DISEÑO Y PROGRAMACIÓN DEL DSPIC30F4013 PARA CONEXIÓN A ETHERNET

LYNDA EVELIN ACUÑA HERNANDEZ

MIGUEL ENRIQUE MARTÍNEZ ARCE

CRISTIAN FERNANDO URIBE MEJIA

SALVADOR SANTONJA CLIMENT

RIGEL RODRIGUEZ SANCHEZ

LIZBETH SANCHEZ MONROY

Profesor

IGNACIO MIRO OROZCO

Laboratorio de Sistemas Electrónicos Digitales

ESCUELA POLITÉCNICA SUPERIOR DE ALCOY

INGENIERIA TÉCNICA EN TELECOMUNICACIONES: ESPECIALIDAD TELEMÁTICA

Marzo 2008

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

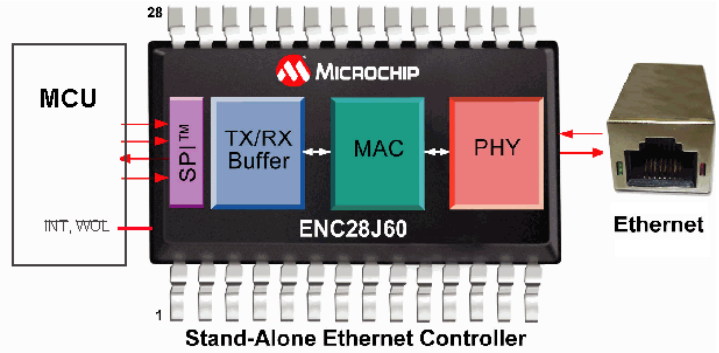
TABLA DE CONTENIDO

Página

INTRODUCCIÓN

Con el presente trabajo investigativo se busca realizar una interfaz Ethernet para el microcontrolador DSPIC30F4013, este microcontrolador no dispone de la capacidad para controlar este tipo de comunicación, por lo que necesariamente se debe utilizar un controlador Ethernet externo.

Entre los controladores disponibles en el mercado se puede encontrar el ENC28J60 de Microchip, este controlador es el que más se adecua debido a su potencia, tamaño y versatilidad además de mantenerse dentro de un mismo fabricante. No obstante al final del trabajo se exponen algunos modelos más de controladores Ethernet de otros fabricantes como el WIZ W5100 con interfaz SPI (modo 0,3), fácil implementación TCP/IP sin necesidad de Sistema Operativo, con capa MAC (Control Acceso al Medio) y PHY (Capa Física) de 10BaseT/100BaseTX incorporadas, también encontramos el RTL8111C Controlador Gigabit Ethernet con SPI y PCI (Interconexión de Componentes Periféricos) Express™ integrado.



Este trabajo incluye la parte de Hardware, con el esquema de conexionado ya la explicación de todas sus partes; y la parte de Software, parte de vital importancia para la comunicación Ethernet y el control de su protocolo. Hay que tener claro que no se están conectando dos micros con un cable de red, con lo que se podría utilizar un protocolo personalizado y comunicarlos como se quisiera; sino que se va a usar Internet, una red pública y estandarizada, por lo que para que los datos

circulen correctamente por ella se debe cumplir con el protocolo IP y procesar correctamente las solicitudes de control y descubrimiento de la red (ICMP y ARP).

El software de Microchip incluye muchas características interesantes, pero tiene algunas limitaciones y no es muy sencillo de utilizar. También hay una pequeña librería para el uso del controlador que se encuentra con el compilador **mikroC**, pero no soporta la fragmentación de paquetes y, no siendo posible modificarla, no corresponde mucho a la realización de un Server HTTP ni de cualquier aplicación que tenga un intercambio de datos considerable (el límite es más o menos de 1.4KB). Realizar un propio software personalizado (a partir de ahora **stack**) es una forma óptima de aprender los mecanismos que permiten el funcionamiento de las redes, pero principalmente ayuda la implementación de protocolo y funcionalidad no previstos en otros stack.

