caso el sufijo es uno debido a que se trata de una orden. Por último, la rutina de atención a la interrupción coloca un uno en la salida *Request To Send* de la UART que va conectada al *Clear To Send* de la UART del transmisor para que no envie más datos, y coloca un “uno lógico” lógico en una variable llamada <recibido>, esto le indica al programa principal que hay datos para enviar por la red eléctrica.

#### 2.4.3. Función para ensamblar los bloques de datos X-10

Una vez que se recibieron las dos palabras, y se almacenaron en los vectores, se puede ensamblar los bloques de datos X-10 para ser enviados. Para realizar esta acción se invoca a una función llamada “armarpalabra”, cuyo código se puede ver a continuación:

```
void armarpalabra (int* palabra,int *cadenalista){
    int x=0;
    int i;
    for(i=4;i<22;i=i+2){
        cadenalista [i]=palabra[x];
        x++;
    }
    for(i=5;i<22;i=i+2){
        cadenalista [i]=cadenalista [i-1]^1;
```

```

    }
}

```

Esta función recibe dos parámetros. El primero de estos parámetros es un puntero (llamado <palabra>) que contiene la dirección de memoria de un vector de 9 elementos. En este vector se encuentran los datos que se desea convertir al formato X-10. En este caso se pasa como parámetro la dirección de memoria de alguno de los vectores en los que se almacenaron los datos recibidos por la UART (<palabra1> o <palabra2>).

El segundo parámetro es un puntero (llamado <cadenalista>) que contiene la dirección de memoria de un vector de 22 elementos<sup>8</sup>. En este vector se almacenará el bloque de datos de X-10 creado por la función. Las primeras cuatro posiciones de <cadenalista> contienen el *Start Code*(1110) del bloque X-10.

El primer lazo *for* de la función se utiliza para guardar los datos del vector <palabra> en las posiciones impares de <cadenalista>. Se debe comenzar a almacenar los datos en la quinta posición de <cadenalista> para que el primer dato quede después del *Start Code*.

Con el último lazo *for* se recorren todas las posiciones impares del vector <cadenalista> (comenzando desde la quinta). Mediante una operación “or exclusiva”, se calcula el complemento de los valores almacenados en cada una de las posiciones, y se los almacena en la posición siguiente. Por ejemplo, si el quinto elemento de <cadenalista> es un “uno lógico” se almacenará un “cero lógico” en la sexta posición.

El resultado de la función es un vector de 22 elementos que contiene un bloque de X-10 compuesto por el *Start Code*, seguido por un código de casa y un código numérico, ambos en formato complementario. En la rutina de atención a la interrupción externa se utilizarán dos de estos vectores, uno para transmitir el primer bloque de datos X-10 (al que se llamará <cadenadir>), y otro para transmitir el segundo bloque de datos (<cadenaord>).

#### 2.4.4. Rutina de atención a la interrupción externa

En la Figura 2.42 se muestra el diagrama de flujo de la rutina de atención a la interrupción externa. Cada vez que se produce un cruce por cero de la señal de la red eléctrica, se genera un flanco en la entrada RB7 (Ver Subsección 2.2.1). Cuando ocurre esto el microcontrolador interrumpe el programa principal para ingresar en esta rutina. Como se puede ver en el diagrama de flujo, la rutina varía dependiendo de la configuración del módulo.

Si se lo utiliza como parte de la interfaz 802.3/X-10<sup>9</sup>, se encargará de enviar los datos a los módulos actuadores, y luego será el encargado de recibir la respuesta con la confirmación de los mismos. Cuando se utiliza el módulo como actuador bidireccional, la rutina trabajará de manera inversa a la anterior, en primer lugar, recibirá las órdenes y por último enviará la confirmación al transmisor.

El primer bloque de datos X-10 de la confirmación está compuesto por el *start code*, la dirección de casa, y el código numérico del actuador que recibió la orden. Mientras que el segundo bloque de datos esta compuesto por el *start code*, la dirección de casa del actuador que recibió la orden, y el código numérico del *Hail Acknowledge*(10011).

Esta parte del software fue programada como una máquina de estados. En la parte izquierda de la Figura 2.42, se puede ver la rama del transmisor. Cuando la UART termina de recibir los datos

<sup>8</sup>Para simplificar la explicación se llamará a los vectores con el mismo nombre que los punteros pasados como parámetros

<sup>9</sup>Para simplificar, en el diagrama de flujo se lo denominó transmisor

provenientes del módulo Ethernet, y se ensamblan los bloques de datos X-10, se coloca el valor cero en una variable llamada <estado>. De esta forma, se le indica a la rutina que debe comenzar a transmitir datos.

En primer lugar, se debe transmitir el primer bloque de datos dos veces, este bloque se encuentra almacenado en el vector llamado <cadenadir>. Cada vez que se produce un cruce por cero, se lee un bit del vector y se lo transmite utilizando el método explicado en la Subsección 2.4.5. Luego de dejar tres ciclos de la señal de línea sin transmitir información, se repite la operación con el segundo bloque de datos, que se encuentra almacenado en el vector <cadenaord>.

Una vez que el módulo termina de transmitir los datos, se habilita la entrada del filtro pasobandas para recibir la confirmación. En primer lugar, se recibe el código de comienzo del primer bloque de datos, para esto se utiliza el método explicado en la Subsección 2.4.6. Si esta parte del bloque no es correcta, se comienza a transmitir la orden de nuevo. Si lo es, se continua con el siguiente estado.

El siguiente paso es recibir y almacenar el código de casa del bloque de datos en un vector llamado <recibidadadir1> . Luego se debe comprobar que el código recibido sea correcto, estos dos pasos se explican con detalle en la Subsección 2.4.7.

De una manera muy similar a la explicada en la Subsección 2.4.7, se recibe el código numérico del primer bloque de datos, y se comprueba que coincida con el del actuador al que se le envió la orden. Si el código numérico es incorrecto se comienza a transmitir nuevamente. Si es correcto se comienza a recibir y verificar por segunda vez el primer bloque de datos, este es almacenado nuevamente en el vector <recibidadadir1>.

Si los datos del primer bloque son correctos por segunda vez, se debe esperar que pasen tres ciclos de la señal de red. Luego se debe recibir y verificar el segundo bloque de datos dos veces. Si el código numérico de este bloque coincide con el *Hail Acknowledge* (10011), se coloca un cero en la salida *Request To Send* de la UART para habilitar la recepción de datos nuevamente. En este punto se finaliza con la rutina.

En la parte derecha de la Figura 2.42, se puede observar el diagrama de flujo de la rutina de atención a la interrupción que utiliza el módulo cuando está configurado como actuador bidireccional. El procedimiento para recibir la orden es similar al que se utiliza para recibir la confirmación en la rama del transmisor. La única diferencia, es que en este caso se compara el código de casa y el código numérico recibido con las direcciones programadas por el usuario con los *jumpers*. Si cualquiera de estas direcciones no coinciden con las programadas, se vuelve al primer estado, en el que se espera al código de *start*.

Una vez que se recibieron los dos bloques X-10 se decodifica la orden. Si se le ordena al módulo que active su salida, se coloca un “uno lógico” en la salida RB6 del microcontrolador. Este terminal comanda el relé que alimenta con 220V la salida del actuador. Si se recibe la orden de desactivar la salida, se coloca un “cero lógico” en RB6 y se desactiva el relé.

Una vez que se recibió y se ejecutó la orden, se ensamblan los dos bloques X-10 para enviar la confirmación al módulo transmisor en la próxima interrupción. Otra acción que se realiza antes de enviar la confirmación es desactivar la entrada de señal del filtro pasobandas colocando un cero en la salida RB9. El procedimiento para enviar estos bloques, es el mismo que utiliza el módulo para enviar las órdenes, cuando está configurado como transmisor.

Una vez que se terminan de enviar los datos de la confirmación, se concluye con la rutina, y el módulo queda a la espera de nuevos datos. Para finalizar, se debe modificar el tipo de flanco con el cual se activa la interrupción externa (flanco positivo o negativo), de lo contrario no se activará en el próximo semi-ciclo (ver Sección 2.2.1), esta es la última acción que se realiza cada vez que se ingresa a la rutina.

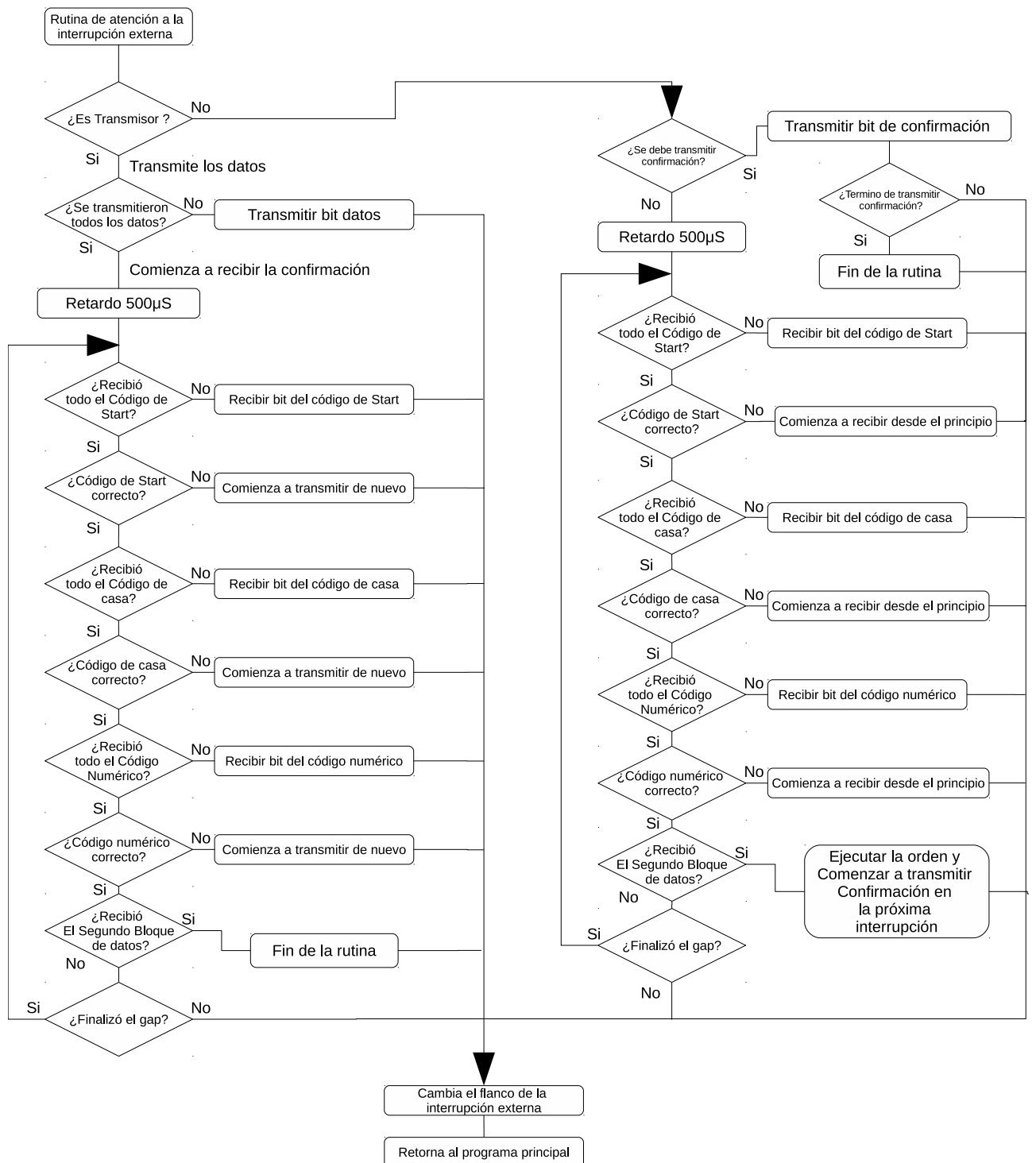


Figura 2.42: Rutina de atención a la interrupción externa

### 2.4.5. Transmisión de datos

A continuación se muestra el código utilizado para transmitir un bloque X-10 a la red eléctrica, este se usará como ejemplo para explicar como se envían los datos:

```
if (estado==0){  
    if (cursor<22)  
    {  
        if (cadenadir[cursor]==1)  
        {  
            enviaruno();  
        }  
        cursor++;  
    }  
    else{  
        estado=1;  
        cursor=0;  
    }  
}
```

Esta parte del código corresponde al primer estado de la rutina de atención a la interrupción de la rama del transmisor<sup>10</sup>. La variable <estado> tiene el valor cero, debido a que es el comienzo de la rutina, por lo tanto, el microcontrolador ejecutará esta sección del código. En primer lugar, se lee el valor de la variable <cursor> que se utiliza para desplazarse por el vector que contiene el bloque X-10.

Si el valor de <cursor> es menor a 22, se lee un bit del vector <cadenadir> que contiene el primer bloque X-10. Si el valor de ese bit es un “uno lógico”, se llama a la función “enviaruno”, que activa el módulo *Output Compare* para que genere la portadora de 120KHz por el lapso de 1ms. Por último se incrementa la variable <cursor>.

Si el valor presente en el vector es un “cero lógico”, solamente se incrementa la variable <cursor>. Cuando el valor de <cursor> es 22, se terminó de transmitir el bloque de datos. Se pasa al estado siguiente incrementando la variable <estado> y colocando un “cero lógico” en la variable <cursor>. El método explicado anteriormente se utiliza para transmitir cada uno de los bloques X-10.

Luego de transmitir un bloque de datos dos veces, se debe dejar una separación de tres ciclos antes de transmitir los dos próximos bloques, esto se realiza con el siguiente código:

```
if (estado==2)  
{  
    if (cursor<6)  
    {  
        cursor++;  
    }  
    else  
    {  
        estado=3;  
        cursor=0;  
    }  
}
```

---

<sup>10</sup>Como se dijo anteriormente, para simplificar se denomina transmisor al módulo configurado como parte de la interfaz 802.3/X-10

En este caso solo se incrementa la variable <cursor>. Después de tres ciclos de la señal de 220V, el valor de <cursor> es 6, por lo tanto, cuando se produzca la siguiente interrupción se pasará al próximo estado.

#### 2.4.6. Recepción del código de start

Durante la recepción, los cuatro primeros estados se utilizan para recibir el *start code*. A continuación se muestra el código del primero de ellos:

```
if (estado==5){
delay500useg();
if (PORTBbits.RB10==1){
    estado=6;
    recibidadir1[cursor]=PORTBbits.RB10;
    cursor++;
}
else{
    estado=0;
    cursor=0;
}
}
```

Cuando se produce el cruce por cero, la rutina llama a la función que produce el retardo de  $500\mu s$ , luego sensa el pin RB10 que está conectado a la salida del detector de envolvente. En este estado se espera el primer “uno lógico” del *start code*. Si el valor recibido es correcto, se lo almacena en el vector <recibidadir1>, se incrementa la variable <cursor> que utiliza para recorrer el vector, y se pasa al estado siguiente.

Si el valor recibido es incorrecto, se coloca el valor cero en la variable <estado>. Esto provoca que en la siguiente interrupción se comience a transmitir la orden iniciado el proceso nuevamente.

Este procedimiento es el que se utiliza para recibir los cuatro bits del *start code* en los primeros cuatro semi ciclos. Por lo tanto, si cualquiera de los bits del *start code* de la confirmación es incorrecto se comienza a transmitir la orden nuevamente.

#### 2.4.7. Recepción del código de casa y el código numérico

En el siguiente código se muestra el procedimiento para recibir y chequear el código de casa de un bloque de datos X-10:

```
if (estado==9){
delay500useg();
if (cursor<12) {
    recibidadir1[cursor]=PORTBbits.RB10;
    cursor++;
}
if (cursor==12){
    estado=10;
    for (i=4;i<12;i++){
        igual= recibidadir1[i]^cadenadir[i];
```

```
    if (igual==1){
        estado=0;
        cursor=0;
        break;
    }
}
}
```

Cuando se produce el cruce por cero, la rutina llama a la función que produce el retardo de  $500\mu s$ . Si el valor de `<cursor>` es menor a 12, todavía hay datos por recibir. Por lo tanto, se sensa el pin RB10, y se almacena el valor recibido en una posición del vector `<recibidadir1>11`. Luego se incrementa la variable `<cursor>` que se utiliza para recorrer el vector.

Cuando el valor de `<cursor>` es 12 se terminó de recibir la totalidad de los bits del *house address*, y se debe verificar que esta parte del bloque recibido sea correcta. Esto se realiza con el bloque de código que se encuentra dentro de la segunda sentencia `if`. Los códigos de casa se encuentran almacenados entre el cuarto y el onceavo elemento de cada uno de los vectores que contienen los bloques X-10. En este caso se comparan los elementos del vector `<recibidadir1>` y `<cadenadir>` (en el que se encuentra el primer bloque transmitido). Se recorren los dos vectores con el lazo `for`, y se realiza una operación “or exclusiva” entre los elementos de los mismos, el resultado de esta operación se almacena en la variable `<igual>`. Si los bits de los códigos de casa son iguales en ambos vectores, el resultado de la operación “or exclusiva” siempre será cero, y se pasará a recibir el código numérico. Si alguno de los bits no coinciden, la variable `<igual>` tomará el valor uno, se colocará un cero en la variable `<estado>`, se finalizara con el lazo `for` utilizando la sentencia `break` y se comenzará a transmitir nuevamente.

El procedimiento para la recepción y verificación de los códigos numéricos es muy similar. La única variante, es que los datos del código numérico son almacenados entre la doceava y la vigesimosegunda posición del vector en el que se encuentra almacenado el bloque X-10.

## 2.5. Conclusiones

En la Sección 2.1 se presentó en detalle al protocolo X-10, se explicó en detalle la manera en la que se envían datos por la línea eléctrica, y como están compuestos los bloques de datos que utiliza el protocolo. Esto sirvió como base teórica para realizar la explicación del funcionamiento del hardware y el software del módulo X-10.

En la Sección 3.2 se describió a cada uno de los siguientes circuitos que componen el módulo:

**-Detector de cruce por cero:** Este es el circuito que se encarga de sincronizar al módulo con los cruces por cero de la señal de 220V. Para lograr esto genera una señal cuadrada cuyos flancos (positivos y negativos) coinciden con los cruces por cero. Para proteger a los componentes sensibles del módulo (microcontrolador y otros componentes), y principalmente para garantizar la seguridad del usuario se utilizó un opto-acoplador para sensar la señal de red aislado de la misma.

**-Etapa de entrada/salida:** La etapa de entrada/salida es el circuito que interactúa con la red eléctrica. Se utiliza para inyectar la señal de X-10 cuando se transmite un mensaje, y para ingresar

---

<sup>11</sup>La posición del vector en la que se guarda el dato es la que coincide con el valor de `<cursor>`

al módulo la portadora proveniente de la línea. También posee un transistor que se utiliza para desconectar el circuito de recepción, y para proteger los componentes del mismo en el momento en el que se transmite. Se diseñó siguiendo la premisa de aislar al circuito de la línea eléctrica, la misma que se tuvo con el detector de cruce por cero.

**-Amplificador de salida:** Este circuito se utiliza para amplificar la portadora proveniente del microcontrolador. El mismo alimenta el transformador de la etapa de entrada/salida con el cual se conecta el módulo a la red de 220V.

**-El filtro pasobandas:** Debido al ruido presente en la línea eléctrica, se debe filtrar la señal recibida antes de que acceda al amplificador sintonizado del receptor. Se estudiaron varias opciones, y se concluyó que un pasobandas del tipo Chebyshev era lo más adecuado. Se trata de un filtro con una frecuencia central de 120KHz y con un ancho de banda de 3dB de aproximadamente 90KHz.

**-Amplificador sintonizado y detector de envolvente:** Este circuito sirve para amplificar y demodular la señal recibida por la red eléctrica con un detector de envolvente. Sirve para lograr que el microcontrolador pueda detectar la presencia de los unos o ceros lógicos presentes en el mensaje recibido. El circuito que se utilizó para realizar esta tarea es uno de los mas frecuentemente utilizados en aplicaciones de X-10, debido a que es sencillo y económico.