1. DISEÑO DSPIC30F4013 PARA CONEXIÓN A ETHERNET

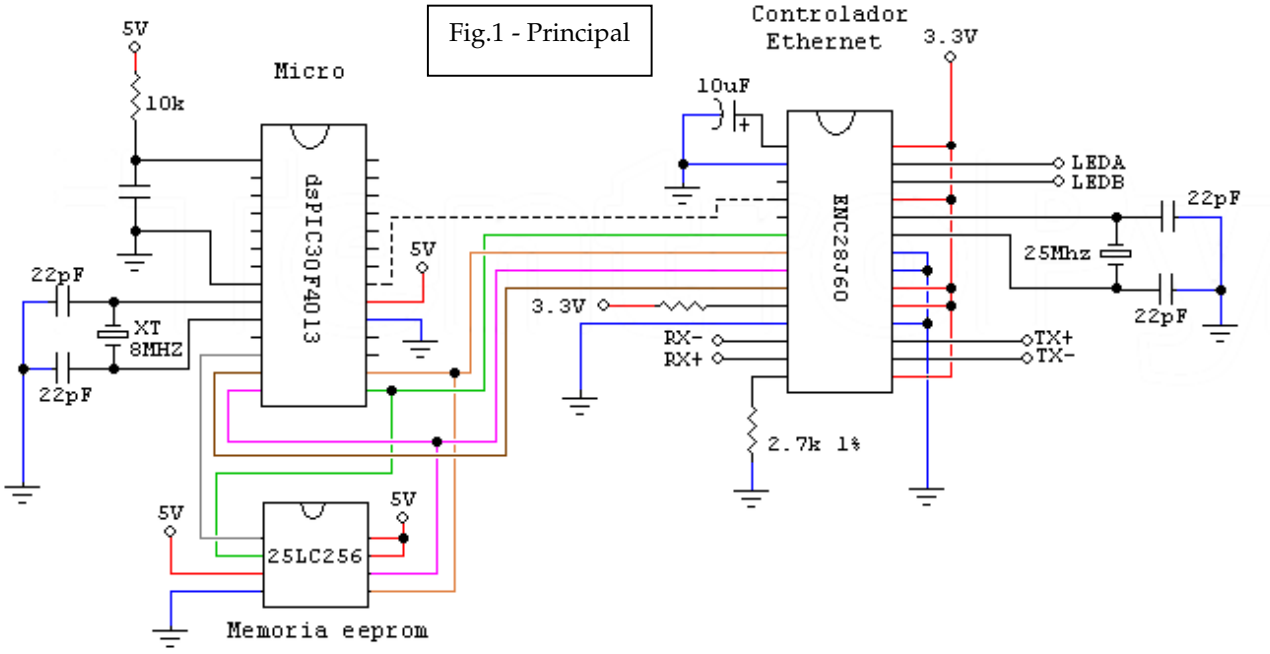


Fig.1 - Principal

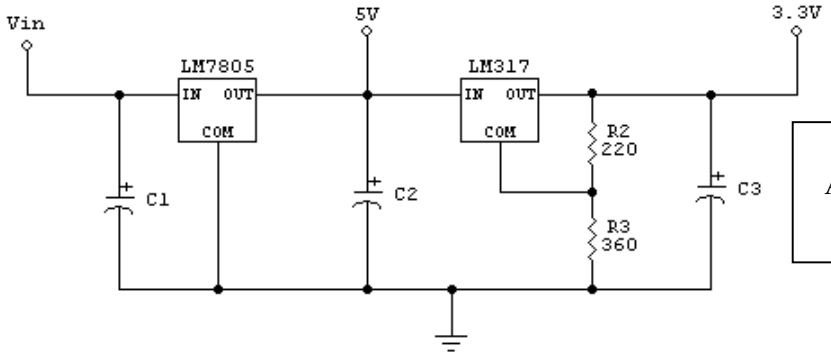


Fig.2
Adaptador de nivel

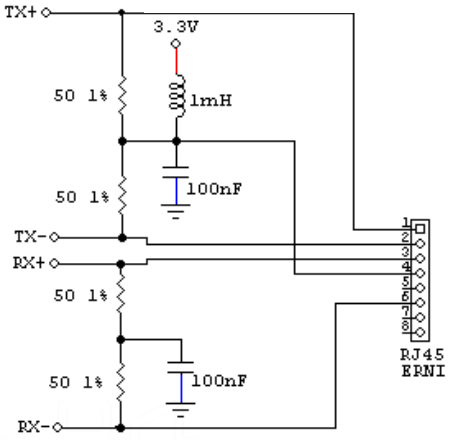


Fig.3 - Conector

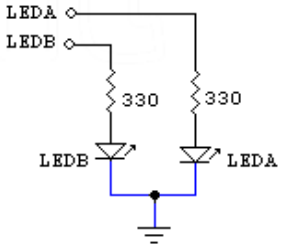
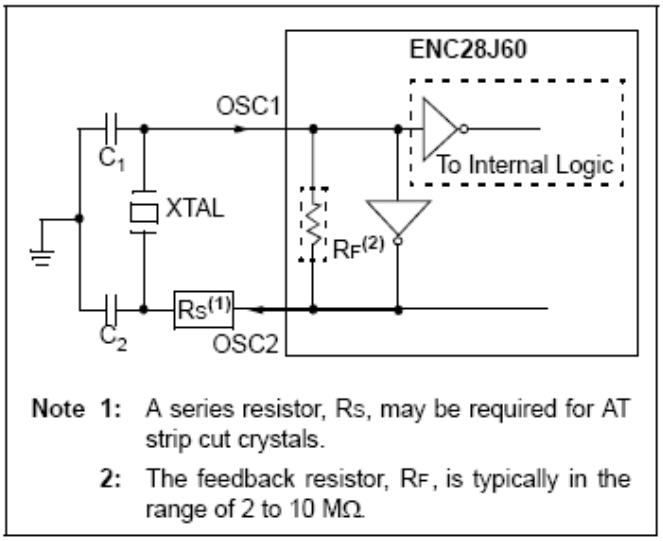


Fig.4 – Leds estado

En este diagrama (fig.1) se puede observar cada uno de los elementos y pines utilizados para hacer posible la conexión a Ethernet del microcontrolador **DSPIC30F4013.** Para ello, como ya hemos comentado, es necesario utilizar el controlador **ENC28J60**, debido a que el microcontrolador por si solo no puede manejar la conexión a Ethernet, por lo tanto más adelante se describirán las características más importantes tanto del microcontrolador como del controlador ethernet, y sus funcionalidades. Aquí se explicará la forma en que se debe realizar la conexión para que funcione correctamente.

Por una parte están los relojes: el microcontrolador utiliza un oscilador de cristal de 8MHZ mientras que el controlador utiliza uno de 25MHZ. Además, se añaden unos condensadores para estabilizar su frecuencia. Esto, más un pequeño circuito interno del integrado, conforman el reloj en sí mismo.

CRYSTAL OSCILLATOR OPERATION

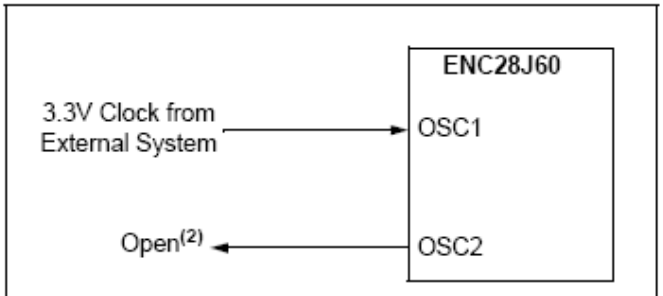


Note 1: A series resistor, Rs, may be required for AT strip cut crystals.

2: The feedback resistor, RF, is typically in the range of 2 to 10 MΩ.

Ambos dispositivos se comunican a través del puerto **SPI**, del que hablaremos más adelante, a través de los pines **SDI1**, **SDO1**, **SCK** y **Vss**. Por **SCK** tendremos una señal Fosc/4 (velocidad del micro/4), que maneja la sincronización de la comunicación, la cual controla el microcontrolador.

EXTERNAL CLOCK SOURCE(1)



Note 1: Duty cycle restrictions must be observed.

2: A resistor to ground may be used to reduce system noise. This may increase system current.

Es muy importante en este tipo de aplicaciones no introducir ruido en el circuito, pues se trabaja con transmisiones de datos y sincronismos, señales muy sensibles a estos problemas. Por lo tanto será imprescindible disponer de condensadores de desacoplo en todos los pines que conecten con tierra, con una capacidad aproximada de 100nF.

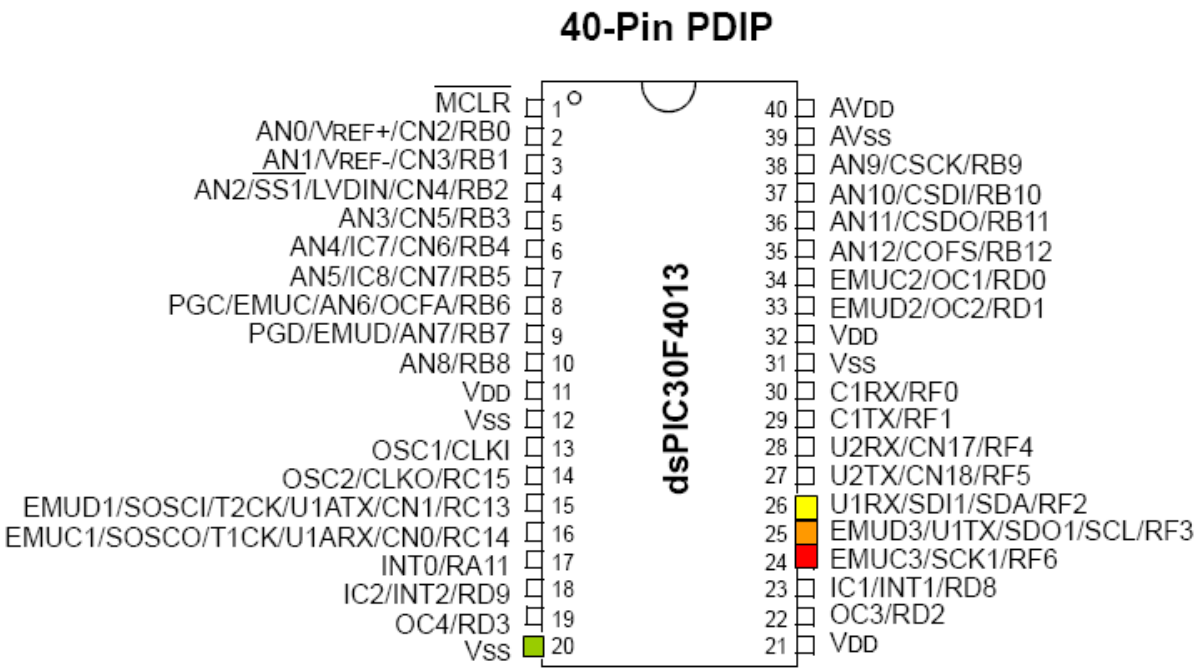
En caso de necesitar el uso de aplicaciones que consuman más memoria que la propia integrada en el ENC28J60, se puede añadir una memoria externa (por ejemplo si hay que montar una página http). En el esquemático adjunto (*fig.*1) hay montada una memoria eeprom 25LC256, pero no se utilizará en el presente trabajo.

Si en lugar del DSPIC30F4013 se utilizara un micro que trabajase exclusivamente a 5v, habría que adaptar la tensión a ambos lados del SPI, pues el ENC28J60 trabaja a 3,3v. Por lo tanto, se instalaría el adaptador de nivel (que se observa en la *fig.2*). En este caso no es necesario ya que tanto el micro como el controlador utilizan 3.3 v.

En la *fig.3* se observa el módulo de conexión a la red, basado en un conector RJ45 y unos bobinados. En la práctica se puede utilizar este montaje o incluso comprar los conectores con la circuitería de bobinado ya integrada, pero hay que tener en cuenta que estos bobinados son necesarios, pues aislan el circuito de la red, evitan reflexiones y adaptan la capacidad del cable, permitiendo una comunicación más limpia. Como el controlador ENC28J60 trabaja a 10Mbps, utilizamos sólo 4 cables del conector (los 8 se utilizan para determinadas comunicaciones especiales y para Gigabit Ethernet), por lo que los 4 pines sobrantes lse conectarán a tierra para evitar interferencias.

La *fig.4* conforma el sistema de LEDs para verificar que el dispositivo transmite y recibe correctamente.

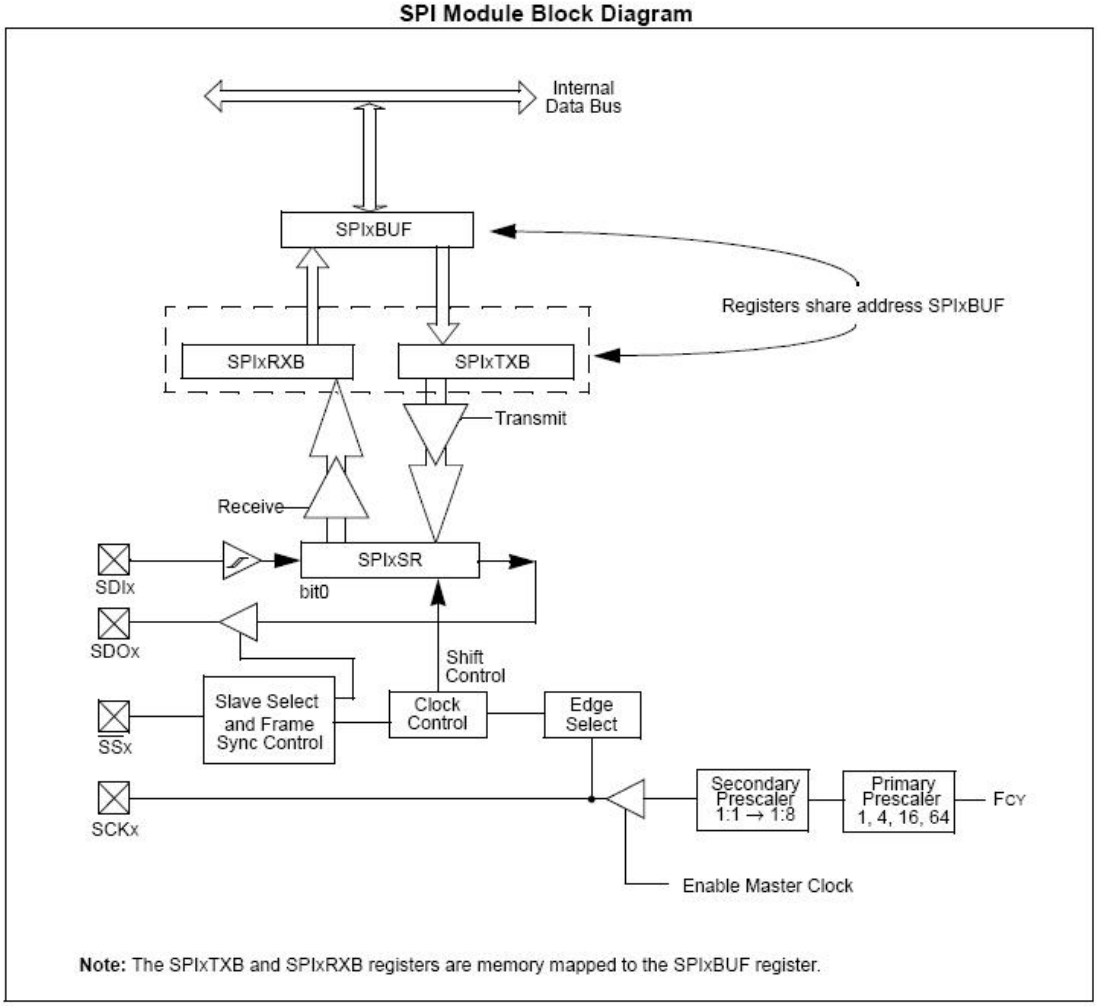
2. EL MICROCONTROLADOR DSPIC30F4013



2.1 Puertos SPI

La conexión del controlador al micro se realiza a través del puerto serie SPI. Este puerto consta de los siguientes Registros de Función Especial (SFR):

SPI Module Block Diagram



Note: The SPIxTXB and SPIxRXB registers are memory mapped to the SPIxBUF register.