**-Fuente de alimentación:** Se optó por utilizar una fuente conmutada preensamblada de 25V 400 mA. Estas fuentes de alimentación presentan muchas ventajas, en primer lugar, su tamaño es muy reducido, son muy económicas, y su salida se encuentra aislada de la red eléctrica, esto hace que sean más seguras que las fuentes sin transformador que se utilizan en otras aplicaciones de X-10.

En la Sección 2.3 se justificó la elección del microcontrolador PIC24H128GP502 que se utilizó en el módulo. Su gran velocidad, su encapsulado, la gran cantidad de temporizadores que posee, sus 4 módulos de Comparación/Captura/PWM, su gran capacidad de memoria, y sus pines remapeables son algunas de las particularidades que lo convierten en el más adecuado para el proyecto. También se describieron sus características principales. Se comenzó con la descripción del oscilador, y luego se mostró como generar las señales de la portadora y los retardos con los temporizadores y el módulo *Output Compare*.

En la última sección se explicaron las partes más importantes del software con el que se encuentra programado el microcontrolador, en primer lugar, se describió la configuración del PIC24HJ128GP502. Luego se realizó una explicación detallada de la rutina de atención a la interrupción generada por la UART, que se produce cada vez que ingresa un dato del módulo Ethernet. Esta interrupción es utilizada para recibir las palabras que contienen las órdenes enviadas por el usuario, y ensamblar los bloques de datos X-10 que se enviarán por la línea eléctrica.

Por último, se realizó una explicación detallada de rutina de atención a la interrupción externa, que se produce cada vez que hay un cruce por cero de la señal de 220V. Esta rutina está programada como una máquina de estados, se utiliza para enviar y recibir datos. Si el módulo se utiliza como parte de la interfaz 802.3/X-10, se encargará de enviar los datos a los módulos actuadores, y luego será la encargada de recibir la respuesta con la confirmación de los mismos. Cuando se utiliza el módulo como actuador bidireccional, la rutina trabajará de manera inversa a la anterior, en primer lugar, recibirá las órdenes y por último enviará la confirmación al transmisor.

En el siguiente capítulo se presentará en detalle el software y el hardware del módulo Ethernet, que se encarga de realizar todas las tareas relacionadas con Internet, y es la pieza que completa la interfaz 802.3 / X-10. Se trata de un servidor web que tiene alojada una página de Internet. El usuario podrá ingresar a dicha página web utilizando una computadora o un dispositivo móvil para comandar los diferentes actuadores de la red y ver el estado de los mismos.



## Capítulo 3

# El Módulo Ethernet

El módulo Ethernet es la parte de la interfaz 802.3/X-10 que tiene a su cargo todas las tareas relacionadas con Internet. Se trata de un servidor web embebido en un microcontrolador<sup>1</sup>, que se conecta a la red mediante un *router* (o un *modem*) ADSL[UNI99] y tiene alojada una página de Internet.

El usuario podrá ingresar a dicha página web utilizando una computadora<sup>2</sup>, para comandar los diferentes actuadores de la red y ver el estado de los mismos. Cuando el módulo Ethernet recibe una orden proveniente de la página web, la decodifica, y envía un mensaje al módulo X-10. Como se explicó en las secciones anteriores, este mensaje se utiliza para enviar las órdenes por medio de la red eléctrica.

El hardware del módulo está comandado por la pila TCP/IP de Microchip, que es un software especialmente diseñado para el desarrollo de aplicaciones de Internet embebidas en microcontroladores.

En la siguiente sección se comenzará con una descripción general de la pila TCP/IP. Luego, en la Sección 3.2 se describirá el hardware con el cual se implementó el módulo Ethernet. Se finalizará el Capítulo con una explicación de las modificaciones que se efectuaron en el software de la pila TCP/IP para la realización de esta aplicación (Sección 3.3).

### 3.1. La pila TCP/IP de Microchip

La pila TCP/IP de Microchip (Microchip TCP/IP Stack)[Inc02], es un software con estructura modular que implementa un conjunto importante de protocolos de Internet. Proporciona servicios básicos de red a aplicaciones basadas en microcontroladores, como por ejemplo, servidores HTTP, clientes de e-mail, entre otras. Pero no se limita solo a esto sino que también brinda soporte a aplicaciones genéricas basadas en el protocolo TCP/IP.

El concepto de pila denota una estructura en capas o niveles. Cada capa es responsable de una función determinada de la comunicación. En la Figura 3.1 se pueden ver las cuatro capas del protocolo TCP/IP, junto con la estructura del *Stack* TCP/IP. En la figura también se pueden observar los protocolos soportados por el *Stack* TCP/IP en cada una de dichas capas. Estos protocolos son: ARP[ARP], IP[IP], ICMP[ICM], UDP[UDP], TCP[TCP], DHCP[DHC], HTTP[HTT], y FTP[FTP].

El software está totalmente programado en lenguaje C, y requiere aproximadamente 32Kb de memoria de programa en el microcontrolador. Otra característica importante, es que puede ser utilizado

<sup>1</sup> “Sistema embebido” es el nombre genérico que reciben los equipos electrónicos que incluyen un procesamiento de datos, pero que a diferencia de una computadora están diseñados para satisfacer una función específica.

<sup>2</sup>O cualquier dispositivo capaz de manejar el protocolo TCP/IP.

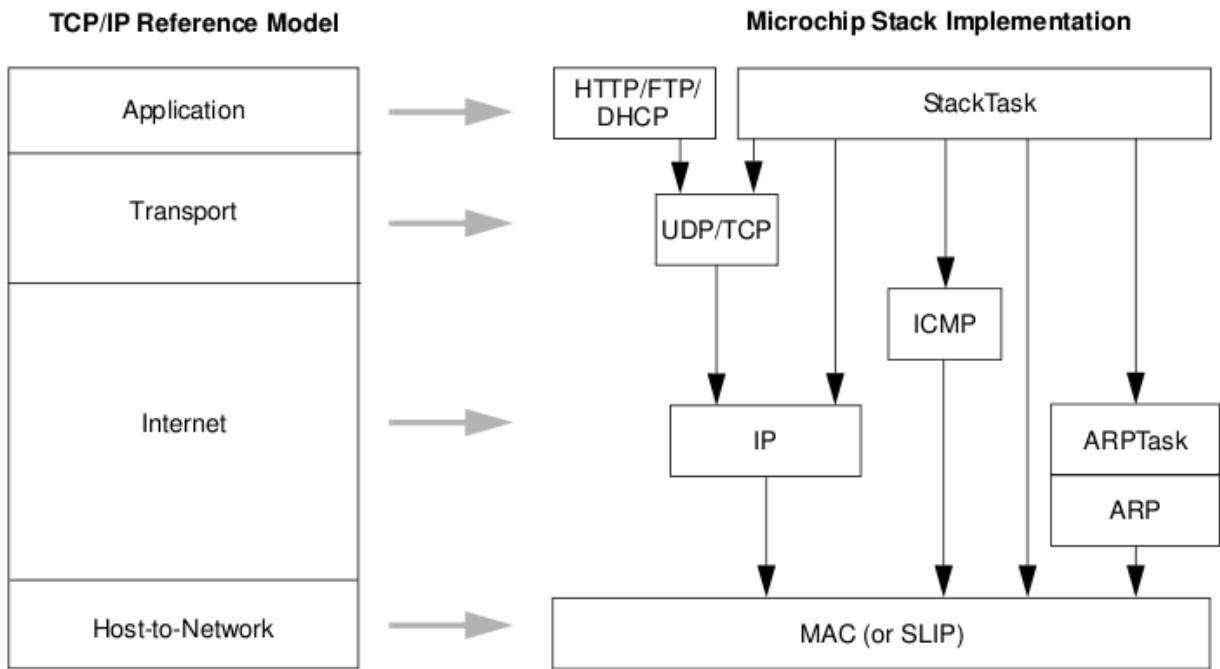


Figura 3.1: Protocolos soportados por el Microchip TCP/IP *Stack* en cada una de las capas TCP/IP

de forma gratuita en las distintas familias de Microchip. Es decir, funciona en microcontroladores de 8, 16 y 32 bits.

### 3.1.1. La estructura del stack

La pila TCP/IP proporcionada por Microchip es un software modular, altamente configurable para adaptarse a los requisitos específicos de cada aplicación. Como se mencionó anteriormente se puede utilizar con una gran cantidad de microcontroladores y DSPs de la compañía.

Para poder utilizar esta herramienta se debe instalar el programa de desarrollo MPLAB. Además, se debe descargar e instalar el *Stack* de la página de la compañía. En él se encuentran todos los programas necesarios y una serie de librerías que Microchip provee para el manejo de USB y Ethernet.

El código es actualizado frecuentemente por sus desarrolladores, y con cada nueva versión suelen añadirse funcionalidades y modificarse otras. Sin embargo, esto no supone un gran inconveniente, puesto que en general el propio código está bien documentado mediante comentarios. Si bien su análisis no es trivial, el diseño modular de la pila simplifica esta tarea, debido a que la implementación de cada protocolo, se realiza en archivos independientes poco acoplados entre sí.

Por otro lado, la mayoría de las aplicaciones prácticas no requerirán de un estudio detallado de la implementación de la pila TCP/IP. Esto se debe a que en la práctica la totalidad de los parámetros configurables son fácilmente modificables mediante directivas de preprocesador.

La pila tiene soporte directo para todos los controladores de Ethernet externos, así como también para la familia de microcontroladores PIC con controlador Ethernet integrado. La implementación de la capa MAC de estos dispositivos se realiza, como en el resto de protocolos, en archivos independientes.

En ellos se define una interfaz de programación de aplicaciones (API) que proporciona un método conveniente para la transmisión y recepción de tramas Ethernet, así como para la configuración y monitoreo del dispositivo.

En la mayoría de los casos, el acceso a estas funciones no se realiza directamente desde la aplicación de usuario, sino que queda reservado al código de otros protocolos superiores. En general, las funciones asociadas al protocolo IP sólo serán accedidas por las de los protocolos de transporte, y no por la aplicación de usuario.

El funcionamiento correcto de muchos de los protocolos implementados en la pila requiere la realización de determinadas tareas, y no sólo cuando la aplicación principal solicita alguno de sus servicios, sino como respuesta a eventos que tienen lugar de manera asíncrona.

Por ejemplo, un fragmento TCP deberá ser retransmitido si no se recibe una confirmación válida transcurrido un determinado período de tiempo. Para cumplir con estos requisitos y permanecer relativamente independiente de la aplicación, la pila TCP/IP emplea un esquema de multitarea cooperativo. Este sistema permite prescindir de un sistema operativo que gestione los tiempos de ejecución de cada tarea, puesto que son las propias tareas las que ceden el control después de realizar algunas funciones. Cualquier aplicación que utilice la pila deberá seguir este mismo esquema, ya sea repartiendo el trabajo en múltiples tareas de menor tamaño o implementando una máquina de estados. Este mecanismo se observa en el esquema típico de la Figura 3.2, donde las tareas definidas por la aplicación de usuario comparten el tiempo de ejecución con las asociadas al funcionamiento de la pila. Para ello es importante que ninguna de estas tareas monopolice el uso de la CPU. En general, será suficiente con ceder el control a la pila cada 50 ms, aunque este tiempo varía considerablemente dependiendo de los protocolos habilitados. En la Figura 3.3 se muestra una captura de pantalla del programa MPLAB.

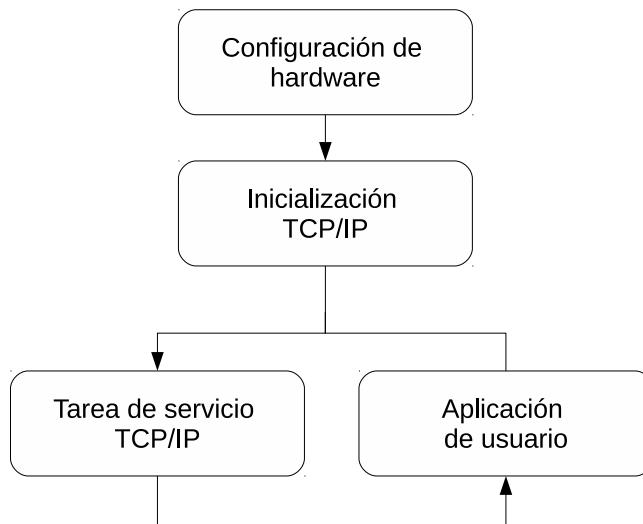


Figura 3.2: Flujo de una aplicación TCP/IP genérica

En ella se puede observar la estructura de archivos de la pila TCP/IP.

Se puede observar que la estructura dividida en dos tipos de archivos. El código fuente que se encuentra en los archivos con extensión ".c", y los archivos de cabecera que tienen la extensión ".h". Existen cuatro archivos que se destacan por sobre el resto, debido a que son los que modifica el usuario para realizar la configuración de su aplicación:

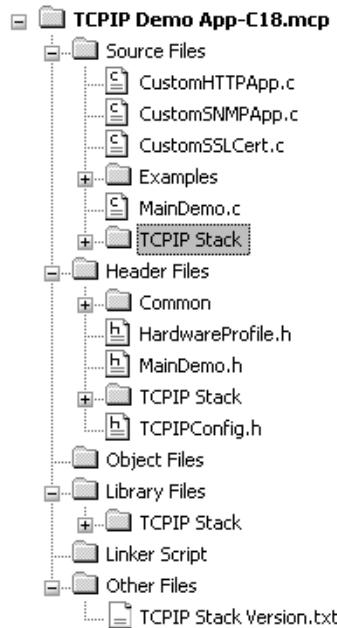


Figura 3.3: Estructura de archivos de la pila TCP/IP.

**HardwareProfile.h:** Este archivo se utiliza para configurar los denominados Perfiles de hardware. Un perfil de hardware es un conjunto de macros válidas para una pieza concreta de hardware. Con ellos se definen algunos parámetros muy importantes, como es el caso de la frecuencia de trabajo del sistema. Esto es necesario para la correcta inicialización de los temporizadores que marcan la ejecución de eventos temporales. Otra tarea importante que se realiza en la configuración del perfil de hardware, es el mapeo de terminales del microcontrolador. Aquí se definen los terminales encargados de realizar la comunicación con el controlador Ethernet, o con la memoria EEPROM externa.

**TCPIPCConfig.h:** Este archivo se utiliza para habilitar o deshabilitar funciones del *Stack* como DNS, FTP, SNTP, SSL y UART. Esto dependerá de las características de la aplicación y la de la memoria disponible en el microcontrolador. La configuración de este archivo puede realizarse de dos maneras distintas. La primera es comentando (o quitando comentarios) el código para activar o desactivar las diferentes funciones. La segunda es utilizando la aplicación “TCP/IP Configuration Wizard”. Esta herramienta está incluida en los programas que se instalan con el *Stack* y permite modificar el archivo TCPIPCConfig.h de manera gráfica.

**CustomHTTPApp.c:** En este archivo se encuentran las funciones *callback*, es decir, aquellas que interactúan con la página web. Se utiliza para configurar las variables dinámicas de la página y las funciones de *callback* con las que se actualizan dichas variables. Otra de las funciones importantes del CustomHTTPApp.c, es el procesamiento de los datos enviados a través de un formulario HTML en cualquiera de los métodos GET y POST. Con estos métodos se puede enviar información desde la página web al microcontrolador.

**MainDemo.c:** Es el programa principal, y se utiliza para configuraciones importantes, como el mapeo de puertos, la configuración de los registros de la USART, los ADC y la inicialización de la placa. En él se encuentra el bucle principal en donde se realizan las tareas principales de la pila TCP/IP. Como se mencionó anteriormente, ninguna de ellas puede monopolizar el control de la CPU durante períodos de tiempo prolongados. Se comentan a continuación algunas de las funciones más importantes contenidas en este programa:

- StackTask(): Esta es la única tarea esencial para el funcionamiento de la pila. Implementa una máquina de estados finita que, entre otras cosas, realiza las tareas temporales necesarias para el funcionamiento de las conexiones TCP. Se encarga de la retransmisión, cierre, envío de asentimientos, etc. Además, analiza el tipo de paquete recibido e invoca a los módulos correspondientes para su posterior procesamiento.
- HTTPServer(): Se trata de una aplicación TCP que responde a las peticiones realizadas desde un navegador web. Llevando el control de las conexiones establecidas.
- NBNSTask(): Esta tarea es un servidor UDP que responde a peticiones de nombre NetBIOS con la dirección IP local del controlador.
- DHCPSTask(): Se trata también de una aplicación UDP que responde a peticiones DHCP. Por defecto, este servidor asigna un tiempo de validez de la configuración DHCP de sólo 15 segundos. Por otra parte, el servidor DHCP puede ser desactivado por el usuario mediante la modificación de un bit de la estructura AppConfig. Para desactivarlo sólo es necesario no invocar a esta tarea en el bucle de ejecución.

### 3.1.2. Configuración de los servicios del stack con TCPIPConfig.h

Se puede tener una buena idea de las posibilidades del *Stack* estudiando en detalle el archivo TCPIPConfig.h. En él se encuentran todos los servicios que puede prestar la pila TCP/IP. Este archivo puede ser modificado de dos maneras diferentes, como se mencionó en la sección anterior.

Una de las formas de modificar el archivo TCPIPConfig.h es manualmente. Esto se realiza comentando líneas del programa. De esta forma, se habilitan o deshabilitan los diferentes macros definidos en el mismo. Estas macros permiten la compilación condicional de bloques de código. Esto le brinda al desarrollador la capacidad de encontrar un compromiso entre prestaciones y memoria consumida. Es importante comentar los macros de los de protocolos que no se usan, para no desperdiciar la memoria de programa del microcontrolador. A continuación se muestran algunos macros del programa con parte de los servicios más importantes del *Stack*:

```
#define STACK_USE_UDP          // Protocolo de transporte de los mensajes OSC
#define STACK_USE_ICMP_SERVER   // Servidor ICMP para respuestas de Ping
#define STACK_USE_HTTP2_SERVER  // Servidor HTTP
#define STACK_USE_DHCP_SERVER   // Servidor DHCP
#define STACK_USE_DNS            // Cliente DNS
#define STACK_USE_NBNS           // Servidor de resolución de nombres NetBIOS
```

Cuando se incluye un protocolo de nivel superior que requiere los servicios de otro protocolo no incluido explícitamente, la pila da soporte también para este último. Por ejemplo, la primera de las macros es redundante, puesto que DHCP requiere la inclusión del módulo UDP. Lo mismo ocurre con TCP, incluido de manera implícita por el servidor HTTP.

La forma más sencilla de modificar el archivo TCPIPConfig.h es mediante la utilización del “TCP/IP Configuration Wizard”. Se trata de una aplicación gráfica que permite modificar los parámetros del *Stack*. Si bien esta aplicación permite modificar la dirección IP y MAC, y seleccionar entre protocolos como HTTP y FTP, no permite que el usuario agregue su propio código. El “TCP/IP Configuration Wizard” permite agregar soporte para distintos servicios: Servidor Web, Cliente de correo electrónico, Telnet, FTP y SSL. En la Figura 3.4 se puede ver la primera pantalla del “TCP/IP Configuration Wizard”.

Lo primero que se debe hacer es seleccionar la carpeta desde la cual se va a trabajar. De esta manera, se le informa al asistente que archivo se desea modificar. El proyecto básico se encuentra en: C:\ Microchip Solutions library\ TCPIP Demo App. Si bien hay varios ejemplos realizados con el *Stack*, es conveniente empezar con este.