- **SPIxBUF**: Dirección SFR es el espacio que se utiliza para buffer de datos que se transmiten y reciben. Esta dirección es compartida por los registros **SPIxTXB** y **SPIxRXB**.

- **SPIxCON**: Un registro de control que configura el módulo para diversos modos de operación.

- **SPIxSTAT:** Un registro que indica el estado de diversas condiciones. Además, hay un registro de 16 bits de desplazamiento, **SPIxSR**, que no es de memoria mapeada. Se utiliza para cambio de los datos de entrada y salida del puerto SPI.

Internamente el registro **SPIxBUF** en realidad se compone de dos registros **SPIxTXB** y **SPIxRXB**.

El buffer **SPIxRXB** recibe, y la transmisión la realiza el buffer **SPIxTXB**. Son dos registros unidireccionales de 16 bits. Estos registros SFR comparten la dirección **SPIxBUF**. Si un usuario quiere transmitir, escribe datos en la dirección **SPIxBUF**, e internamente los datos se escriben en el registro **SPIxTXB**.
Del mismo modo, cuando el usuario desea leer los datos, éstos se reciben en el **SPIxBUF** e internamente se pasan al registro **SPIxRXB**. Esta característica permite transferencias de datos continuos. La transmisión y la recepción se producen al mismo tiempo.

Nota: El usuario no puede escribir en el registro **SPIxTXB** o leer en el registro de la **SPIxRXB** directamente. Todas las escrituras y lecturas se realizan en el registro **SPIxBUF**.

El puerto SPI serie consta de cuatro pines:

- **SDIx**: entrada de datos en serie
- **SDOx**: salida de datos en serie
- **SCKx**: cambio de reloj de entrada o salida
- **SSx**: activa bajo esclavo o selecciona marco de sincronización de E / S de impulsos

Nota: El módulo SPI puede configurarse para funcionar a 3 o 4 pines. En la modalidad de 3 pines, el pin **SSx** no se utiliza.

2.2 Descripción del Funcionamiento

Cada módulo consiste en un registro de 16 bits de desplazamiento, **SPIxSR** (donde x = 1 o 2), utilizado para transferir los datos y un buffer de registro de control **SPIxBUF**. Un registro **SPIxCON**, configura el módulo. Además, un registro de estado, **SPIxSTAT**, indica el estado de diversas condiciones.

La serie consta de 4 pines, ya enumerados antes. En el modo de operación de Master, **SCK** es un reloj de salida, pero en Modo esclavo, es un reloj de entrada. Una serie de ocho (8) o dieciséis (16) pulsos de reloj cambia los bits de la **SPIxSR** al pin **SDOx** o al pin **SDIx** respectivamente.

Se genera una interrupción cuando la transferencia se ha completado, correspondiente al conjunto bit (SPI1IF o SPI2IF). Esta interrupción puede ser desactivada a través de una interrupción (SPI1IE o SPI2IE).

2.3 Saturación

Si el buffer de recepción está lleno de nuevos datos cuando se transfiere de **SPIxSR** a **SPIxBUF**, el módulo establece la SPIROV bit, que indica una condición de desbordamiento. La transferencia de los datos de **SPIxSR** a **SPIxBUF** no se completa y los nuevos datos se pierden.

El módulo no responde a las transiciones SCL mientras SPIROV es '1'. Por software se desactiva el módulo hasta que se lee **SPIxBUF**.

2.4 Transmisión

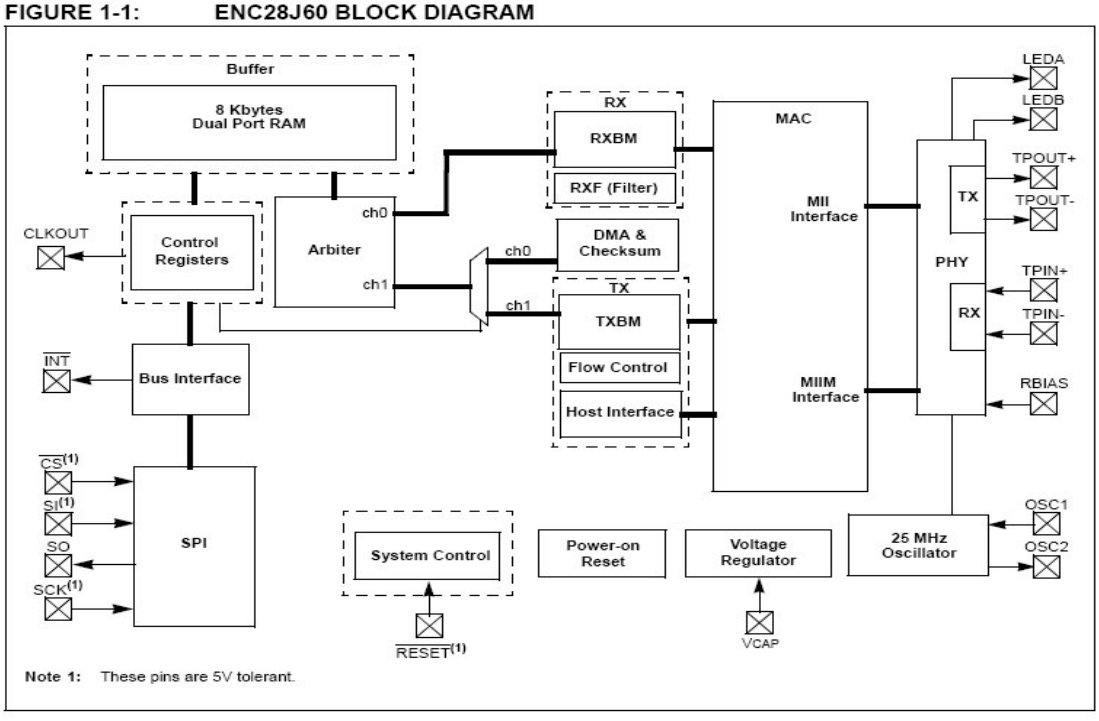
El usuario escribe los datos en **SPIxBUF**. Cuando la transferencia maestro-esclavo se haya completado, el contenido del registro de desplazamiento (**SPIxSR**) se traslada al buffer de recepción.

3. CONTROLADOR DE ETHERNET ENC28J60

El ENC28J60 es un controlador independiente de Ethernet, con interfaz SPI. Sirve como una interfaz de red Ethernet para cualquier micro equipado con SPI. El ENC28J60 resuelve todo lo relacionado con IEEE 802.3 (Ethernet) incorporando unos esquemas de filtración para limitar los paquetes entrantes.
También proporciona un módulo interno de acceso directo a la memoria para mayor rendimiento del procesado de datos rápido y el cálculo asistido por hardware para el checksum (CRC) o suma de comprobación (control de errores), que se utiliza en varios protocolos de red.

Para conectar un microcontrolador a una red Ethernet sólo se necesita un ENC28J60, dos transformadores y algunos componentes pasivos. El dispositivo también contiene otros bloques de ayuda, tales como el oscilador, regulador de voltaje, traductores para proporcionar 5V I/O en caso de ser necesario, y lógica de control de sistema.

FIGURE 1-1: ENC28J60 BLOCK DIAGRAM

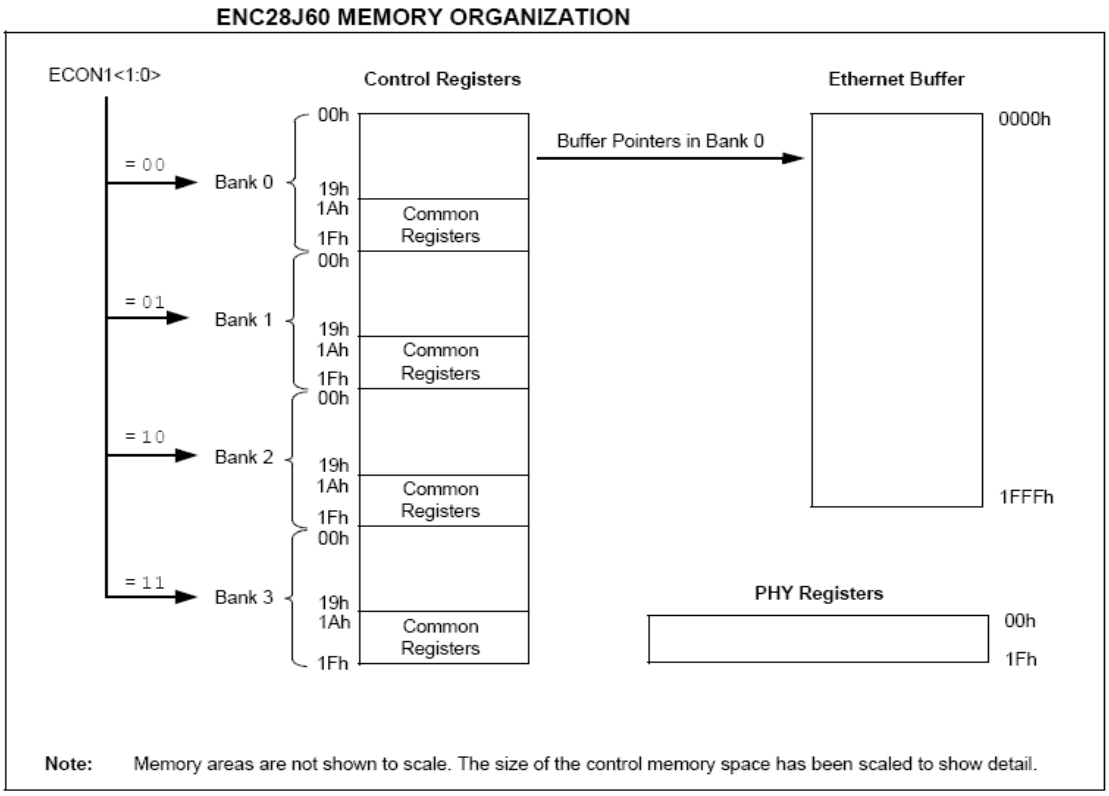


3.1 Organización de la Memoria

Toda la memoria en el ENC28J60 se pone en ejecución como RAM estática. Hay tres tipos de memoria en el ENC28J60:

1. Registros de control.
2. Memoria de Ethernet.
3. Registros PHY (Registros capa física)

La memoria contiene los registros que se utilizan para la configuración, el control y la recuperación del estado del ENC28J60. Los registros de control pueden leerse y escribirse directamente por la interfaz de SPI. El buffer de Ethernet contiene la memoria para transmisión y recepción, usada por el controlador de Ethernet en una sola memoria física. El tamaño de cada área de memoria (la destinada a recepción y la destinada a transmisión) se programan también a través de la interfaz SPI. Los registros de PHY (Capa física) se utilizan para la configuración y el control.



3.1.1 Registros de Control

Al escribir en estos registros se controla la operación de la interfaz, mientras que la lectura de los registros permite que el micro supervise la operación. La memoria del registro de control se reparte en cuatro bancos, seleccionados por los bits **BSEL1** y **BSEL0**, en el registro **ECON1**. Cada banco es de 32 bytes y tratados por un valor de dirección de 5 bits. Las cinco posiciones pasadas (1Bh a 1Fh) de todos los bancos señalan a un sistema común de registros: **EIE**, **EIR**, **ESTAT**, **ECON2** y **ECON1**. Estos registros dominantes son usados para controlar y supervisar la operación del dispositivo. Al leer y escribir los registros que contienen bits reservados, debemos observar cualquier regla indicada en la definición del registro. Los registros de control para el ENC28J60 genéricamente se agrupan como **ETH**, **MAC** y **MII (**Interfaz de medios independientes**)**. Los nombres de registro que empiezan con **E** pertenecen al grupo de **ETH**. De igual forma, nombres de registros comenzados con **MA** pertenezca al grupo y a los registros del módulo **MAC**, y prefijados con **MI** pertenecen al grupo **MII (**Interfaz de medios independientes**).**