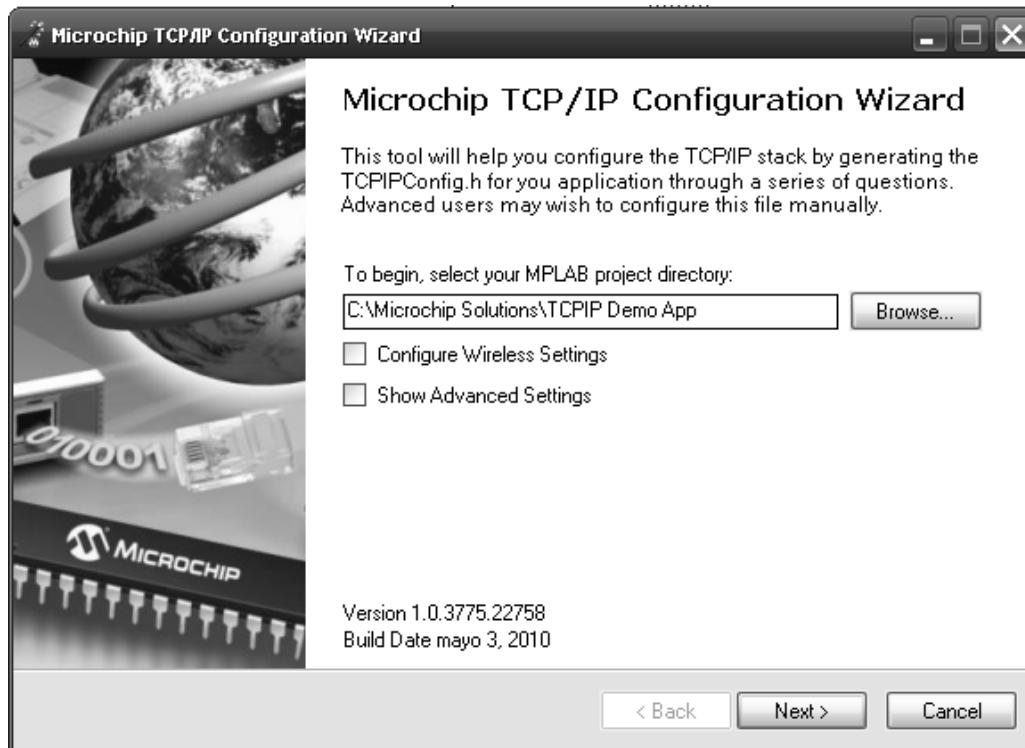


Figura 3.4: Pantalla inicial del “TCP/IP Configuration Wizard”.

Como se mencionó anteriormente, cada servicio deshabilitado por medio del asistente es comentado en el *Stack* para que no sea compilado. En la Figura 3.5, se puede ver la segunda pantalla del “TCP/IP Configuration Wizard” aquí se pueden configurar los protocolos de la capa de transporte como UDP o TCP, y los de la capa de aplicación.

No todas las funciones del *Stack* están disponibles a través del “TCP/IP Configuration Wizard”. Si no se dispone de demasiada memoria en el microcontrolador es importante acceder al archivo

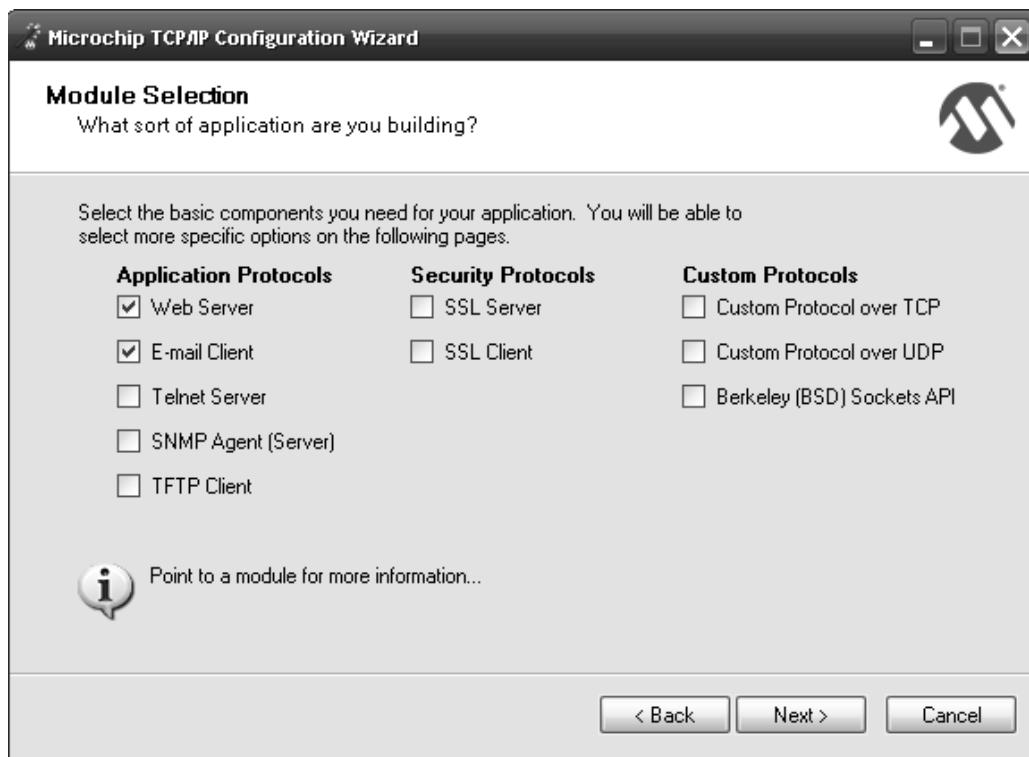


Figura 3.5: Configuración de los protocolos de la capa de transporte y aplicación

TCPIPConfig.h y comentar los servicios que no son necesarios. Por ejemplo, para deshabilitar el DNS, la instrucción debe quedar comentada de la siguiente manera:

```
//#define STACK_USE_DNS
```

Por medio del “TCP/IP Configuration Wizard” se pueden agregar funciones especiales al *Stack*. En la Figura 3.6 se puede ver la pantalla que se utiliza para la activación de estas funciones especiales.

Una de estas aplicaciones es el *Serial to Ethernet bridge*. Se trata de un conversor RS232 - Ethernet. Cuando es utilizado, todos los paquetes que ingresen por el conector de red serán retransmitidos por la UART del PIC y viceversa. De esta forma, se puede dar conectividad Ethernet a sistemas que originalmente solo tenían comunicación RS232 [Dep69].

Otra función interesante es medir el tiempo de viaje de un paquete entre el cliente y el *host*. Esto se puede realizar activando el “TCP Performance Test”. Lo que hace esta función es habilitar el uso del *ping*. De esta forma, cuando se le envía un *ping* al *Stack* este devuelve el tiempo de viaje en milisegundos.

En TCPIPConfig.h se define también la configuración de red que el controlador toma por defecto. Esta incluye el nombre de *host*, las direcciones MAC e IP locales, la máscara de subred, la IP de la puerta de enlace, y las de los servidores DNS. Estos parámetros se almacenan en memoria de programa. De esta forma, el usuario de la aplicación puede restaurar la configuración original del controlador si fuera necesario.

En la Figura 3.7 se puede ver la pantalla con la que se configuran el nombre de *host* y la dirección MAC. El nombre de *host*, se utiliza para identificar a la placa en la red. Si el servicio de DNS está

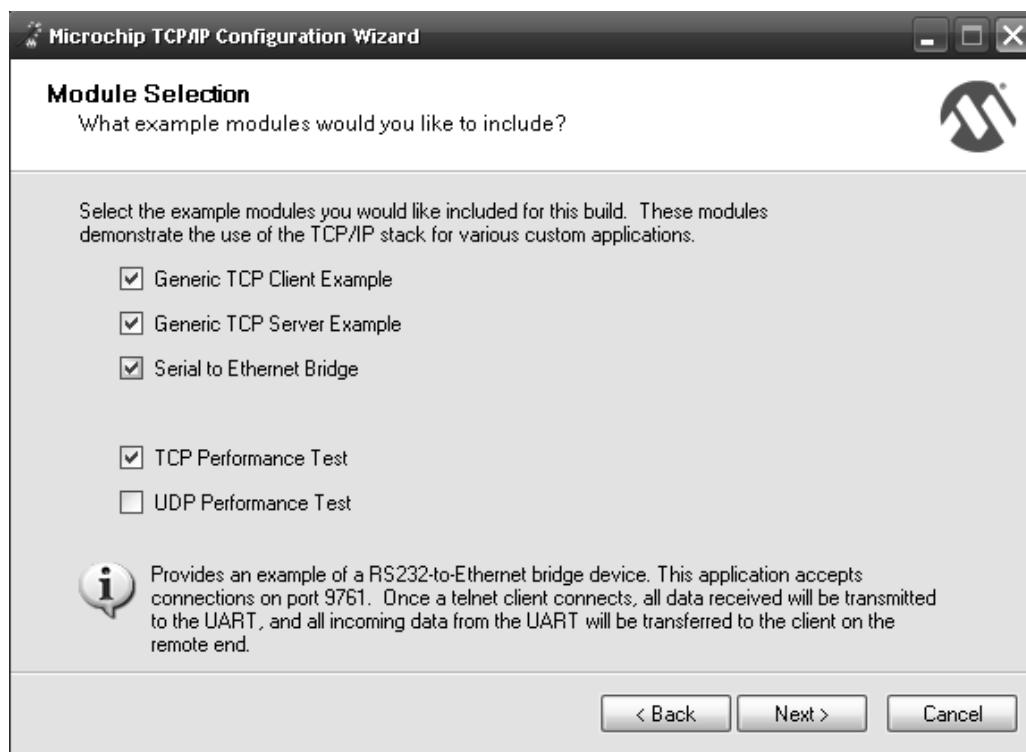


Figura 3.6: Habilitación de funciones especiales

habilitado se puede asociar la placa a este nombre, que será más fácil de recordar que la dirección IP. Por otra parte, como la IP puede ser asignada por el servidor en forma dinámica, resulta conveniente definir un nombre de *host*.

Por defecto se habilita el cliente DHCP, por tal motivo se puede conectar la placa a un *router* con DHCP habilitado y así obtener conectividad desde una computadora en la misma red. En este caso la MAC asignada es 00.04.A3.00.00.00. En la Figura 3.8 se puede ver un menú con las distintas opciones de la placa. Estas opciones son: envío de correos electrónicos, servidor HTTP y autenticación. Este último permite agregar una contraseña a la página web, y de esta forma incrementar la seguridad del sistema.

Otra de las cosas que se pueden configurar utilizando el archivo TCPIPConfig.h es el nombre de la página principal, es decir, aquella que se va a mostrar por defecto. En la Figura 3.8 se puede ver esta configuración, en este caso el nombre utilizado fue “index.htm”.

Por otra parte, es posible definir el número máximo de *sockets* UDP disponibles en la aplicación. Esto hace referencia a la cantidad de usuarios que pueden monitorizar el sistema simultáneamente. Cómo máximo pueden ser 32, pero se deben considerar parámetros como ancho de banda y estabilidad de la red antes de establecerlo. Un número alto implica la definición de más variables globales. Por lo tanto, puede no ser deseable en microcontroladores con poca memoria RAM. También se debe tener en cuenta que una excesiva demanda puede deteriorar el rendimiento del sistema.

Cuando se utiliza TCP, no sólo es posible especificar el número de *sockets*. También se puede variar la manera en la que se asigna la RAM disponible a sus *buffers* asociados de transmisión y recepción. Es importante asegurar la disponibilidad de un *socket* TCP por cada conexión HTTP que

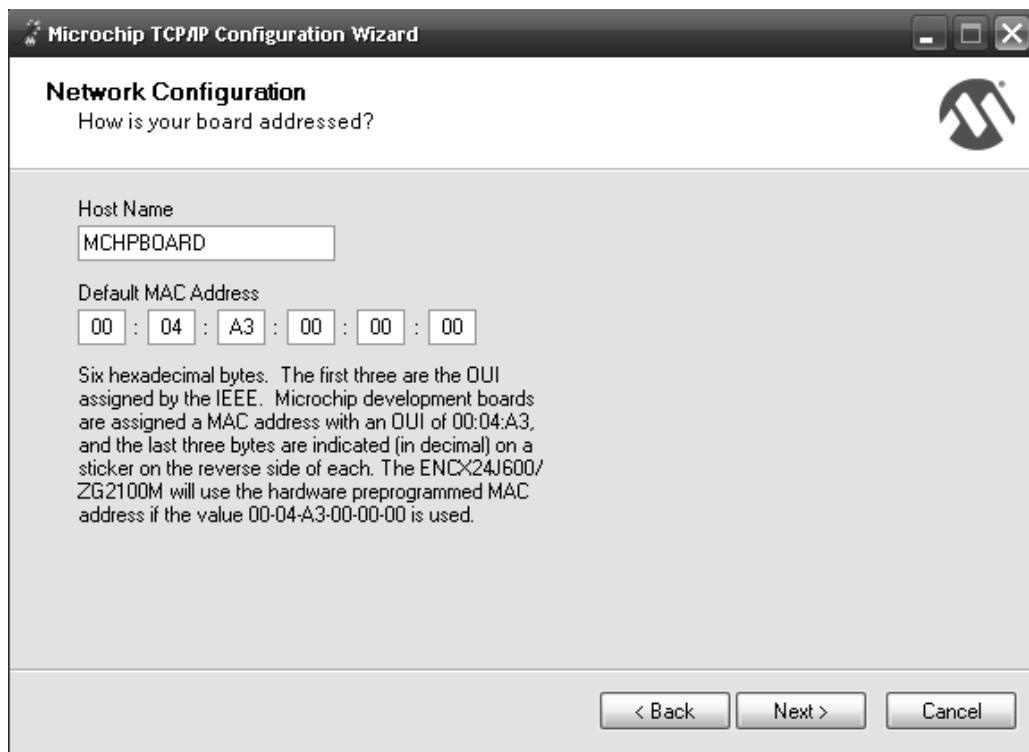


Figura 3.7: Configuración del nombre de *host* y la dirección MAC

pueda establecerse de manera simultánea.

En el caso de un servidor web, se pueden establecer diferentes ubicaciones para almacenar la página que verá el usuario final. La misma puede estar alojada en la memoria de programa del microcontrolador o en una memoria externa del tipo EEPROM.

Si bien existen otras opciones como memoria *Flash* y sistema FAT, las mencionadas anteriormente son las más representativas. Debido a que determinan si la página está alojada en la memoria del microcontrolador o en una memoria externa. La ventaja de una memoria externa, es que se puede disponer de toda la memoria de programa del microcontrolador para escribir el código y alojar el *Stack* TCP/IP. La principal desventaja de almacenar la página web en una memoria externa es el costo adicional. En la Figura 3.9 se pueden observar las opciones de almacenamiento para la página web.

### 3.1.3. El formato MPFS

Para poder almacenar los archivos de la página web en memoria, se utiliza un sistema de archivos sencillo implementado por Microchip. Se denomina “*Microchip Pic File System (MPFS)*”. Se trata del sistema de archivos que utiliza el microcontrolador para leer la página web. Las macros más importantes relacionadas con la configuración de este sistema, se encuentran en el archivo TCPIPConfig.h y son las siguientes:

```
#define MPFS_USE_EEPROM
#define USE_EEPROM_25LC1024
```

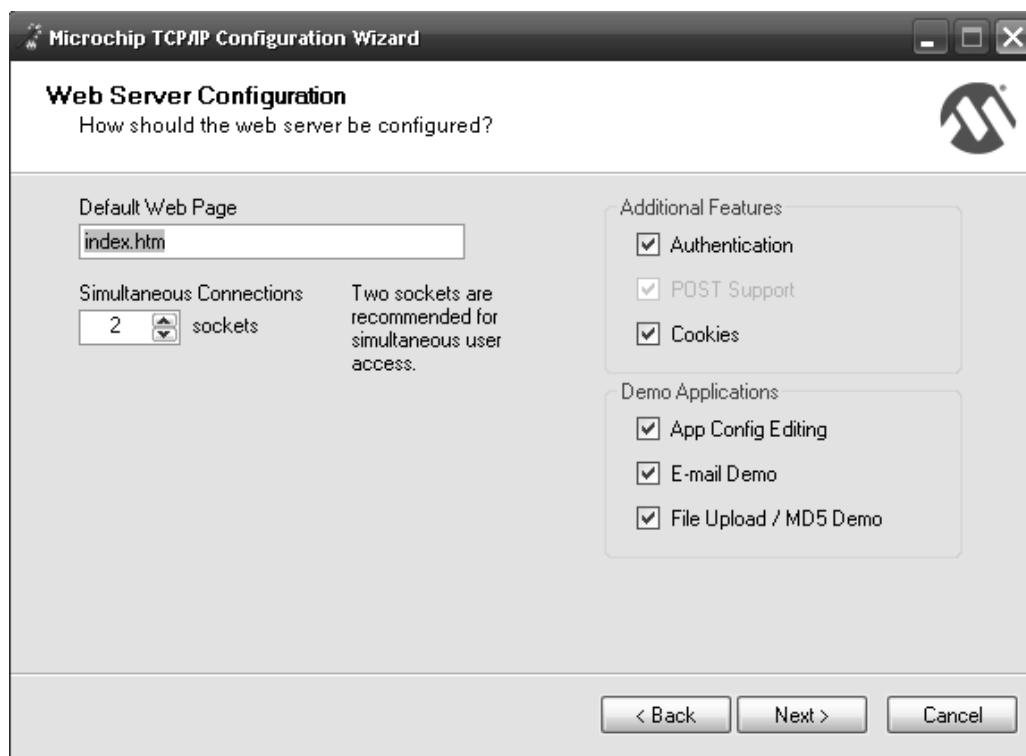


Figura 3.8: Configuración del nombre de la página principal y el número de *sockets*

```
#define MPFS_RESERVE_BLOCK
```

La primera de ellas indica que el conjunto de archivos asociados al servidor web se almacena precisamente en la EEPROM externa. En el caso de no definir esta macro, se debe generar una imagen MPFS en un archivo compilable de C. Este archivo se utiliza para guardar la página web en la memoria de programa del microcontrolador. La finalidad de la segunda macro es la definición del modelo de EEPROM, necesario para el correcto direccionamiento y operación del dispositivo.

Mediante la última macro, es posible reservar un bloque contiguo de datos al inicio del espacio de direcciones de la EEPROM. El bloque no se destina al almacenamiento de la imagen MPFS. Se usa en aquellas aplicaciones basadas en un microcontrolador que carece de EEPROM interna, sirve para almacenar parámetros de configuración, y esta activado por defecto en la pila TCP/IP.

Microchip proporciona una herramienta gratuita para la generación de la imagen MPFS. Así como también para su almacenamiento en memoria no volátil a través del servidor HTTP. Otras opciones serían la utilización de un servidor FTP o la escritura directa en la EEPROM mediante un dispositivo programador específico.

Una vez que el sitio web se encuentra terminado con los archivos html, jpg o gif, se debe convertirlo al sistema MPFS. De esta forma se lo puede cargar en el microcontrolador o en la memoria externa. La pila TCP/IP incluye un software especial para realizar esta conversión el: "MPFS.exe". Este software agrupa todos los archivos que componen la página web y genera una imagen MPFS.

Si se abre la imagen MPFS resultante con el block de notas, se puede ver que el contenido de cada archivo que compone la página web está ordenado secuencialmente. La aplicación MPFS.exe convierte

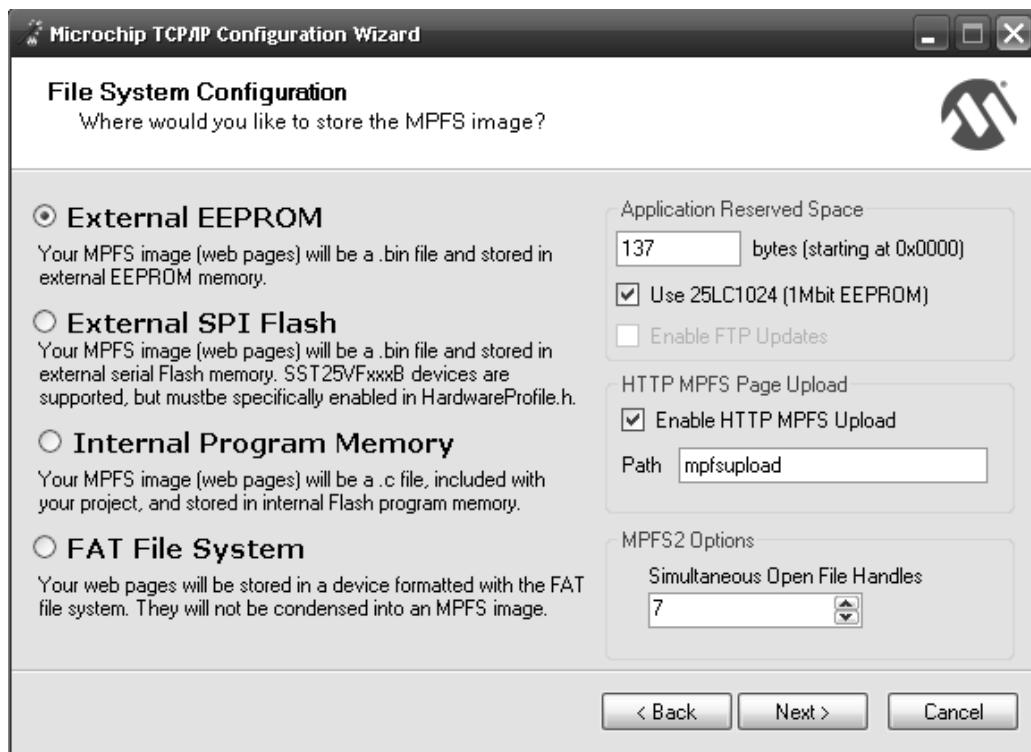


Figura 3.9: Opciones de almacenamiento para la página web

el conjunto de archivos que componen la página web en dos formatos posibles, “.bin” (para cargar la página web en la memoria EEPROM externa) y “.c” (para cargarla en la memoria de programa del microcontrolador).

Hay dos alternativas para cargar la web en la memoria externa. La primera de estas alternativas es a través del “MPFS.exe”. Este brinda la opción de transferir la web automáticamente a la memoria externa luego de compactarla. Para esto la placa debe estar conectada a la red y tener cargada la última versión de la pila TCP/IP. Se debe tener en cuenta que cada variable agregada en la web requiere su correspondiente variable en el *Stack*. Por lo tanto, es necesario recompilar el software y cargarlo en el microcontrolador, antes de compactar la web y transferirla a la memoria.

La segunda opción es cargar la web remotamente a través del explorador web. Es un sistema de transferencia de archivos por medio de HTTP. Las aplicaciones más seguras no permiten cargar la web de esta forma. Debido a que cualquiera podría ingresar a la dirección mencionada, conociendo solo la IP y modificarla. Se puede proteger el sistema de carga con contraseña, o bien cargar la web en la memoria de programa del PIC.

Es importante tener en cuenta que al agregar o quitar variables de la página web, se modifica el archivo “HTTPPrint.h”, por lo que hay que volver a compilar el *Stack* antes de cargar la aplicación en el microcontrolador. En aplicaciones que deben ser extremadamente económicas o bien donde la seguridad es una prioridad, se puede optar por almacenar la web en la memoria de programa del microcontrolador. Para esto se ejecuta el MPFS.exe y se selecciona la opción “C18/C32 Image” desde el menú principal. Esto generará un archivo “.c” que se debe agregar al proyecto de MPLAB para que sea compilado con el resto del código. Por defecto, el nombre del archivo es “MPFSImg2.c”. Se debe

comentar la sentencia “define MPFS\_USE\_EEPROM” dentro del archivo “TCPIPConfig.h”. De este modo se le indica a la pila TCP/IP que la web no estará en la memoria EEPROM externa. Luego se debe compilar el proyecto completo y cargarlo en el PIC. De esta forma se evita el uso de la memoria EEPROM externa, con el consecuente ahorro de dinero y espacio físico en la placa.

### 3.1.4. Desarrollo de una aplicación web

En esta sección se explicará en detalle como construir aplicaciones web utilizando el servidor web del *Stack*. Si se utilizaran solamente las opciones explicadas hasta aquí, el servidor estaría preparado para aceptar solamente peticiones de páginas web estáticas. En la mayoría de las aplicaciones el objetivo buscado será algo más práctico que la presentación de un documento estático. Para mejorar la interactividad con el usuario, el servidor ofrece dos funcionalidades importantes: la sustitución dinámica de variables, y la lectura de valores enviados desde el cliente mediante formularios HTML. Estas técnicas se analizarán a continuación. Primero se comentarán de forma general los archivos y herramientas implicados en la aplicación.

En la Figura 3.10 se muestra un esquema simplificado del proceso de construcción de una aplicación web. Debido a las limitaciones propias de un sistema embebido como este, existirá una fuerte relación entre el *firmware* del microcontrolador y el propio contenido web. Por ejemplo, el servidor reconocerá secuencias especiales de caracteres en las páginas web y realizará llamadas a determinadas funciones de usuario asociadas a las mismas (lo que se conoce como sustitución de variables dinámicas).

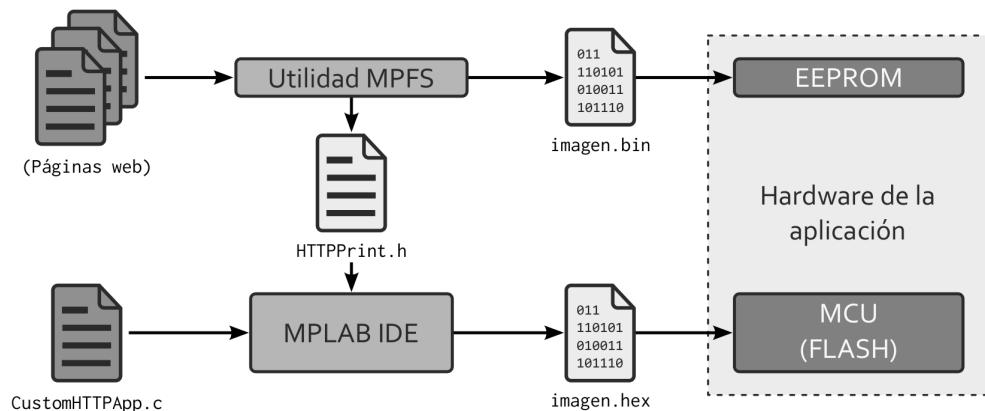


Figura 3.10: Proceso de construcción de una aplicación web

De forma simplificada, el proceso de desarrollo de una aplicación web requiere los siguientes pasos:

- Creación del conjunto de páginas web en un directorio único. Todos los archivos incluidos en él serán procesados por la utilidad MPFS y posteriormente almacenados en la EEPROM o en el microcontrolador, por lo que es importante reducir en lo posible el tamaño de los mismos.
- Generación de la imagen MPFS. En este paso, la herramienta MPFS aplicará por defecto compresión GZip a los archivos de texto, salvo a aquellos con contenido dinámico, o a los expresamente indicados por el usuario. Por cada variable dinámica encontrada en cualquiera de los archivos, la herramienta producirá la declaración de una función que el programador debe implementar para manejarla.

El conjunto de declaraciones se guarda automáticamente en el archivo de cabecera “HTTPPrint.h”, el cual no debe ser modificado. La otra salida generada por la herramienta es la imagen MPFS del sistema de archivos. Esta debe almacenarse en el sistema por cualquiera de los métodos mencionados anteriormente.

La definición de todas las funciones declaradas en “HTTPPrint.h”, así como de otras necesarias para el procesamiento de formularios, se realiza por defecto en el archivo “CustomHTTPApp.c”. Ambos archivos deben incluirse en el proyecto de la aplicación junto con el resto de archivos de la pila y los definidos por el usuario. La modificación posterior de las páginas web requiere la repetición de todo el proceso, salvo que sólo afecte al contenido estático.

### 3.1.4.1. Sustitución de variables dinámicas

A continuación se ilustrará con ejemplos el empleo de variables dinámicas para la generación flexible de contenido web. Lo que sigue es igualmente válido para cualquier archivo de texto, ya sea una página web, una hoja de estilos o un código JavaScript, entre otros. Esta característica ofrece un amplio rango de posibilidades al diseñador de la aplicación.

#### Variables dinámicas sin argumentos

Una variable dinámica es simplemente una cadena de caracteres delimitada por dos caracteres tilde (~). Cuando el servidor encuentra una de estas variables mientras procesa la respuesta a una petición HTTP la cual invoca a una función de usuario asociada antes de continuar con el llenado del buffer de transmisión TCP. El nombre de esta función comienza siempre con la cadena HTTPPrint\_ , seguida del nombre de la propia variable (recuérdese que las declaraciones de estas funciones las genera automáticamente la herramienta de construcción de imágenes MPFS). Considérese un ejemplo en el que se pretende consultar el estado del pin RA0 del microcontrolador mediante el servidor HTTP. Para ello se puede definir una variable denominada ~pin~ e incluirla en el siguiente código HTML:

```
<h1>Actualmente el pin está en el estado ~pin~</h1>
```

El programador debe también implementar la siguiente función en “CustomHTTPApp.c”:

```
void HTTPPrint_pin(void)
{
    if(PORTAbits.RA0)
        TCPPutROMString(sktHTTP, (ROM BYTE*)"alto");// RA0=1, sustituye ~pin~ por "alto"
    else
        TCPPutROMString(sktHTTP, (ROM BYTE*)"bajo");// RA0=0, sustituye ~pin~ por "bajo"
}
```

Como puede observarse, el programador toma el control del *buffer* TCP mientras se ejecuta la función HTTPPrint\_pin() (función *callback*). La función TCPPutROMString() pertenece a una familia de funciones definidas en la API del módulo TCP para el llenado del *buffer* de transmisión. Debe tenerse en cuenta que, debido a la implementación interna del servidor, cuando se invoca a la función *callback* sólo se garantiza la existencia de 16 bytes libres en este *buffer*. Por lo tanto, si fuera necesario escribir más de 16 bytes, el programador debe asegurarse de la disponibilidad de espacio suficiente (un ejemplo de esto es una llamada a TCPIsPutReady()).

Si no existiese suficiente espacio libre, la función *callback* debe ceder el control a la pila para la transmisión del segmento TCP, y ser invocada posteriormente para continuar la escritura. Veamos un ejemplo de este caso. Si se supone que la variable global *cadena* contiene un conjunto de caracteres recibidos a través de la UART, y que éstos se quieren mostrar en una página web mediante la variable dinámica *~uart~*, la función *callback* asociada podría escribirse de la siguiente forma:

```
void HTTPPrint_uart(void)
{
if(strlen((char*)cadena) > TCPIsPutReady(sktHTTP))
{
// No hay espacio suficiente en el buffer. Intentarlo en la siguiente llamada.
curHTTP.callbackPos = 0x01; // callbackPos != 0 para invocar de nuevo a la función.
return;
}
// Sí hay espacio. Escribir y limpiar callbackPos para dar por finalizada a la función.
TCPPutString(sktHTTP, cadena);
curHTTP.callbackPos = 0x00;
}
```

La estructura “*curHTTP*” mantiene información sobre la conexión HTTP que está siendo procesada, y se almacena en la RAM del microcontrolador. Nótese que en este ejemplo se supone que en algún momento existirá espacio suficiente para toda la cadena. Si ésta es demasiado grande, se deben realizar escrituras parciales en el *buffer* de transmisión.

### 3.1.4.2. Variables dinámicas con argumentos

En determinadas ocasiones el sistema explicado en el apartado anterior puede ser ineficiente en términos de consumo de memoria ROM. Para solucionarlo, el servidor hace posible la reutilización de código mediante el paso de parámetros a las funciones *callback*. Se pueden definir variables dinámicas con uno o dos parámetros enteros de 16 bits.

Supongamos que se quiere extender la funcionalidad del primer ejemplo, de forma que se pueda leer el estado de todos los pines del puerto A del microcontrolador. El código HTML quedaría en este caso de la siguiente manera:

```
<h1>Actualmente el pin RA0 está a nivel ~pin(0)~</h1>
<h1>Actualmente el pin RA1 está a nivel ~pin(1)~</h1>
<!-- ... etc ... -->
```

El código de la función asociada podría definirse así:

```
void HTTPPrint_pin(WORD i)
{
BYTE temp;
temp = PORTA;
// Lectura del puerto A
while(i!=0)
{
```

```

temp >>= 1;// Desplazar "i" veces los bits a la derecha
i--;
}
temp &= 1;// temp es 1 sólo si el bit "i" de PORTA también lo es
if(temp)
    TCPPutROMString(sktHTTP, (ROM BYTE*)"alto");// temp=1, sustituye ~pin(i)~ por "alto"
else
    TCPPutROMString(sktHTTP, (ROM BYTE*)"bajo");// temp=0, sustituye ~pin(i)~ por "bajo"
}

```

De esta forma es posible definir una única función para producir el mismo resultado que ocho funciones independientes, con el consiguiente ahorro de memoria y la mayor facilidad en la depuración de errores.

### 3.1.4.3. Procesamiento de formularios

La sustitución de variables dinámicas es una técnica adecuada para mostrar información acerca del estado interno del microcontrolador. Si se desea además que el usuario de la aplicación web pueda tener cierto control sobre el funcionamiento del dispositivo, será necesario utilizar alguna de las técnicas que se exponen a continuación. El servidor permite el procesamiento de los datos enviados a través de un formulario HTML en cualquiera de los métodos GET y POST de HTTP.

#### Método GET

Cuando un formulario HTML se envía mediante el método GET de HTTP, el conjunto de datos del mismo (*form data set*) se anexa a la URI del recurso en el servidor. Este conjunto de datos está formado por pares nombre=valor asociados a los elementos de entrada concretos de un formulario. Los datos se separan de la URI del recurso mediante un signo de interrogación (?), mientras que cada par se separa del siguiente con un carácter (&).

El método GET es el que se maneja de forma más sencilla por el servidor HTTP. Esto se debe a que toda la información proporcionada por el usuario se encuentra inmediatamente disponible en memoria cuando se invoca a la función encargada de su análisis. Sin embargo, por razones técnicas de la implementación del servidor, la longitud máxima del conjunto de datos del formulario es de unos 100 bytes. Cuando se necesita procesar más información es necesario emplear el método POST.

Cuando el servidor recibe una petición de recurso con un conjunto de datos anexionados a su URL, éste se decodifica automáticamente y se almacena en el *array <curHTTP.data>*. Posteriormente se invoca a la función *HTTPExecuteGet()*, definida también en el archivo *CustomHTTPApp.c* y que el programador debe adaptar a su aplicación.

Siguiendo el ejemplo anterior, supongamos que se pretende cambiar de nivel algún pin del puerto A mediante un formulario web. Los pasos a seguir podrían ser:

- El primer paso consiste en la creación de un formulario HTML con un campo de control apropiado. Si se desea que el usuario pueda elegir mediante un menú desplegable el pin sobre el que quiere actuar, el código asociado podría ser el siguiente:

```

<form action='index.html' method='get'>
<select name='pin'>

```

```
<option>0</option>
<option>1</option>
<!-- siguientes ... -->
</select>
<input type="submit" value="Modificar estado">
</form>
```

- Cuando se envía el formulario<sup>3</sup> el servidor invoca a la función `HTTPExecuteGet()`, como se indicó anteriormente. Esta función es invocada siempre que se recibe una petición sobre un recurso con argumentos. Por lo tanto se debe comprobar el nombre del recurso para poder diferenciarlos. La función `MPFSGetFilename()` devolverá el nombre del archivo asociado a la petición HTTP en la dirección que recibe como segundo argumento:

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
BYTE *ptr;
BYTE filename[20];
MPFSGetFilename(curHTTP.file, filename, 20);
if(!memcmppgm2ram(filename, (ROM void*)"index.html", 10))
{
// El recurso solicitado es "index.html", pueden procesarse los argumentos
```

- Los argumentos almacenados en `<curHTTP.data>` pueden obtenerse mediante las funciones `HTTPGetArg()` y `HTTPGetROMArg()`, que reciben el nombre asociado a cada uno de ellos. En el caso del ejemplo, existe un único nombre llamado `<pin>` que puede recibir 8 valores numéricos diferentes:

```
ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"pin");
switch(*ptr)
{
case '0':
PORTAbits.RA0 ^= 1;
break;
case '1':
PORTAbits.RA1 ^= 1;
break;
// Continuar con el resto de pines...
}
}
}
// Fin "if"
return HTTP_IO_DONE;
}
}
// Fin de HTTPExecuteGet()
```

De esta forma se puede variar el valor del puerto del microcontrolador utilizando la página web cargada en el servidor.

---

<sup>3</sup>O cuando se invoca al recurso mediante una URL con argumentos, como por ejemplo: /index.html?pin=1

## 3.2. Hardware necesario para la implementación del módulo

En la siguiente sección se detallará el hardware con el que fue implementado el módulo Ethernet, en la Figura 3.11 se puede observar un diagrama en bloques del mismo. La pieza principal del módulo es el microcontrolador, en el se encuentra alojado el *Stack TCP/IP* y la página web. Este se encarga de decodificar las órdenes provenientes de Internet y enviarlas al módulo X-10 por medio de su módulo UART.

Para que el microcontrolador pueda ser conectado a Internet, se necesita un controlador de Ethernet. El mismo está especialmente diseñado para actuar como un puente entre una red Ethernet y un microcontrolador. La conexión entre estos dos elementos se lleva a cabo utilizando el protocolo SPI.

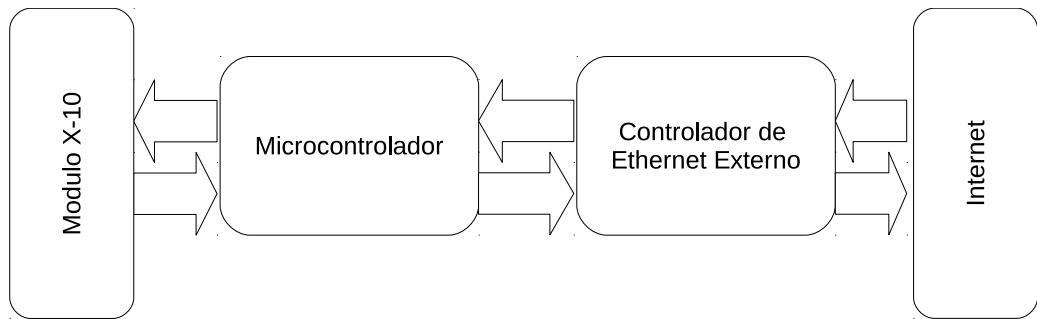


Figura 3.11: Diagrama en bloques del módulo Ethernet

### 3.2.1. Microcontrolador

Después de analizar diferentes opciones, se decidió realizar el diseño de este módulo con el mismo microcontrolador con el que se implementó el módulo X-10 un PIC24HJ128GP502 (ver Sección 2.3). Este circuito integrado puede manejar sin problemas la pila TCP/IP, por su elevada velocidad (80MHz). Los 128Kb de memoria de programa hacen que se pueda almacenar todo el software necesario y la página web sin ningún inconveniente.

Por otro lado, el uso del PIC24HJ128GP502 presenta varias ventajas. En primer lugar, se pueden utilizar el mismo compilador y secciones de código desarrolladas anteriormente, como por ejemplo la configuración del oscilador y de la UART, que son iguales a las utilizadas en el módulo X-10. El hardware con las conexiones básicas para el funcionamiento del microcontrolador es el mismo que se utilizó en el módulo X-10 (ver Sección 2.3).

#### 3.2.1.1. Mapeo de pines

En la Figura 3.12 se puede ver el mapeo de pines del microcontrolador. En este caso no se utiliza la totalidad de los pines del mismo, debido a que este módulo posee menos hardware que el módulo X-10. Solo se utilizan dos módulos del integrado, el SPI y la UART. Esto hace que la cantidad de pines que se deben mapear sea mucho menor que en el caso anterior. Esta configuración se realiza dentro del perfil de hardware, y mediante la función InitializeBoard que se encuentra en el programa MainDemo.c (como se verá en secciones posteriores).

El pin RBO se configura como salida (se la denomina I/O) y se utiliza para manejar la entrada de *Chip Select* (CS) del controlador Ethernet. Las salidas *Data Output* (SDO), y *Clock Output* (SCK)

del módulo SPI fueron mapeadas en los pines RB1, y RB2 respectivamente. Mientras que la entrada *Data Input* (SDI) de dicho módulo fue mapeada en el pin RB3.

Luego se habilitó una de las entradas de interrupción externa del microcontrolador (INTX). Esta entrada fue mapeada en el pin RB10, y se utiliza para conectar a la salida *Interrupt Output* (INT) del controlador de Ethernet.

La configuración de los pines de la UART es la misma que se utilizó en el módulo X-10. La entrada Rx se mapeó en la pata RP14 y Tx se mapeó en la pata RP13. También se utilizó el control de flujo, para lo cual son necesarios dos pines mas, *Clear to Send* (U1CTS), que se mapeó en RP12, y *Request to Send* (U1RTS) mapeada en RP11. El pin RB15 se utilizó para conectar un led, que al destellar indica que el microcontrolador ha ingresado en el *loop* principal del *Stack*.

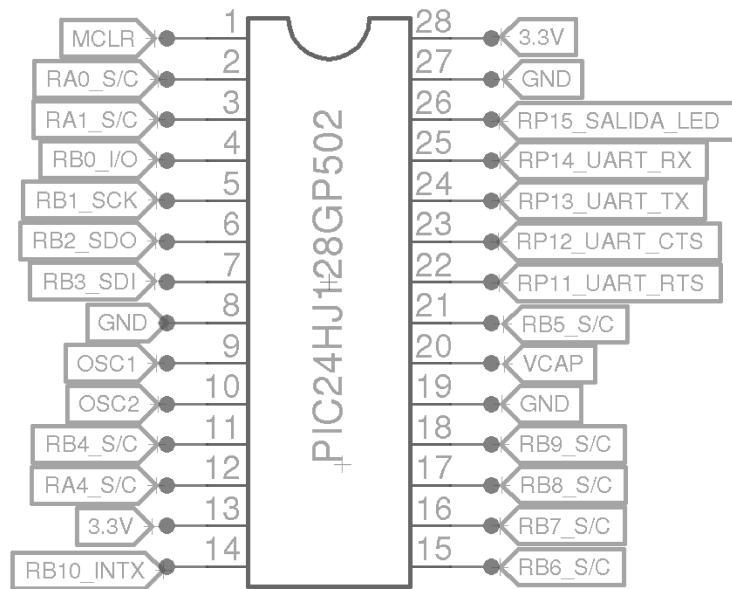


Figura 3.12: Mapeo de pines del microcontrolador

### 3.2.2. Controlador Ethernet externo

La pila TCP/IP proporciona soporte directo a toda la familia de controladores de Ethernet de la familia de Microchip. Por lo que se tomó la decisión del controlador a utilizar teniendo en cuenta solo los modelos de este fabricante.

En la familia de Microchip hay varios modelos de microcontroladores que poseen un controlador Ethernet integrado. Por otro lado, existen dos modelos de controladores Ethernet externos que se conectan al microcontrolador por medio del protocolo SPI [Inc08][Inc10a]. La primer opción fue descartada cuando se decidió utilizar el mismo microcontrolador con el que se implementó el módulo X-10. Por lo tanto, a la hora de elegir el circuito integrado a utilizar, solo se tuvieron en cuenta los dos modelos de controladores externos.

Finalmente, se eligió al ENC28J60, este es un controlador Ethernet 10BASE-T diseñado para aplicaciones integradas. Debido a su reducido tamaño (sólo 28 pines) y a su interfaz de comunicación serie SPI, ha alcanzado una enorme popularidad y actualmente es usado en numerosas aplicaciones

comerciales. Su disponibilidad en formato DIP facilita la realización de prototipos y es especialmente deseable en una aplicación como la que nos ocupa.

Como se mencionó anteriormente, la comunicación con el microprocesador se realiza mediante un bus SPI estándar de sólo cuatro líneas. Esto supone la posibilidad de emplear procesadores con un menor número de entradas y salidas, y por consiguiente, más baratos. En la Figura 3.13 se muestra el diagrama de bloques simplificado de la estructura del dispositivo, en él se pueden ver los principales bloques que lo componen, y como están interconectados entre sí. Como resumen global de características:

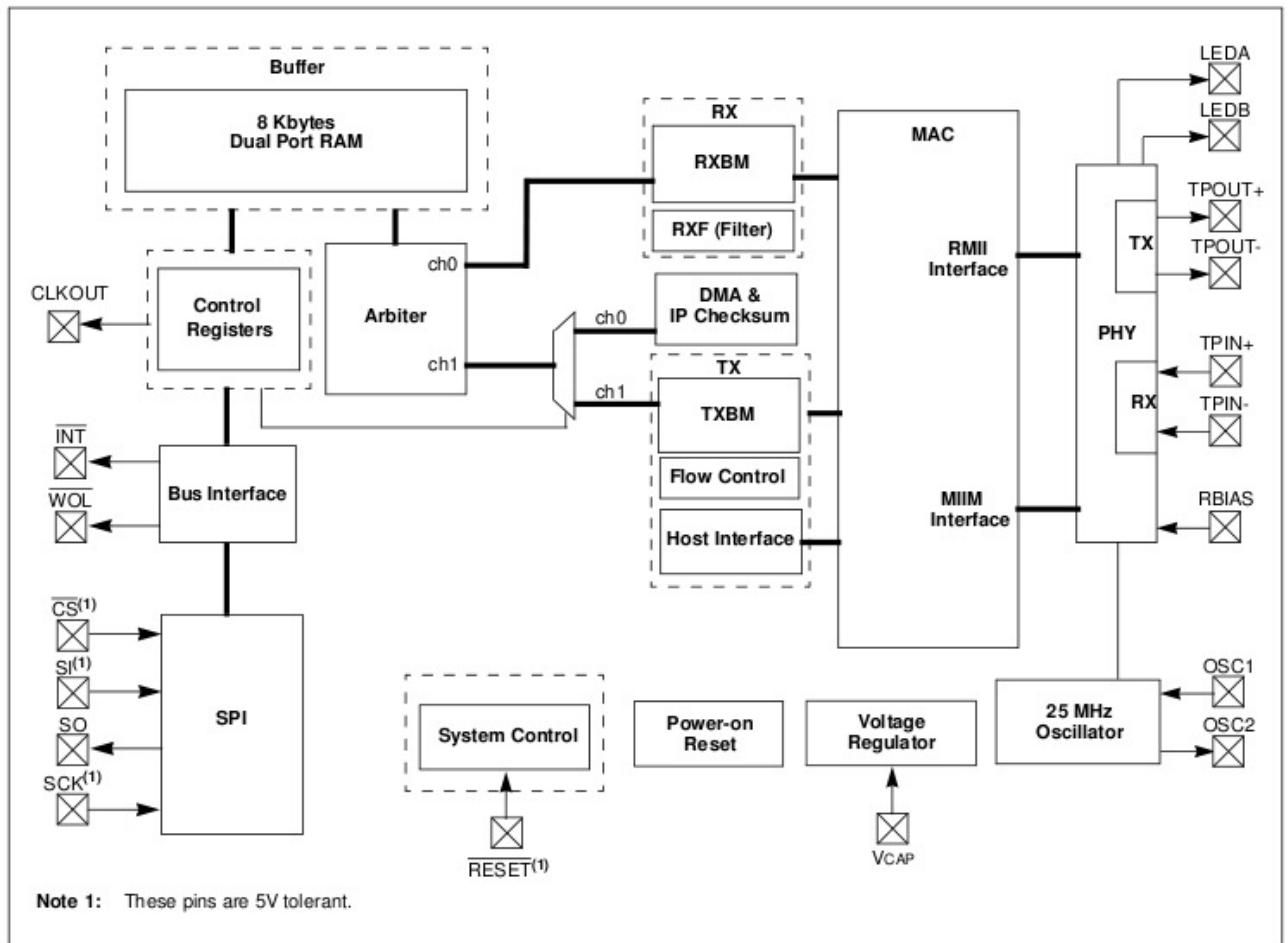


Figura 3.13: Diagrama en bloques del ENC28J60

cas, se pueden destacar los siguientes puntos:

- El dispositivo integra tanto la capa física (PHY) Ethernet a 10 Mpbs como la correspondiente capa MAC de acceso al medio, según el estándar IEEE 802.3. La capa PHY comprende toda la circuitería analógica necesaria para la codificación y decodificación de datos en el canal, mientras que la capa MAC se encarga del control de la transmisión y la recepción de tramas, la detección de colisiones, el cálculo de CRCs, etc.

- El ENC28J60 contiene una memoria RAM de doble puerto de 8 KBytes, que el microcontrolador puede asignar como *buffer* de transmisión o de recepción. Esta característica proporciona un método eficiente de almacenamiento y modificación de paquetes, liberando al procesador de las tareas asociadas a la administración del *buffer* y eliminando necesidades adicionales de memoria RAM.

- Una de las características más relevantes del ENC28J60 es su interfaz de comunicación con el microcontrolador, basada en un bus SPI que requiere sólo cuatro líneas: dos para la transferencia bidireccional de los datos, una para sincronización y otra línea para la selección del dispositivo. Esto permite que incluso los procesadores con un número reducido de líneas de entrada y salida puedan ser empleados en este tipo de aplicaciones.

Se puede destacar la existencia de una API gratuita de acceso al controlador, proporcionada por el fabricante y compatible con varios compiladores, que facilita considerablemente su operación y permite al usuario abstraerse de detalles específicos sobre la estructura interna del dispositivo.

El dispositivo es capaz de funcionar en los modos *fullduplex* y *halfduplex*. Sin embargo, es importante mencionar que la capa física no soporta la autonegociación, lo que en la práctica supone que se suela configurar en modo *halfduplex* para maximizar su compatibilidad. Otras cualidades interesantes de la MAC del ENC28J60 son el filtrado configurable de paquetes y la posibilidad de interrumpir la ejecución de código en el microprocesador como respuesta a hasta seis tipos diferentes de eventos, incluyendo *WakeOnLAN2*.

En cuanto a sus características eléctricas, el ENC28J60 es un dispositivo que requiere 3,3V de alimentación; sin embargo, sus entradas soportan niveles lógicos de 5V, facilitando su integración en sistemas que trabajan con esta tensión de alimentación. En la Figura 3.14 se puede observar un esquema con la distribución de pines del ENC28J60 con encapsulado SPDIP.

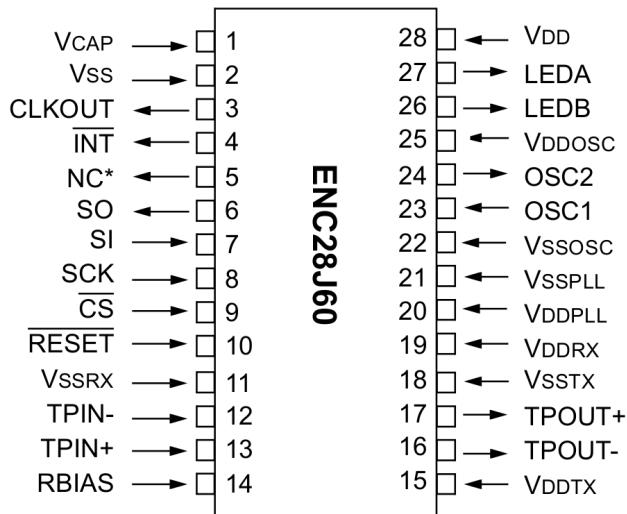


Figura 3.14: Esquema con la distribución de pines del ENC28J60

Para funcionar y poder conectarse a la red Ethernet, el ENC28J60 necesita de algunos componentes externos, tal como puede verse en la Figura 3.15. En primer lugar, se pueden ver las conexiones de las líneas SPI que comunican al microcontrolador con el controlador de Ethernet. En la figura se observa

un circuito para adaptar los niveles lógicos de las señales que ingresan al microcontrolador. En este caso ese circuito no es necesario debido a que el PIC24HJ128GP502 funciona con una tensión de alimentación de 3,3V.

Los pines TPIN+ y TPIN- deben conectarse a un transformador 1:1 especial para redes 10BASE-T. Los pines TPOUT+ y TPOUT- necesitan de un transformador de pulso con relación 1:1 y punto medio. Este transformador debe ser capaz de proveer una aislación de 2000V. Además, todos los pines mencionados necesitan de un resistor de  $50\Omega$  con un 1% de tolerancia. Sin embargo, en este caso se utilizaron resistores de  $47\Omega$  con una tolerancia del 5%. Debido a que no fue posible conseguir los componentes sugeridos por el fabricante. Esto no causó ningún tipo de problemas.

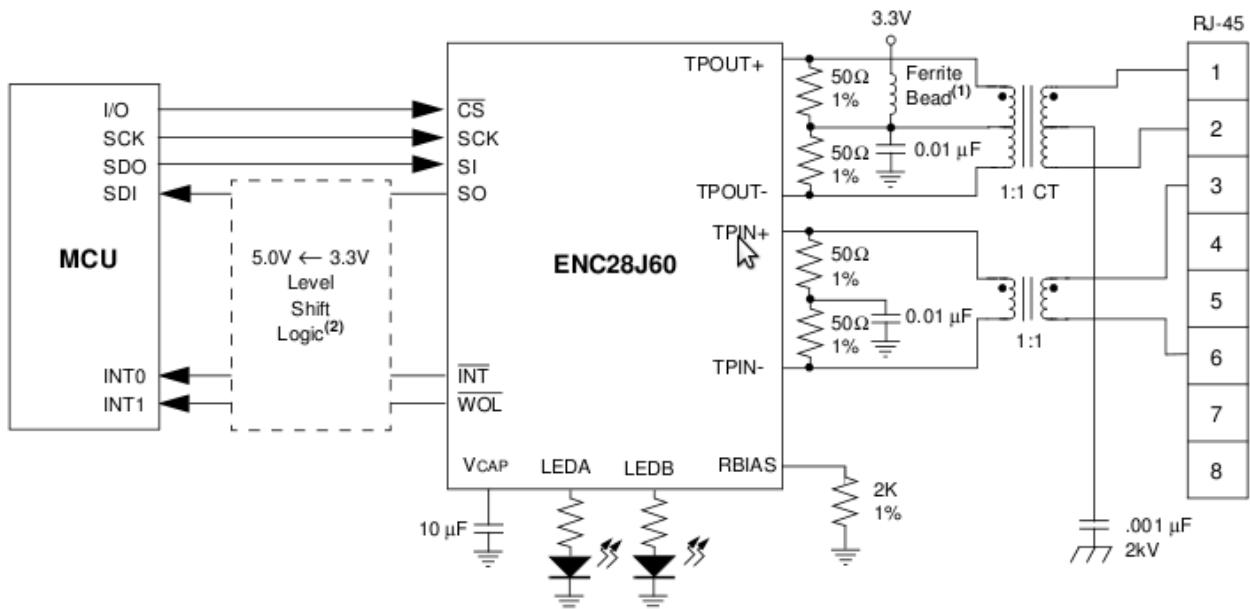


Figura 3.15: Conexiones externas del ENC28J60

La circuitería analógica interna del ENC28J60 también requiere que se conecte un resistor de  $2K\Omega$  con un 1% de tolerancia entre el pin RBIAS y GND. En este caso se utilizó un resistor de  $2.2K\Omega$  con una tolerancia del 5%.

El ENC28J60 es capaz de operar también con una tensión de alimentación de solo 2,5V, en cuyo caso habrá que disponer un condensador de  $10 \mu F$  entre el pin VCAP y GND. Todos los pines de alimentación (Vdd) deben ser conectados a la misma fuente de 3,3V, y todos los pines GND deben conectarse al mismo nodo. Cada par de pines Vdd/Vss debe tener su propio condensador cerámico de  $0,1\mu F$  tan cerca de los pines como sea posible. Hay que tener en cuenta que corrientes relativamente altas van a circular entre el integrado y el par trenzado, por lo que las pistas correspondientes deberían ser lo mas cortas y anchas posibles para minimizar su resistencia eléctrica. Los pines LEDA y LEDB permiten la conexión de sendos leds destinados a brindar información sobre el estado del chip.

El Circuito integrado está diseñado para funcionar con un oscilador de 25MHz, y una de las formas de proporcionárselo es utilizando un cristal de cuarzo como se ve en la Figura 3.16. Se conecta el cristal entre los pines OSC1 y OSC2, y se ponen también ambos extremos a GND mediante C1 y C2. El fabricante no sugiere ningún valor de capacidad en la hoja de datos, por lo que se utilizaron capacitores

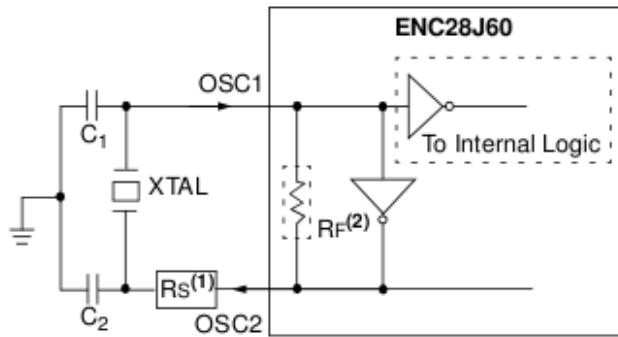


Figura 3.16: Oscilador a cristal

de 15pF. El resistor RF es necesario para brindarle estabilidad al reloj y en este caso se colocó uno de 1MΩ. El resistor Rs se utiliza solo con algunos tipos de cristales (*AT strip cut crystals*), y en este caso no fue necesario. También se puede configurar al oscilador para que trabaje con una señal de reloj externa aplicada al pin OSC. El pin OSC2 se conecta a GND a través de un resistor para minimizar el ruido.

### 3.3. Modificación del software para la implementación del proyecto

Como se mencionó en secciones anteriores el *Stack TCP/IP* le brinda al usuario una serie de ejemplos con aplicaciones estándar. Se utilizó uno de estos ejemplos, llamado “*TCP/IP Web Vend App*” como base para el desarrollo del software del proyecto. En esta sección se mostrarán las modificaciones que se efectuaron en el software de dicho ejemplo para la realización del módulo Ethernet.

En primer lugar, se hablará de las modificaciones que se deben realizar para lograr que el hardware de la placa funcione correctamente. Luego se describirán las configuraciones del *Stack* que se realizaron en el archivo *TCP/IPConfig.h*, y las funciones del archivo *CustomHTTPApp.c*. Por último, se describirán los programas que fueron agregados a la pila *TCP/IP* para lograr ensamblar y enviar los mensajes al módulo X-10.

#### 3.3.1. Definición del perfil de hardware

La pila *TCP/IP* define un conjunto de etiquetas simbólicas para hacer referencia a algunos terminales del microcontrolador. Estos terminales son los encargados de realizar funciones especiales, tales como la comunicación con el controlador Ethernet, o con la memoria EEPROM externa.

También es importante especificar la frecuencia de trabajo del sistema. Este dato es fundamental, debido a que es utilizado para la correcta inicialización de los temporizadores que marcan la ejecución de eventos temporales.

Estas configuraciones se realizan mediante la definición de macros de C. Un conjunto de estas macros válidas para una pieza concreta de hardware se conoce como perfil. En el archivo “*Hardware-Profile.h*” existen perfiles predefinidos para varias placas de desarrollo comercializadas por Microchip. En este proyecto se modificó el perfil de hardware del kit de desarrollo Explorer 16 [Inc05] (que utiliza el ENC28J60), para adaptarlo a la placa utilizada en el módulo Ethernet.

El perfil de hardware se encuentra en la carpeta “*Alternative Configurations*”, dentro de la carpeta que contiene el ejemplo “*TCP/IP WebVend App*”.

La elección de un perfil determinado, se realiza definiendo una macro en las opciones del proyecto de MPLAB. En la Figura 3.17 se puede ver como se efectúa dicha definición. La configuración anterior hace que la macro que se muestra a continuación<sup>4</sup> incluya automáticamente el archivo del perfil de hardware en el proyecto.

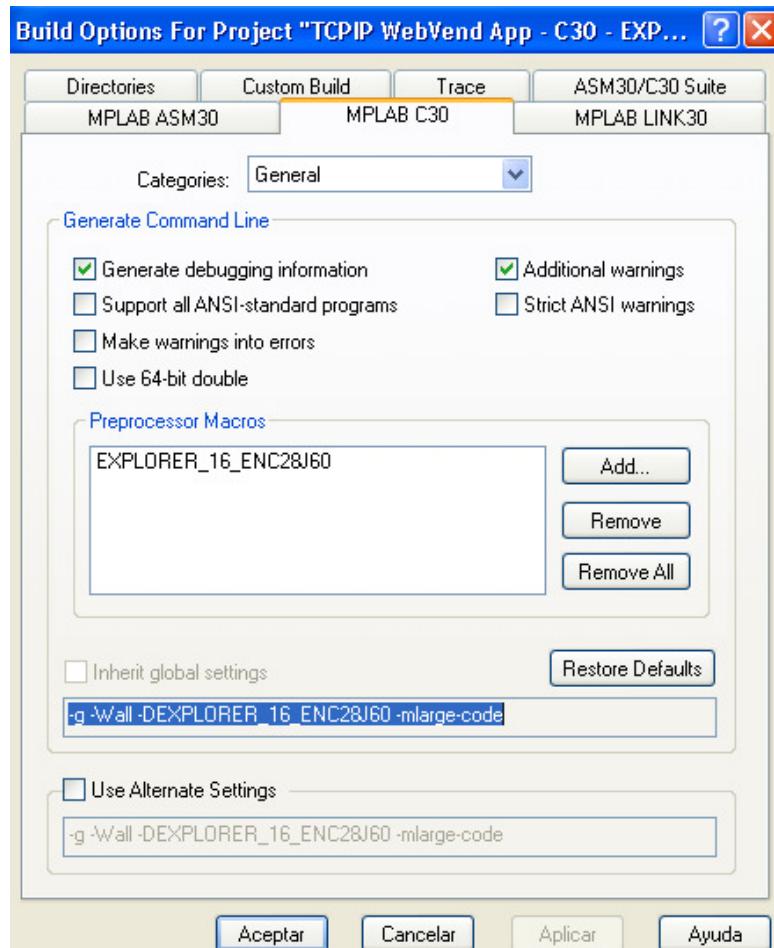


Figura 3.17: Definición de la macro del perfil de hardware