ENC28J60 CONTROL REGISTER MAP

Bank 0 Address	Name	Bank 1 Address	Name	Bank 2 Address	Name	Bank 3 Address	Name
00h	ERDPTL	00h	EHT0	00h	MACON1	00h	MAADR5
01h	ERDPTH	01h	EHT1	01h	Reserved	01h	MAADR6
02h	EWRPTL	02h	EHT2	02h	MACON3	02h	MAADR3
03h	EWRPTH	03h	EHT3	03h	MACON4	03h	MAADR4
04h	ETXSTL	04h	EHT4	04h	MABBIPG	04h	MAADR1
05h	ETXSTH	05h	EHT5	05h	—	05h	MAADR2
06h	ETXNDL	06h	EHT6	06h	MAIPGL	06h	EBSTSD
07h	ETXNDH	07h	EHT7	07h	MAIPGH	07h	EBSTCON
08h	ERXSTL	08h	EPMM0	08h	MACLCON1	08h	EBSTCSL
09h	ERXSTH	09h	EPMM1	09h	MACLCON2	09h	EBSTCSH
0Ah	ERXNDL	0Ah	EPMM2	0Ah	MAMXFLL	0Ah	MISTAT
0Bh	ERXNDH	0Bh	EPMM3	0Bh	MAMXFLH	0Bh	—
0Ch	ERXRDPTL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRDPTH	0Dh	EPMM5	0Dh	Reserved	0Dh	—
0Eh	ERXWRPTL	0Eh	EPMM6	0Eh	Reserved	0Eh	—
0Fh	ERXWRPTH	0Fh	EPMM7	0Fh	—	0Fh	—
10h	EDMASTL	10h	EPMCSL	10h	Reserved	10h	—
11h	EDMASTH	11h	EPMCSH	11h	Reserved	11h	—
12h	EDMANDL	12h	—	12h	MICMD	12h	EREVID
13h	EDMANDH	13h	—	13h	—	13h	—
14h	EDMADSTL	14h	EPMOL	14h	MIREGADR	14h	—
15h	EDMADSTH	15h	EPMOH	15h	Reserved	15h	ECOCON
16h	EDMACSL	16h	Reserved	16h	MIWRL	16h	Reserved
17h	EDMACSH	17h	Reserved	17h	MIWRH	17h	EFLOCON
18h	—	18h	ERXFCON	18h	MIRDL	18h	EPAUSL
19h	—	19h	EPKTCNT	19h	MIRDH	19h	EPAUSH
1Ah	Reserved	1Ah	Reserved	1Ah	Reserved	1Ah	Reserved
1Bh	EIE	1Bh	EIE	1Bh	EIE	1Bh	EIE
1Ch	EIR	1Ch	EIR	1Ch	EIR	1Ch	EIR
1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT

3.1.2 Buffer de Recepción

Después que la memoria señalada por **ERXND** se escriba; el hardware escribirá automáticamente el byte siguiente de datos recibidos a la memoria señalada por **ERXST**. Consecuentemente, el hardware de recepción nunca se escribirá fuera de los límites del primero en salir o en entrar.

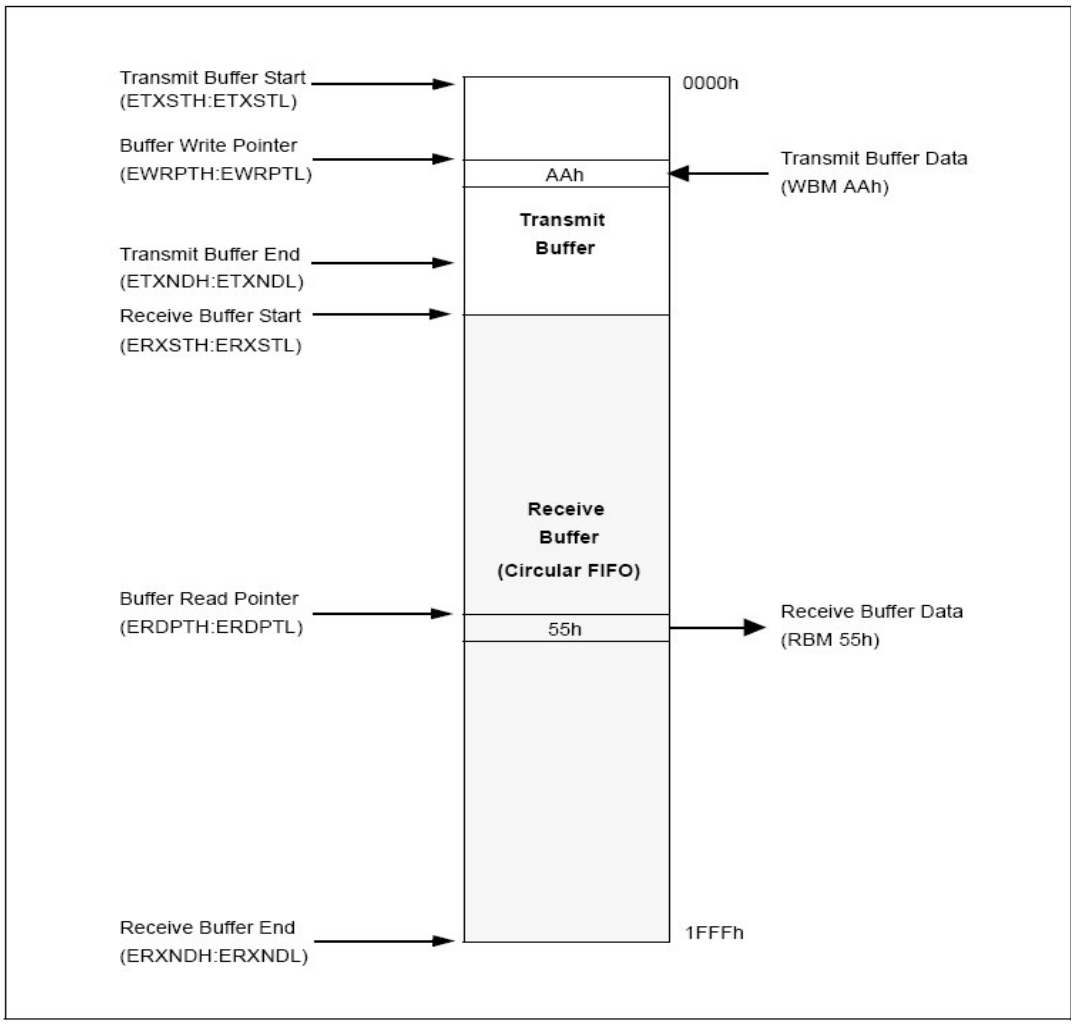
3.1.3 Buffer de Transmisión

Cualquier espacio dentro de la memoria 8-Kbyte que no se programa como parte del buffer de recepción, se considera buffer de transmisión. La responsabilidad de manejar donde se sitúan los datos en el buffer de transmisión pertenece al micro. Siempre que el micro decida transmitir un paquete, los indicadores de **ETXST** y de **ETXND** se programan con las direcciones que especifican donde, dentro del buffer de transmisión, el paquete a transmitir estará situado. El hardware no comprueba que las direcciones de inicio y fin no se solapen con el buffer de recepción.