```
#elif defined(EXPLORER_16_ENC28J60)
#if defined(__C30__)
#include "Alternative Configurations/HardwareProfile EXPLORER_16_ENC28J60 C30.h"
```

En el código anterior se puede ver que se incluirá el archivo “HardwareProfile EXPLORER\_16\_ENC624J60 C30.h” en el proyecto si se cumplen dos condiciones. La primera es que se haya definido el perfil EXPLORER\_16\_ENC28J60, y la segunda es que se utilice el lenguaje C30.

<sup>4</sup>Esta macro se encuentra en el archivo “HardwareProfile.h”

Una vez que el archivo está incluido en el proyecto se lo debe modificar para definir el perfil de hardware. A continuación se muestran algunas de esas modificaciones:

```
#elif defined(__dsPIC33F__) || defined(__PIC24H__)

    _FOSCSEL(IESO_OFF & FNOSC_PRI);
    _FOSC(FCKSM_CSECMD & OSCIOFNC_OFF & POSCMD_XT);
    _FWDT(FWDTEN_OFF);

#endif
```

En la sección de código anterior se puede ver la configuración del oscilador del microcontrolador. El compilador ingresará en esta parte del programa solo si se utiliza un PIC de las familias dsPIC33 ó 24H. Aquí se seleccionó el oscilador primario a cristal, se activó el cambio de clock para poder utilizar el PLL, y se desactivó el *watchdog timer*. En la línea de código que se muestra a continuación, se le informa al *Stack* cual será la frecuencia máxima del reloj, en este caso 80MHz.

```
#define MAXIMUM_PIC_FREQ (80000000ul)
```

En las líneas de código que se ven a continuación, se definen las etiquetas de la configuración del SPI. En las dos primeras líneas, se define la salida del microcontrolador que se encargará de manejar la entrada *Chip Select* del controlador Ethernet, en este caso RB0.

```
#define ENC_CS_TRIS (TRISEBbits.TRISBO)
#define ENC_CS_IO (LATBbits.LATB0)
// SPI SCK, SDI, SDO pins are automatically controlled by the
// PIC24/dsPIC SPI module
#define ENC_SPI_IF (IFS0bits.SPI1IF)
#define ENC_SSPBUF (SPI1BUF)
#define ENC_SPISTAT (SPI1STAT)
#define ENC_SPISTATbits (SPI1STATbits)
#define ENC_SPICON1 (SPI1CON1)
#define ENC_SPICON1bits (SPI1CON1bits)
#define ENC_SPICON2 (SPI1CON2)
```

### 3.3.2. La función InitializeBoard

La inicialización del hardware se realiza mediante una llamada al procedimiento InitializeBoard(). El mismo se encuentra en el programa principal, y es donde se especifican las entradas y salidas del procesador y se configuran otros parámetros como el PLL y la UART. En la sección del código que se muestra a continuación, se puede ver la configuración de los registros del PLL del microcontrolador, y el procedimiento de cambio de fuente de reloj. Los valores de los registros y los pasos efectuados en el procedimiento son idénticos a los explicados en la Sección [2.4](#).

```
PLLFBDR = 0x20; // Multiply by 32 for 160MHz VCO output (10MHz XT oscillator)
CLKDIV = 0x0000; // FRC: divide by 2, PLLPOST: divide by 2, PLLPRE: divide by 2
__builtin_write_OSCCONH(0b011);
__builtin_write_OSCCONL(0x01);
while(OSCCONbits.OSWEN == 1) {};
```

En la siguiente sección de código se muestra el mapeo de pines del microcontrolador. En primer lugar, se desbloquea el registro OSCON (ver Sección 2.4), y luego se configuran las entradas y las salidas de la UART y el módulo SPI. Por último, se vuelve a bloquear el registro OSCON. El mapeo de pines resultante se puede observar en la Figura 3.12.

```
--builtin_write_OSCCONL(OSCCON & ~(1<<6));// Unlock PPS

// Inputs

RPINR20bits.SDI1R = 3; // SDI1 = RP3
RPINR0bits.INT1R =5; // Assign RP5 to INT1
RPINR18bits.U1RXR=14; // Assign UART1 Receive (U1RX) to RP14
RPINR18bits.U1CTSR=12; // Assign UART1 Clear to Send (U1CTS) to RP12

// Outputs

RPOR0bits.RP1R = 8; // RP1 = SCK1
RPOR1bits.RP2R = 7; // RP2= SD01
RPOR6bits.RP13R=0b00011; // Assign UART1 transmit (U1TX) to RP13
RPOR5bits.RP11R=0b00100; // Assign UART1 Request to Send (U1RTS) to RP11

--builtin_write_OSCCONL(OSCCON | (1<<6)); // Lock PPS
```

Antes de finalizar el procedimiento, se configura la UART como se muestra a continuación. Esta configuración es exactamente igual a la que se realizó en el módulo X-10 (ver Sección 2.4).

```
U1BRG=259;//baud rate de 9600
U1MODEbits.PDSEL=0b01;//8-bit data, even parity
U1MODEbits.RTSMD=0;//pin in Flow Control mode
U1MODEbits.UEN=0b00;//UxTX and UxRX pins are enabled and used
U1MODEbits.STSEL=0; //One Stop bit
U1MODEbits.UARTEN=1;//UART1 is enabled
U1STAbits.UTXEN=1;//Transmit enabled, UxTX pin controlled by UART
```

De esta forma, se finaliza toda la configuración del hardware del módulo.

### 3.3.3. Configuración del servidor HTTP

En la Sección 3.1.2 se explicó como realizar la configuración de la pila TCP/IP, mediante la modificación de determinadas macros en el archivo TCPIPConfig.h. Algunas de estas macros son específicas del servidor HTTP. En este caso, se utilizó el “TCP/IP Configuration Wizard” para configurar el servidor. Las opciones más importantes se comentarán a continuación. En primer lugar, se muestran las principales macros de la capa de aplicación:

#### Application Options

```
#define STACK_USE_ICMP_SERVER
```

```
#define STACK_USE_HTTP2_SERVER
#define STACK_USE_DHCP_CLIENT
#define STACK_USE_GENERIC_TCP_SERVER_EXAMPLE
#define STACK_USE_DNS
```

En la primera línea, se activó el servidor ICMP para poder procesar las solicitudes de *ping* recibidas. En la siguiente línea, se puede ver que se configuró al módulo como servidor HTTP. La pila TCP/IP incluye un servidor HTTP disponible en dos versiones. En este caso, se utilizó la segunda versión del servidor llamada HTTP2. Esta ofrece un conjunto más amplio de funciones, y una API mejorada.

Por último, se activó el cliente DHCP para poder obtener una dirección IP dinámica del *router* ADSL, y también se activó el servidor DNS. En la siguiente sección del archivo TCPIPConfig.h, se muestran las opciones para el almacenamiento de la página web:

#### Data Storage Options

```
#define STACK_USE_MPFS2
//#define MPFS_USE_EEPROM
//#define MPFS_USE_SPI_FLASH
```

Como se puede ver, se utilizó el formato MPFS2 para el almacenamiento de la página web. Además, se dejaron comentadas las dos últimas macros, que son las que se utilizan si la página es almacenada en una memoria EEPROM externa, o una memoria *flash*. Esto le indica al *Stack* que la página será almacenada en la memoria interna del microcontrolador.

A continuación, se muestra la sección del código que contiene la configuración del nombre de *host*, la dirección MAC, la IP, la máscara de red y la puerta de enlace o *gateway*:

#### //Network Addressing Options

```
#define MY_DEFAULT_HOST_NAME "WEBVEND"

#define MY_DEFAULT_MAC_BYTE1          (0x00)
#define MY_DEFAULT_MAC_BYTE2          (0x04)
#define MY_DEFAULT_MAC_BYTE3          (0xA3)
#define MY_DEFAULT_MAC_BYTE4          (0x00)
#define MY_DEFAULT_MAC_BYTE5          (0x00)
#define MY_DEFAULT_MAC_BYTE6          (0x00)

#define MY_DEFAULT_IP_ADDR_BYTE1      (10ul)
#define MY_DEFAULT_IP_ADDR_BYTE2      (0ul)
#define MY_DEFAULT_IP_ADDR_BYTE3      (0ul)
#define MY_DEFAULT_IP_ADDR_BYTE4      (220ul)

#define MY_DEFAULT_MASK_BYTE1         (255ul)
#define MY_DEFAULT_MASK_BYTE2         (255ul)
#define MY_DEFAULT_MASK_BYTE3         (255ul)
#define MY_DEFAULT_MASK_BYTE4         (0ul)
```

```
#define MY_DEFAULT_GATE_BYTE1          (169ul)
#define MY_DEFAULT_GATE_BYTE2          (254ul)
#define MY_DEFAULT_GATE_BYTE3          (1ul)
#define MY_DEFAULT_GATE_BYTE4          (1ul)
```

En el siguiente código, se pueden observar las configuraciones de la capa de transporte. En primer lugar, se configuró al *Stack* para que utilice el protocolo TCP, y se comentó la macro que habilita el uso de UDP. Luego, se especificó la manera en la que se asigna la RAM disponible en el ENC28J60 y el PIC para los *buffers* asociados de transmisión y recepción. Por último, se definieron los nombres de los tipos de *socket* TCP.

```
//Transport Layer Options

#define STACK_USE_TCP
//#define STACK_USE_UDP

#define TCP_ETH_RAM_SIZE (4096ul)
#define TCP_PIC_RAM_SIZE (0ul)
#define TCP_SPI_RAM_SIZE (0ul)
#define TCP_SPI_RAM_BASE_ADDRESS (0x00)

// Define names of socket types

#define TCP_SOCKET_TYPES
#define TCP_PURPOSE_GENERIC_TCP_CLIENT 0
#define TCP_PURPOSE_GENERIC_TCP_SERVER 1
#define TCP_PURPOSE_TELNET 2
#define TCP_PURPOSE_FTP_COMMAND 3
#define TCP_PURPOSE_FTP_DATA 4
#define TCP_PURPOSE_TCP_PERFORMANCE_TX 5
#define TCP_PURPOSE_TCP_PERFORMANCE_RX 6
#define TCP_PURPOSE_UART_2_TCP_BRIDGE 7
#define TCP_PURPOSE_HTTP_SERVER 8
#define TCP_PURPOSE_DEFAULT 9
#define TCP_PURPOSE_BERKELEY_SERVER 10
#define TCP_PURPOSE_BERKELEY_CLIENT 11
#define END_OF_TCP_SOCKET_TYPES
```

Por último, se muestra la configuración de la capa de aplicación. Con la primera macro se define el número máximo de conexiones HTTP simultáneas. Debe asegurarse la existencia de, al menos, un *socket* TCP por conexión HTTP. Luego, se define el nombre del archivo que el servidor devolverá por defecto, en este caso es “index.htm”.

Para finalizar, se habilita el uso del método POST, y se definen las longitudes máximas de los argumentos del GET y POST. También se configura el número mínimo de bytes en el *buffer* de

transmisión, que es necesario para ejecutar funciones de *callback*. En la última línea, se define el puerto que utilizará el servidor, en este caso se utilizó el 80.

```
//Application-Specific Options

//HTTP2 Server options

#define MAX_HTTP_CONNECTIONS (2u)
#define HTTP_DEFAULT_FILE "index.htm"
#define HTTPS_DEFAULT_FILE "index.htm"
#define HTTP_DEFAULT_LEN (10u)
#define HTTP_USE_POST

// Define the maximum data length for reading cookie and GET/POST arguments (bytes)
#define HTTP_MAX_DATA_LEN (100u)

// Define the minimum number of bytes free in the TX FIFO before executing callbacks
#define HTTP_MIN_CALLBACK_FREE (16u)

#define HTTP_PORT      (80u)
```

### 3.3.4. La página web y el archivo CustomHTTPApp.c

En la Figura 3.18 se puede ver una imagen de la página web que se encuentra cargada en el microcontrolador del módulo Ethernet. Esta cuenta con diferentes botones, con los que se pueden activar o desactivar los actuadores de la red X-10. En el caso de la figura, cada uno de estos actuadores comanda una luz de una habitación de la vivienda.

Esta página web, también cuenta con una serie de variables dinámicas, que informan acerca del estado de los diferentes actuadores. Cuando una luz de la vivienda se encuentra activada, se puede ver la palabra “encendida” a la derecha de los botones que la comandan. En la figura se puede ver que dos de las salidas se encuentran activas.

A continuación, se muestra un fragmento del archivo HTML de la página web. Aquí se encuentran definidos los dos botones que comandan una luz, y la variable dinámica que indica su estado.

```
<td background="tabsBitmap.jpg"><div align="center" class="Estilo15">
<a href="index.htm?luz1=on">Encender</a></div></td>
<td>&nbsp;</td>
<td background="tabsBitmap.jpg"><div align="center" class="Estilo15">
<a href="index.htm?luz1=off">Apagar</a></div></td>
<td>&nbsp;</td>
<td background="tabsBitmap.jpg"><div align="center">
<span class="Estilo16">~estluz1~</span></div></td>
```

Los botones de la página, utilizan el método GET para enviar las órdenes al microcontrolador (ver Subsección 3.1.4.3). Por ejemplo, si se presiona el botón para encender la luz 1 en la página web, el navegador envía al servidor una URL con un argumento. En este caso es el siguiente:



Figura 3.18: Imagen de la página web

```
index.htm?luz1=on
```

El par nombre=valor es luz1=on, esto invoca a la función HTTPExecuteGet(), definida en el archivo CustomHTTPApp.c. Dentro de esta función se ejecuta el siguiente código, con el cual se determina el valor del argumento de <luz1>:

```
ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"luz1");
if(strcmppgm2ram((char*)ptr, (ROM char*)"on") == 0)
{
    armarpalabra(0,HouseA,Unit1,On);
    LED1_IO =1;
}
if(strcmppgm2ram((char*)ptr, (ROM char*)"off") == 0)
{
    armarpalabra(0,HouseA,Unit1,Off);
    LED1_IO =0;
}
```

Si el valor de <luz1> es “on” se llama a la función “armarpalabra()”, que se encuentra dentro del archivo X-10.c. Cuando se invoca a la función “armarpalabra()” se envían los cuatro argumentos que se enumeran a continuación:

- Un número de fila que está asociado al número de unidad del módulo al que va dirigido el mensaje. Se utiliza para modificar un elemento de una matriz del archivo X-10.c.
- La dirección de casa y el número de unidad del módulo al que se le envía la orden. Las definiciones con los códigos de casa, unidades, y órdenes del protocolo X-10 se encuentran en el archivo X-10.h.
- La orden. En este caso la orden es encender(*on*). También se coloca un “uno lógico” en la variable auxiliar <LED1\_IO> que se utilizará luego.

Si el argumento enviado mediante la URL es “off”, el procedimiento es el mismo. En este caso, la orden enviada como parámetro de la función es apagar (*off*). Y se coloca un “cero lógico” en la variable auxiliar <LED1\_IO>.

En la última línea del archivo HTML mostrado anteriormente, se puede ver la declaración de la variable dinámica que se utiliza para mostrar el estado de la luz 1, y se la llamó <estluz1>. Esta variable tiene asociada una función de *callback*. Esta función, se encuentra en el archivo CustomHTTPApp.c y se muestra a continuación:

```
void HTTPPrint_estluz1(void)
{
    if (LED1_IO)
        TCPPutROMString(sktHTTP,(ROM void*)"encendida");
    return;
}
```

Aquí es donde se utiliza la variable auxiliar <LED1\_IO>, si el estado de dicha variable es un “uno lógico” (esto sucede cuando la luz esta encendida), se reemplaza la variable dinámica <estluz1>por el valor “encendida”. Si en cambio el valor de <LED1\_IO>es un “cero lógico” (luz apagada) no se reemplaza la variable dinámica, y el espacio en donde antes aparecía la palabra “encendida” permanece vacío.

### 3.3.5. El archivo X-10.C

Con el fin de enviar las órdenes al módulo X-10, se agregó al *Stack TCP/IP* un archivo denominado X-10.c. Este contiene dos funciones, la primera se denomina “armarpalabra()”, y se encarga de codificar las órdenes para enviar. La segunda función se denomina “enviardatos()”, y como su nombre lo indica, se encarga de enviar los datos utilizando la UART del microcontrolador.

En la sección anterior, se mencionó que la función “HTTPExecuteGet()” invoca a la función “armarpalabra()” y se detallaron los argumentos que se envían a dicha función. Estos datos se cargan en una matriz que esta definida en este programa, y que se utiliza para enviar los datos por medio de la UART como se explicará a continuación.

En la Figura 3.19 se muestra un diagrama de la matriz. Esta cuenta con tres columnas, y una cantidad de filas iguales al número de actuadores que se manejan con la página web. Cada actuador manejado por la página cuenta con un número, este coincide con el número de fila que se le asigna en la matriz.

La primera columna se utiliza para guardar una bandera que indica que existen datos para enviar al actuador. La segunda columna de la matriz se utiliza para guardar la primer palabra de ocho bits que se le envía al módulo X-10. Esta contiene el código de casa del actuador y su número de unidad sin el sufijo (ver Sección 2.4). La tercera columna contiene la segunda palabra de ocho bits que se le

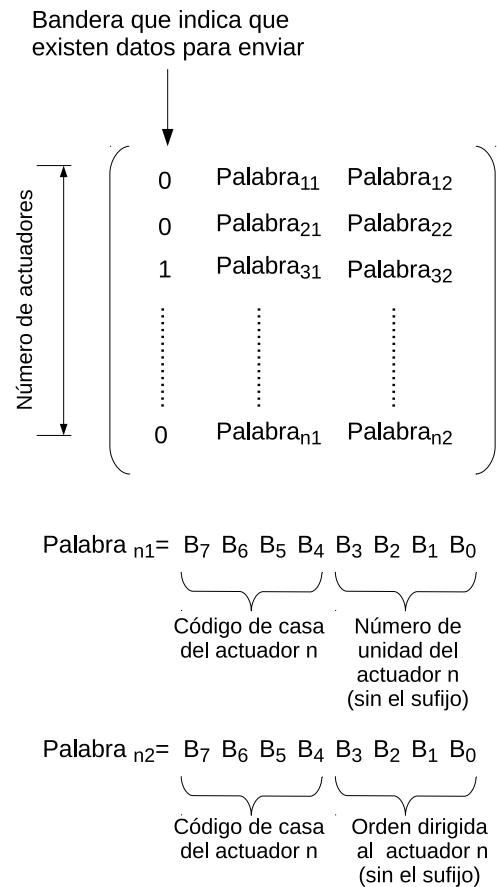


Figura 3.19: Matriz definida en el programa X-10.C

envía al módulo X-10. La misma está formada por el código de casa del actuador, y la orden dirigida al mismo sin el sufijo.

La función “armarpalabra()” se encarga de situarse en la fila correspondiente al módulo. Luego coloca un “uno lógico” en la primera columna de la matriz, y ensambla las palabras de ocho bits. Por último, coloca dichas palabras en la segunda y tercera columna.

En la Figura 3.20 se muestra un diagrama de flujo de la función “enviardatos()”. Esta es invocada en el bucle principal del *Stack*, que se encuentra en el archivo MainDemo.c. Cada vez que sucede esto, la función recorre la primer columna de la matriz, en busca de un “uno lógico”. Si lo encuentra envía la primera palabra de esa fila de la matriz.

Posteriormente, libera al microcontrolador para que pueda seguir realizando otras tareas. Como se mencionó anteriormente ninguna función de la pila TCP/IP puede acaparar el uso del microcontrolador por mucho tiempo. Cuando se ejecuta nuevamente la función, se envía la segunda palabra de esa fila. Por último, se coloca un cero en el elemento de la primer columna.

En la Figura 3.21 se puede ver un gráfico en el que se muestra el flujo de datos que existe entre La PC del usuario, el Modulo Ethernet, el modulo X-10 y el actuador bidireccional cunado se envía una orden desde la pagina web. En primer lugar el usuario envía una orden desde internet. Luego el módulo Ethernet la decodifica (como se vio en la Subsección 3.3.4), y utiliza la UART del microcontrolador

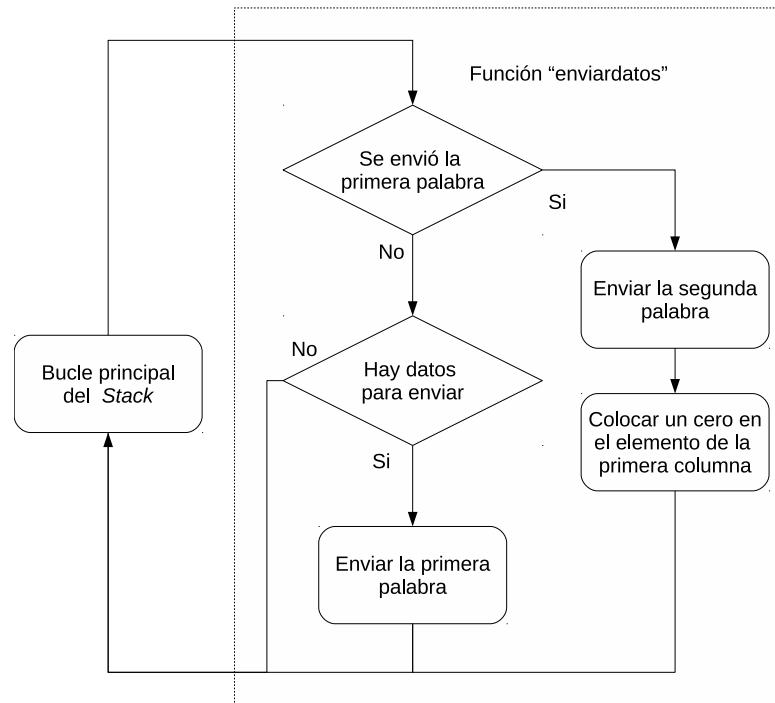


Figura 3.20: Diagrama de flujo de la función “enviardatos”

para enviar dos palabras de ocho bits al módulo X-10. Este decodifica las palabras recibidas, ensambla los bloques de datos X-10 (Subsecciones 2.4.2, 2.4.3) y los envía por la red eléctrica (Subsección 2.4.5). Cuando el actuador bidireccional de X-10 recibe correctamente la orden ensambla y envía la confirmación por la red eléctrica.

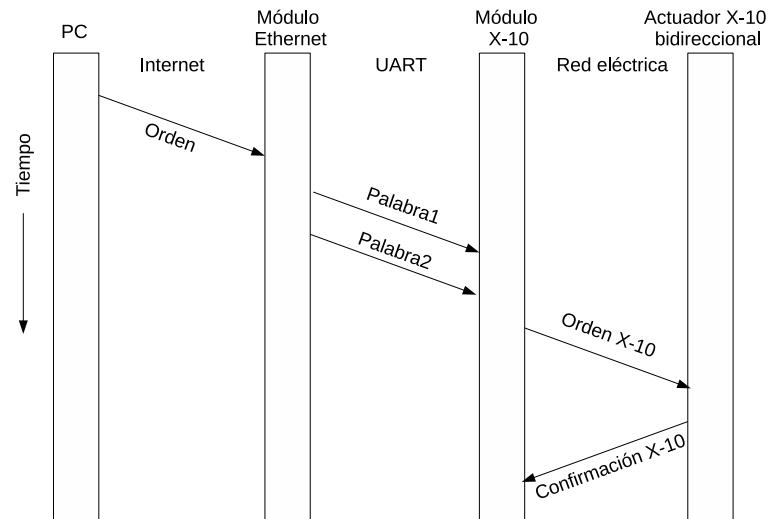


Figura 3.21: Flujo de datos entre los componentes del sistema

### 3.4. Conclusiones

En este capítulo se realizó una explicación detallada del módulo Ethernet. Se trata de un servidor web embebido en un microcontrolador, que se conecta a la red mediante un *router* (o un *modem*) ADSL y tiene alojada una página de Internet. Mediante dicha página web el usuario puede controlar los diferentes actuadores de la red domótica.

En la Sección 3.1 comenzó describiendo la estructura de la pila TCP/IP de Microchip, que es el software con el cual está programado el microcontrolador del módulo. Se trata de un software con estructura modular que implementa un conjunto importante de protocolos de Internet. Se describió su funcionamiento, y los principales programas que la componen. Luego se realizó una introducción teórica acerca de el procesamiento de formularios y el método GET.

En la Sección 3.2 se presentó el hardware del módulo, se justificó la elección del microcontrolador y del controlador de Ethernet. Después de analizar diferentes opciones, se decidió realizar el diseño de este módulo con el mismo microcontrolador con el que se implementó el módulo X-10 un PIC24HJ128GP502. Este circuito integrado puede manejar sin problemas la pila TCP/IP, por su elevada velocidad. Los 128Kb de memoria de programa hacen que se pueda almacenar todo el software necesario y la página web sin ningún inconveniente. Por otro lado, el uso del PIC24HJ128GP502 presenta varias ventajas. En primer lugar, se pueden utilizar el mismo compilador y secciones de código desarrolladas anteriormente, como por ejemplo la configuración del oscilador y de la UART, que son iguales a las utilizadas en el módulo X-10.

Se eligió al controlador de Ethernet ENC28J60 por su reducido tamaño (sólo 28 pines), su interfaz de comunicación serie SPI, y su disponibilidad en formato DIP, la cual facilita la realización de prototipos. Luego se describió en detalle el funcionamiento de dicho controlador (ENC28j60), y se mencionaron los componentes necesarios para realizar la conexión a internet de este circuito integrado.

En la Sección 3.3 se explicaron las modificaciones realizadas en los archivos de la pila TCP/IP de Microchip para adaptar el software de la misma al hardware de la placa del módulo Ethernet. En primer lugar, se realizó la definición de un conjunto de macros de C que se conoce como perfil de hardware. Luego se inicializó el hardware configurando un procedimiento llamado InitializeBoard(), que se encuentra en el programa principal de la pila, y es donde se especifican las entradas y salidas del procesador, y se configuran otros parámetros como el PLL y la UART. Por último, se explicó como realizar la configuración del servidor HTTP de la pila TCP/IP, mediante la modificación de determinadas macros en el archivo TCPIPConfig.h.

En la Subsección 3.3.4 se mostró la página web cargada en el microcontrolador, su código HTML y el archivo CustomHTTPApp. En este archivo se encuentran las funciones que sirven para relacionar a las variables de la página web con el software de la pila TCP/IP.

En la última Subsección se describió al archivo X-10.c que se agrega a la pila TCP/IP para relacionar los módulos Ethernet y X-10. Este contiene dos funciones, la primera se denomina “armarpalabra()”, y se encarga de codificar las órdenes para enviar. La segunda función se denomina “enviardatos()”, y como su nombre lo indica, se encarga de enviar las órdenes al módulo X-10 utilizando la UART del microcontrolador.

En la siguiente sección se plantearán las conclusiones del proyecto y se propondrán posibles mejoras.



## Capítulo 4

# Consideraciones finales

En este proyecto final se estudiaron las principales características de un sistema domótico y los principales protocolos existentes en el mercado. Con esta información se logró encontrar un protocolo con el que se puede automatizar una vivienda de forma sencilla y económica.

Se diseñó un dispositivo con el cual se pueden controlar los artefactos eléctricos de una vivienda mediante Internet. Este dispositivo puede ser utilizado en cualquier vivienda. Su instalación es muy sencilla, y puede ser realizada por personas sin conocimientos de electricidad ni de electrónica. Con este desarrollo una persona con movilidad reducida es capaz de mejorar su calidad de vida, ya que le permitirá realizar diversas tareas domésticas sin la colaboración de un tercero.

En la Figura 4.1 se puede ver el prototipo del módulo X-10, este está dividido en tres plaquetas. En la figura se pueden observar como están distribuidas los diferentes circuitos explicados en la Sección 3.2. Durante la etapa de diseño se priorizó la seguridad del usuario en todo momento, por este motivo se colocaron los circuitos que interactúan con la red eléctrica en una plaqueta separada a la que contiene el microcontrolador. Además, se rediseñaron circuitos existentes en otros dispositivos similares.

En la Figura 4.2 se puede observar el prototipo del módulo 802.3, y los dos circuitos integrados que forman parte de él, el microcontrolador y el controlador de Ethernet externo.

## Posibles mejoras

A continuación se muestra una lista de posibles mejoras y ampliaciones:

- Se podría agregar un sistema de autenticación a la página web para que ninguna persona no autorizada pueda comandar la red domótica. Esto mejoraría considerablemente la seguridad del sistema.
- Se podría modificar el software y agregar temporizadores, para que el sistema encienda o apague los actuadores en un determinado horario, que sería programado por el usuario desde la página web.
- Cuando el módulo X-10 se utiliza como actuador bidireccional, activa su salida de 220V utilizando un relé. Esto no le permite ejecutar los comandos *dim*, *brigth* y *pre-set dim*, que se utilizan para variar la intensidad de la lámpara que es conectada al actuador. Se podría reemplazar el relé por un tiristor para que el módulo sea capaz de ejecutar estos comandos.
- Se pueden rediseñar los prototipos de ambos módulos con componentes de montaje superficial y

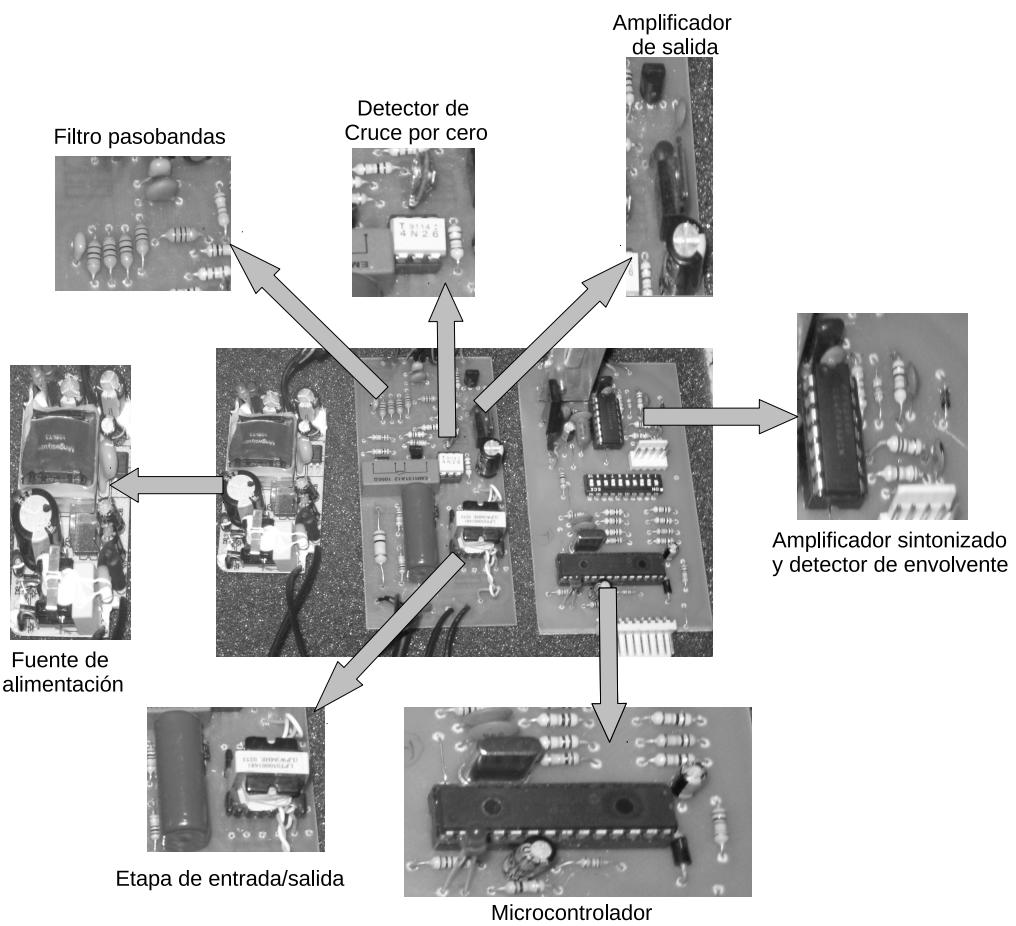


Figura 4.1: Prototipo del módulo X-10

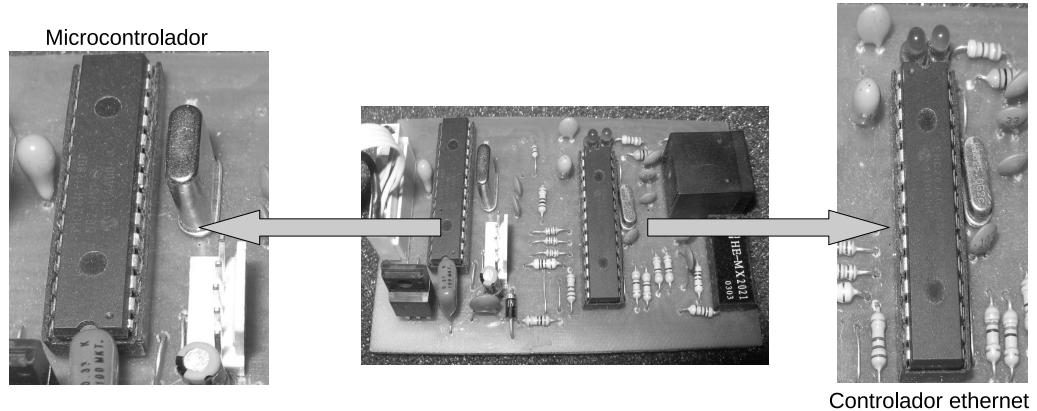


Figura 4.2: Prototipo del módulo Ethernet

---

utilizar circuitos impresos doble faz para lograr reducir su tamaño.

- Se podría reemplazar el controlador Ethernet del módulo 802.3 por un controlador WI-FI para que pueda funcionar de forma inalámbrica.



## Apéndice A

# Diagramas esquemáticos del proyecto

A continuación se muestran los diagramas esquemáticos de las tres plaquetas que componen el proyecto. En el momento de la realización se optó por dividir al hardware del módulo X-10 en dos. En una parte se colocaron todos los circuitos que están en contacto con la red eléctrica, se lo denominó “módulo de alta tensión”. El diagrama esquemático se puede ver en la Figura [A.1](#). El microcontrolador y los componentes mas sensibles del módulo X-10 se colocaron en otro circuito impreso. A esta parte del circuito se la denominó “módulo transceptor”. Su diagrama esquemático se puede ver en la Figura [A.2](#).

En la Figura [A.3](#) se puede observar el diagrama esquemático del módulo Ethernet.

Por último, en las Figuras [A.4](#), [A.5](#), y [A.6](#) se muestran las imágenes de los circuitos impresos del módulo de alta tensión, del módulo transceptor, y del Módulo Ethernet.

APÉNDICE A. DIAGRAMAS ESQUEMÁTICOS DEL PROYECTO

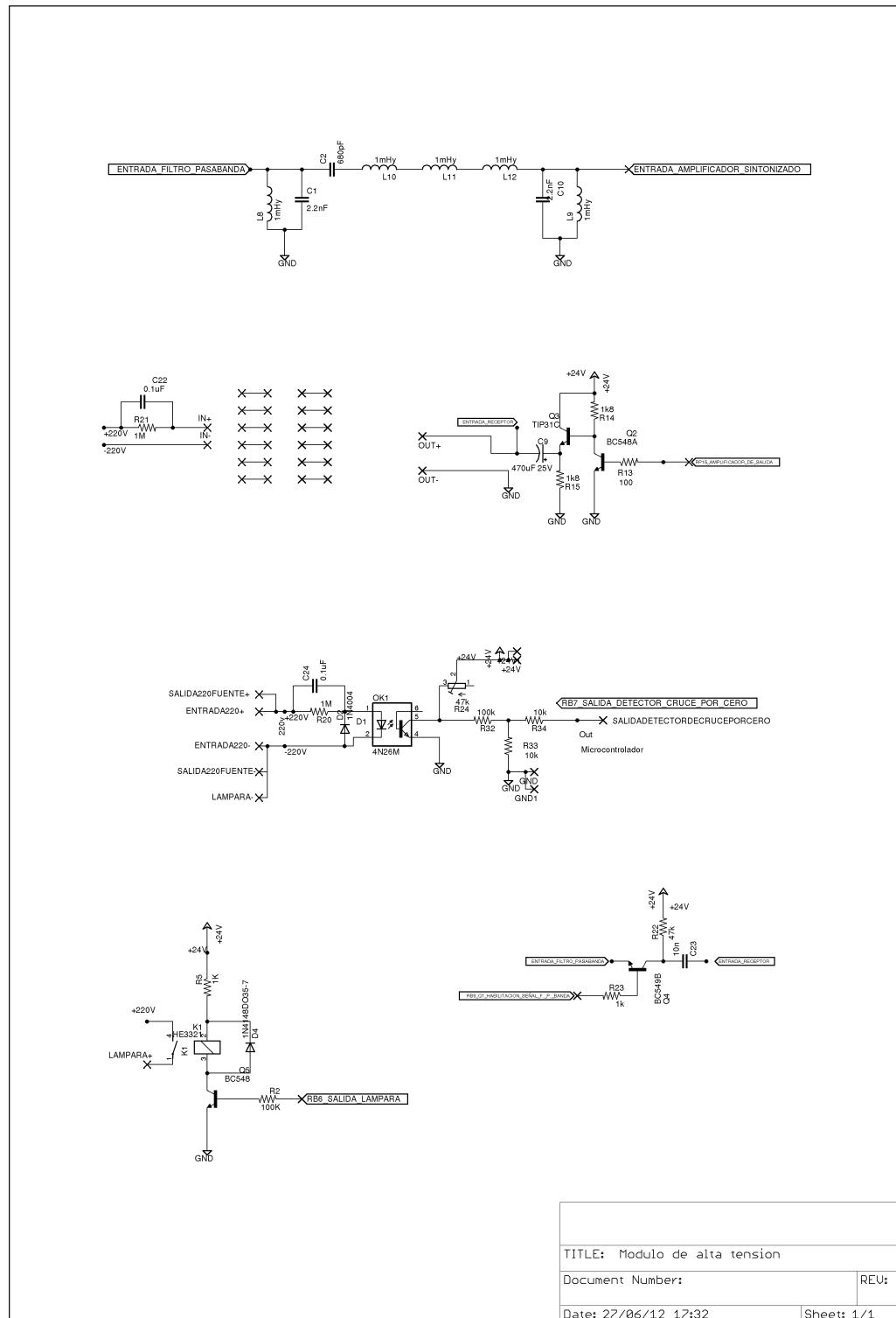


Figura A.1: Diagrama esquemático del módulo de Alta Tensión

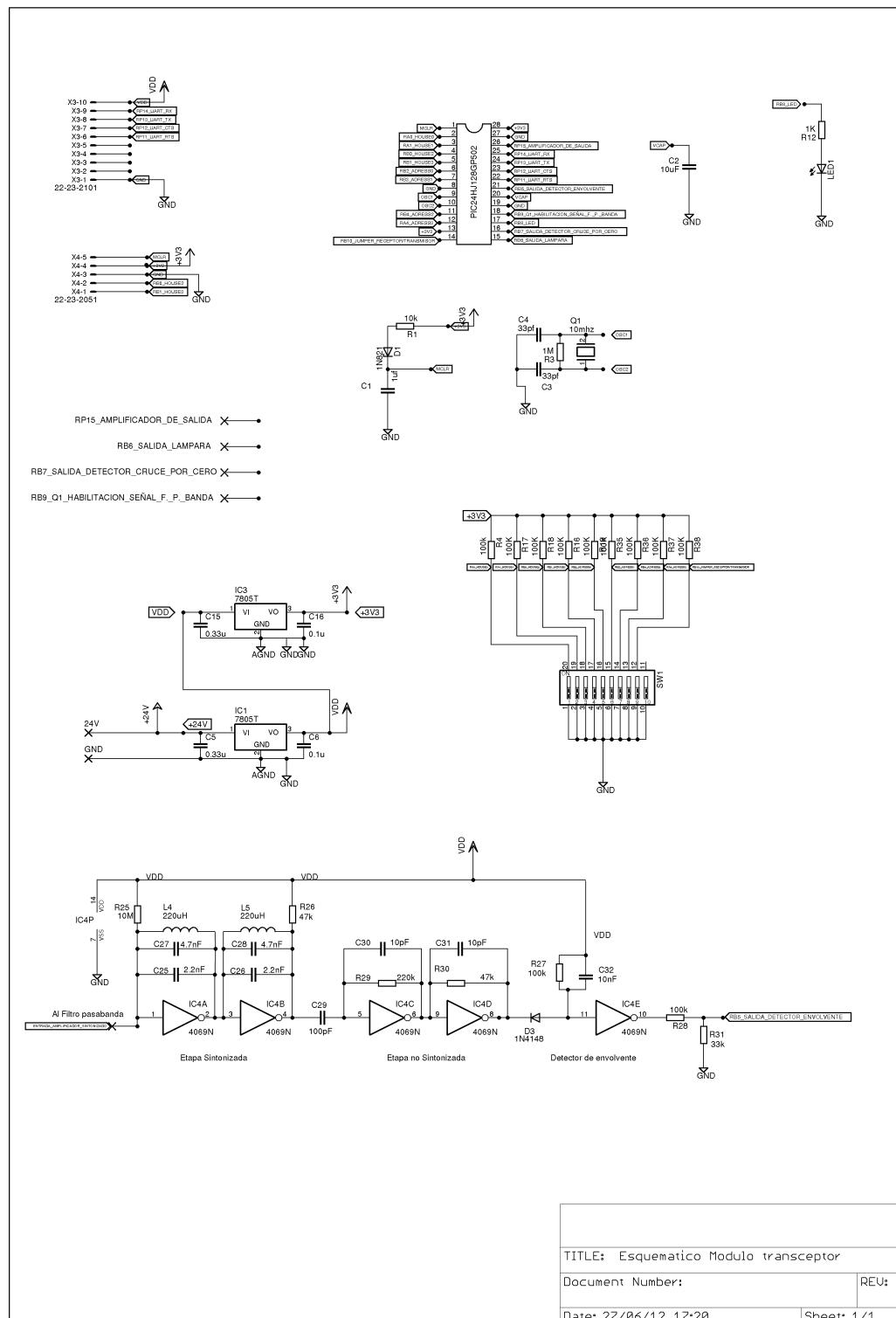


Figura A.2: Diagrama esquemático del módulo Transceptor

APÉNDICE A. DIAGRAMAS ESQUEMÁTICOS DEL PROYECTO

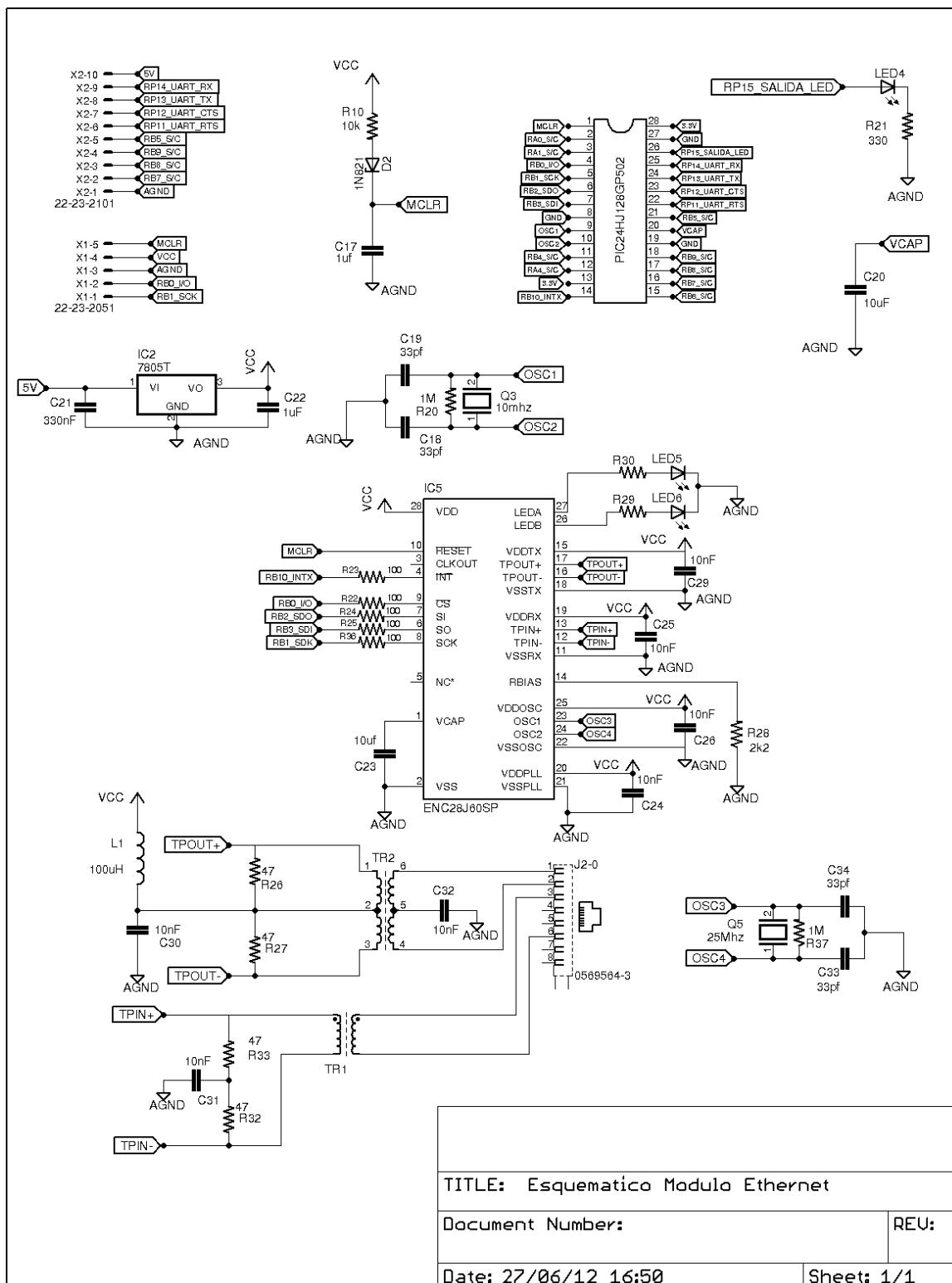


Figura A.3: Diagrama esquemático del módulo Ethernet

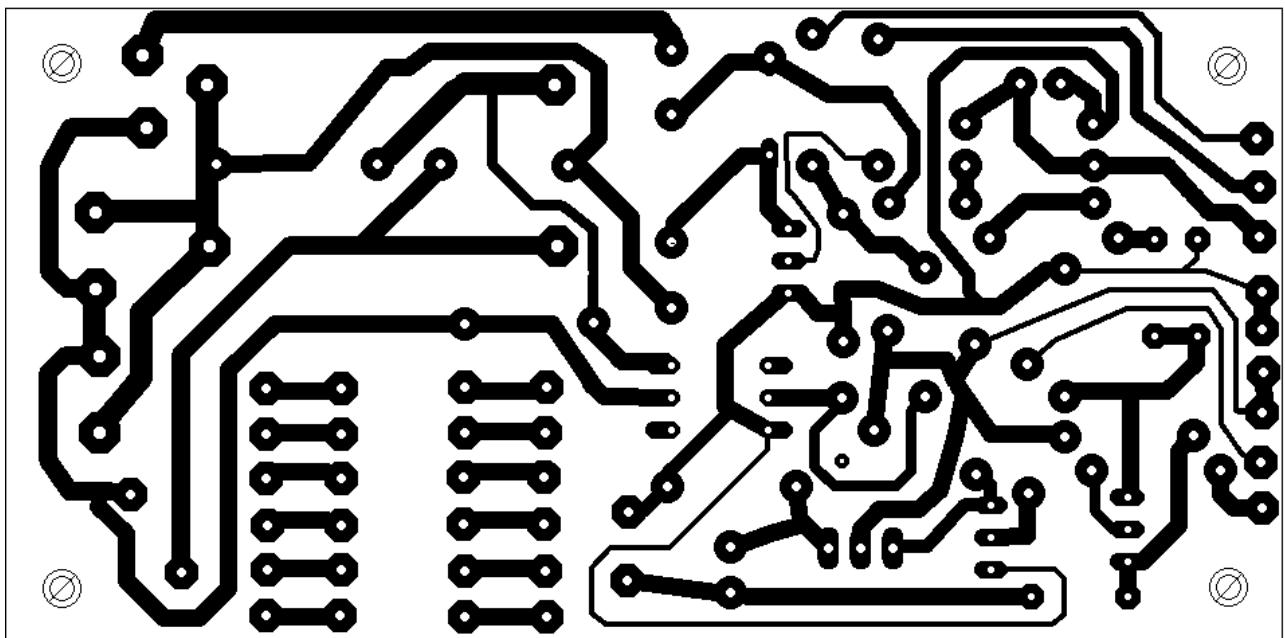


Figura A.4: Circuito impreso del módulo de alta tensión

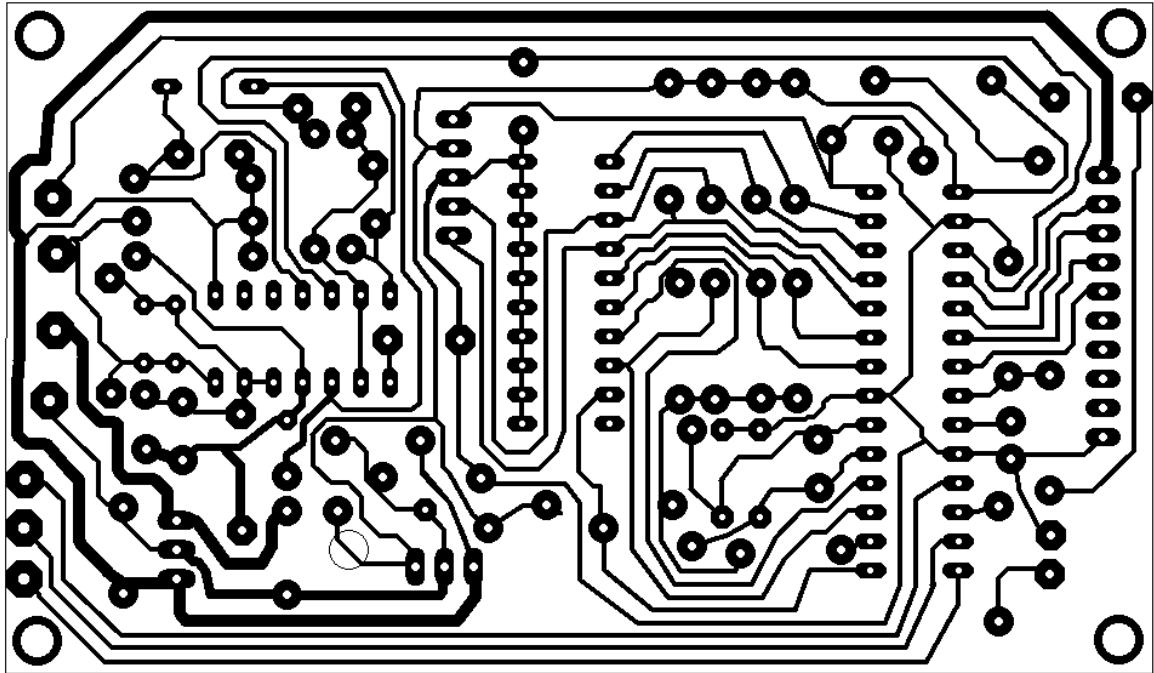


Figura A.5: Circuito impreso del módulo transceptor

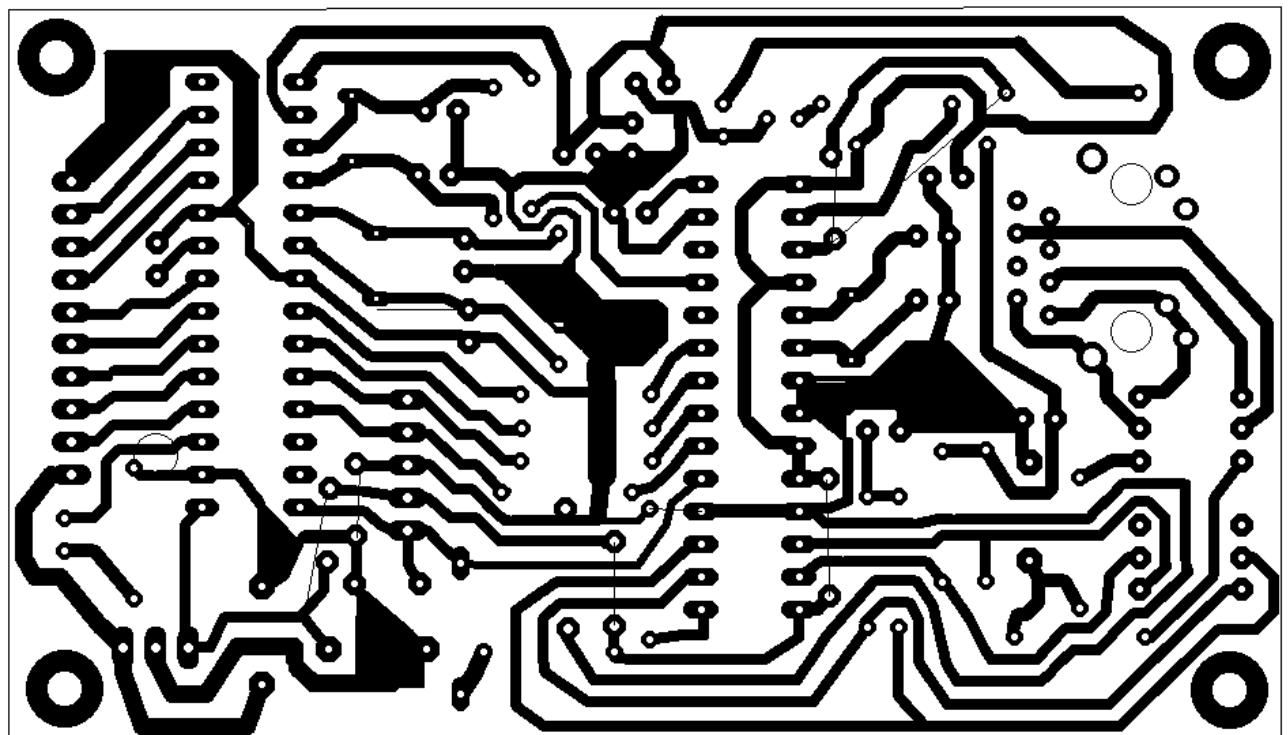


Figura A.6: Circuito impreso del módulo Ethernet

# Bibliografía

- [ARP] *RFC 826 Ethernet Address Resolution Protocol(ARP)*.
- [Bur2 ] "Jon Burroughs", "X-10 Home Automation Using the PIC16F877A ", "2002 ".
- [Cor99] "Fairchild Semiconductor Corporation", *CD4069UBC Inverter Circuits*, "1999".
- [Cor00] "Fairchild Semiconductor Corporation", "TIP31 Series(TIP31/31A/31B/31C)", "2000".
- [Dep69] Electronic Industries Association. Engineering Dept., *EIA standard RS-232-C: Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchang*, 1969.
- [DHC] *RFC 1541 Dynamic Host Configuration Protocol (DHCP)*.
- [FTP] *RFC 959 File Transfer Protocol (FTP)*.
- [HTT] *RFC 2616 Hypertext Transfer Protocol (HTTP) 1.1*.
- [ICM] *RFC 792 Internet Control Message Protocol (ICMP)*.
- [Inc02] Microchip Technology Inc., *AN833, The Microchip TCP/IP Stack*, 2002.
- [Inc05] Microchip Technology Inc., *Explorer 16 Development Board Userâs Guide*, 2005.
- [Inc08] Microchip Technology Inc., *ENC28J60 Data Sheet Stand-Alone Ethernet Controller with SPI Interface*, 2008.
- [Inc09] "Microchip Technology Inc.", "PIC24HJ32GP302/304, PIC24HJ64GPX02/X04 and PIC24HJ128GPX02/X04 Data Sheet", "2009".
- [Inc10a] Microchip Technology Inc., *ENC424J600/624J600 Data Sheet Stand-Alone 10/100 Ethernet Controller with SPI or Parallel Interface*, 2010.
- [Inc10b] "Texas Instruments Incorporated", "POSITIVE-VOLTAGE REGULATORS uA78M00 SERIES", "2010".
- [IP] *RFC 791 Internet Protocol (IP)*.
- [Sem99] "Vishay Semiconductors", *Optocoupler with Phototransistor Output*, "1999".
- [Sem04] "NXP Semiconductors", *1N4148; 1N4448 High-speed diodes*, "2004".
- [TCP] *RFC 793 Transmission Control Protocol (TCP)*.

## BIBLIOGRAFÍA

---

- [Ton11] "James L. Tonne", "<http://tonnesoftware.com/elsie.html>", "2011".
- [UDP] *RFC 768 User Datagram Protocol (UDP)*.
- [UNI99] INTERNATIONAL TELECOMMUNICATION UNION, *Asymmetric digital subscriber line (ADSL)*, 1999.