3.1.4 DMA Acceso al Buffer

El controlador DMA (acceso directo de memoria) integrado debe leer en el buffer cuando calcular una suma de comprobación (CRC) y la debe leer y escribir al buffer cuando copia memoria. El acceso directo de memoria sigue siendo el mismo en los accesos de SPI. Mientras lea secuencialmente, estará conforme a la condición que se desarrolla en el extremo del buffer de recepción.

FIGURE 3-2: ETHERNET BUFFER ORGANIZATION



3.1.5 Registros PHY (Registros de capa física)

Los registros PHY (Capa física) proporcionan la configuración y el control del módulo PHY, así como la información de estado sobre su operación. Todos los registros de PHY (Capa física) son de 16 bits. Hay un total de 32 direcciones; sin embargo, se ponen en ejecución solamente 9.

Todas las localizaciones reservadas se deben escribir como 0. El ETH, el módulo MAC y el MII **(**Interfaz de medios independientes**)** los registros de control, o la

memoria del buffer, los registros de PHY (capa física) no son accesibles directamente a través del interfaz de control de SPI.

En su lugar, el acceso se consigue a través de un sistema especial de los registros de control del módulo MAC. Estos registros de control se refieren como los registros MII **(**Interfaz de medios independientes**)**.

3.2. SPI (Serial de Interfaz Periférico)

El ENC28J60 está diseñado para conectar directamente con el SPI disponible en muchos microcontroladores. La puesta en práctica usada en este dispositivo soporta el modo “0.0” de SPI solamente. Además, el puerto de SPI requiere que SCK esté en marcha lenta.

3.2.1 Sistema de Instrucciones del SPI

La operación del ENC28J60 depende enteramente de los comandos dados por un controlador externo del host (equipo anfitrión) sobre la interfaz de SPI. Estos comandos toman la forma de instrucciones, de uno o más bits, que se utilizan para tener acceso a los espacios de la memoria de control y del buffer de Ethernet. Las instrucciones consisten en un **opcode** (código de operación) de 3 bits, seguido por un argumento de 5 bits que especifican cualquier dirección de registro o una constante de datos.

Las instrucciones del campo de bits también se siguen por uno o más bits de datos. Un total de siete instrucciones se ponen en ejecución en el ENC28J60.

SPI INSTRUCTION SET FOR THE ENC28J60

Instruction Name and Mnemonic	Byte 0								Byte 1 and Following							
	Opcode			Argument					Data							
Read Control Register (RCR)	0	0	0	a	a	a	a	a	N/A							
Read Buffer Memory (RBM)	0	0	1	1	1	0	1	0	N/A							
Write Control Register (WCR)	0	1	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Write Buffer Memory (WBM)	0	1	1	1	1	0	1	0	d	d	d	d	d	d	d	d
Bit Field Set (BFS)	1	0	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Bit Field Clear (BFC)	1	0	1	a	a	a	a	a	d	d	d	d	d	d	d	d
System Reset Command (Soft Reset) (SRC)	1	1	1	1	1	1	1	1	N/A							

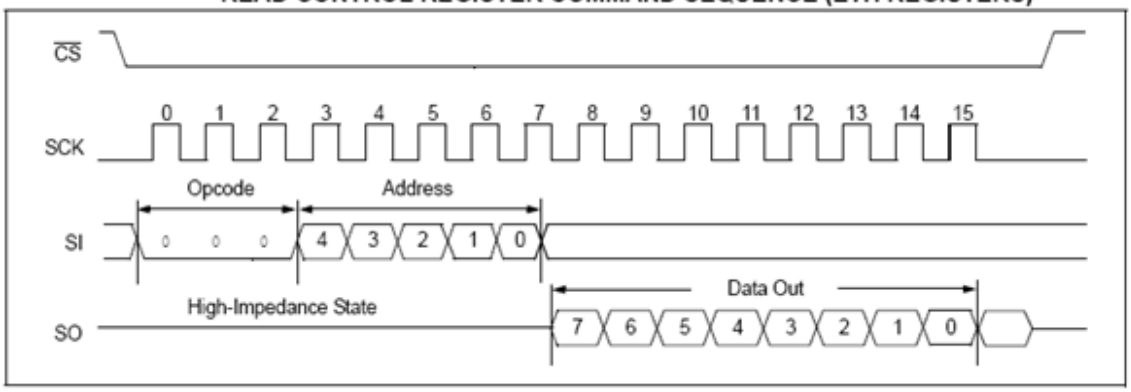
Legend: a = control register address, d = data payload.

3.2.2 Comando de Registro de Control

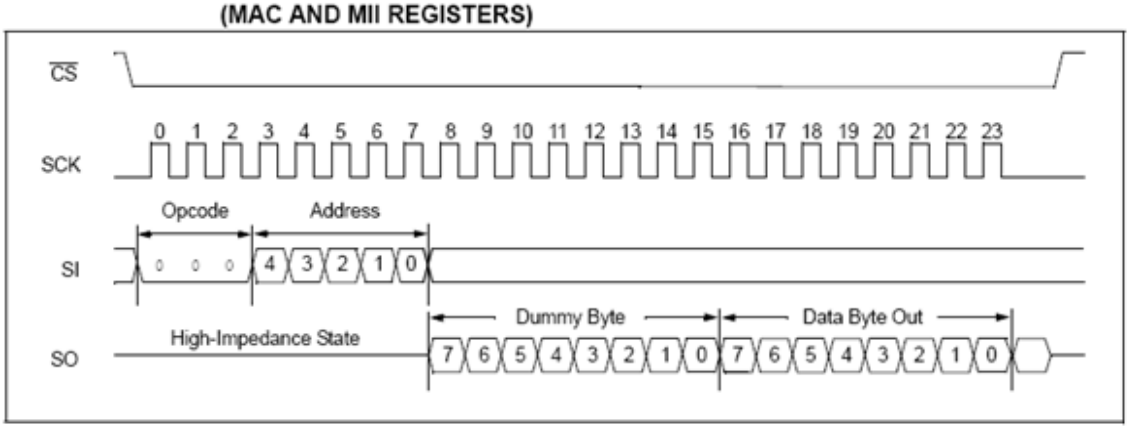
El comando leído del registro de control (RCR) permite que el controlador del host (equipo anfitrión) lea ETH, el módulo MAC y MII **(**Interfaz de medios independientes**)** en cualquier orden. El contenido de los registros de PHY (Capa física) se lee vía interfaz del registro MII **(**Interfaz de medios independientes**)**, esta interfaz ofrece un modo flexible de soportar conexiones de 100Mbps.

El **opcode** (código de operación) del RCR entonces se envía al ENC28J60, seguido por una dirección del registro de 5 bits (A4 con A0). La dirección de 5 bits identifica los 32 registros de control del banco actual.

READ CONTROL REGISTER COMMAND SEQUENCE (ETH REGISTERS)



READ CONTROL REGISTER COMMAND SEQUENCE
(MAC AND MII REGISTERS)



3.3 Inicialización

Antes de que el ENC28J60 se pueda utilizar para transmitir y recibir los paquetes, deben realizarse ciertos ajustes al dispositivo para ser inicializado. Dependiendo del uso, algunas opciones de la configuración pueden necesitar ser cambiadas. Normalmente, estas tareas se pueden lograr después de inicializar y no necesitan ser cambiadas.

3.4 TRANSMISION Y RECEPCION DE LOS PAQUETES

3.4.1 Transmisión de Paquetes

Dentro del ENC28J60 se generarán automáticamente los campos del preámbulo y del delimitador del Start-Of-Frame al transmitir. Además, el módulo MAC puede generar cualquier padding (relleno para cumplir el tamaño de trama mínimo que exige el protocolo) y el CRC (control de errores) si está configurado. El controlador del host (equipo anfitrión) debe generar y escribir el resto de los campos en la memoria del buffer para la transmisión. Cada bit del control del paquete se organiza según las indicaciones. Antes de transmitir los paquetes, los registros del módulo MAC que se alteran se deben inicializar siguiendo ciertos parámetros.

3.4.2 Muestra de la transmisión de un paquete

Para transmitir un paquete, el controlador del host (equipo anfitrión) debe:

1. Programar apropiadamente el indicador de ETXST para señalar la posición de memoria. Señalará el bit de control del paquete. En el ejemplo, sería programado a 0120h. Se recomienda incluso que se utilice una dirección para ETXST.

2. Utilizar el comando de WBM SPI, la dirección de destino (MAC address), la dirección de la fuente, el type/length (Tipo/longitud) y el campo DATA.

3. Programar apropiadamente el indicador de ETXND. Debe señalar al bit de comienzo del campo DATA. En el ejemplo, sería programado a 0156h.

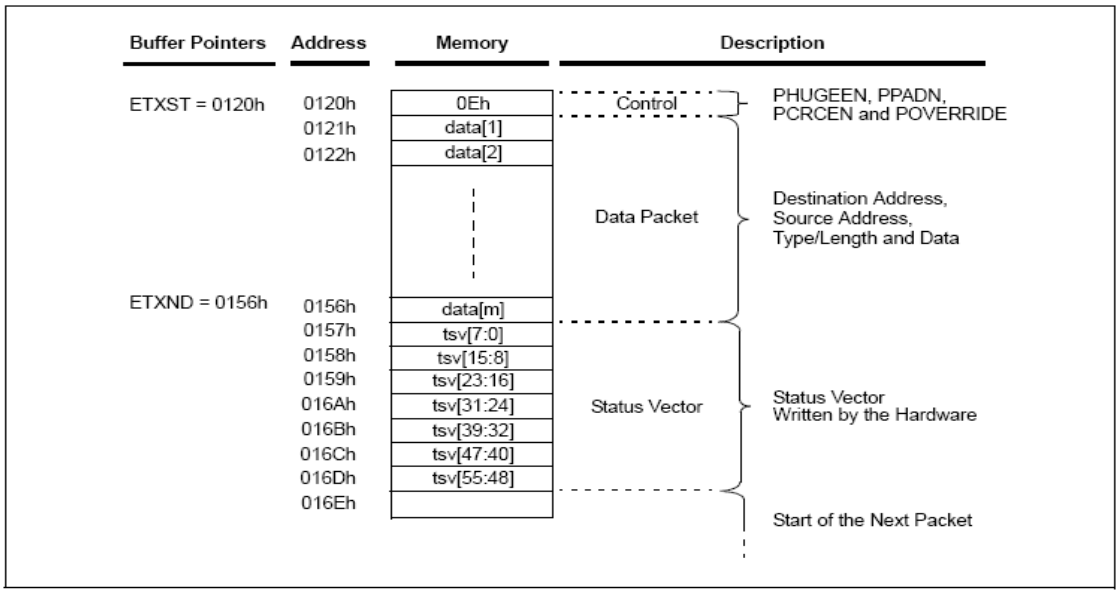
4. Comenzar el proceso de la transmisión fijando ECON1.TXRTS.

Si una operación de acceso directo de memoria estaba en marcha mientras que el bit de TXRTS fue fijado, el ENC28J60 esperará hasta que la operación del acceso directo de memoria este completa antes de transmitir el paquete. Es posible que

retrase el acceso directo de la memoria y de la transmisión pues comparten el mismo puerto del acceso a la memoria. Si el bit de DMAST en ECON1 se fija después de que TXRTS el acceso directo de memoria esperará hasta que el bit de TXRTS llegue a estar claro antes de hacer cualquier procedimiento. Mientras que la transmisión este en marcha.

Además, ningunos de los bytes a ser transmitido se deben leer o escribir con SPI. Si el controlador del host (equipo anfitrión) desea cancelar la transmisión, puede despejar el bit de TXRTS. Cuando el paquete transmitido es abortado debido a un error o una cancelación, el bit de ECON1.TXRTS se liberará. Los indicadores de ETXST y de ETXND no serán modificados. Para comprobar si el paquete fue transmitido con éxito, el bit de ESTAT.TXABRT debe ser leído. Si fue fijado, el controlador del host (equipo anfitrión) puede revisar el bit de ESTAT.LATECOL además de los varios campos en el vector del estado del transmisor para determinar la causa.

SAMPLE TRANSMIT PACKET LAYOUT



3.4.3 Recepción de paquetes

Si se asume que se ha inicializado el buffer de recepción, el módulo MAC se ha configurado correctamente y los filtros de la recepción se han configurado para recibir los paquetes de Ethernet, el controlador del host (equipo anfitrión) debe:

1. Si se desea una interrupción siempre que se reciba un paquete, se fija EIE.PKTIE y EIE.INTIE.

2. Si se desea una interrupción siempre que un paquete sea eliminado debido al espacio del buffer (desbordamiento de buffer), se debe fijar EIR.RXERIF, EIE.RXERIE y EIE.INTIE

3. Se permite la recepción fijando ECON1.RXEN.

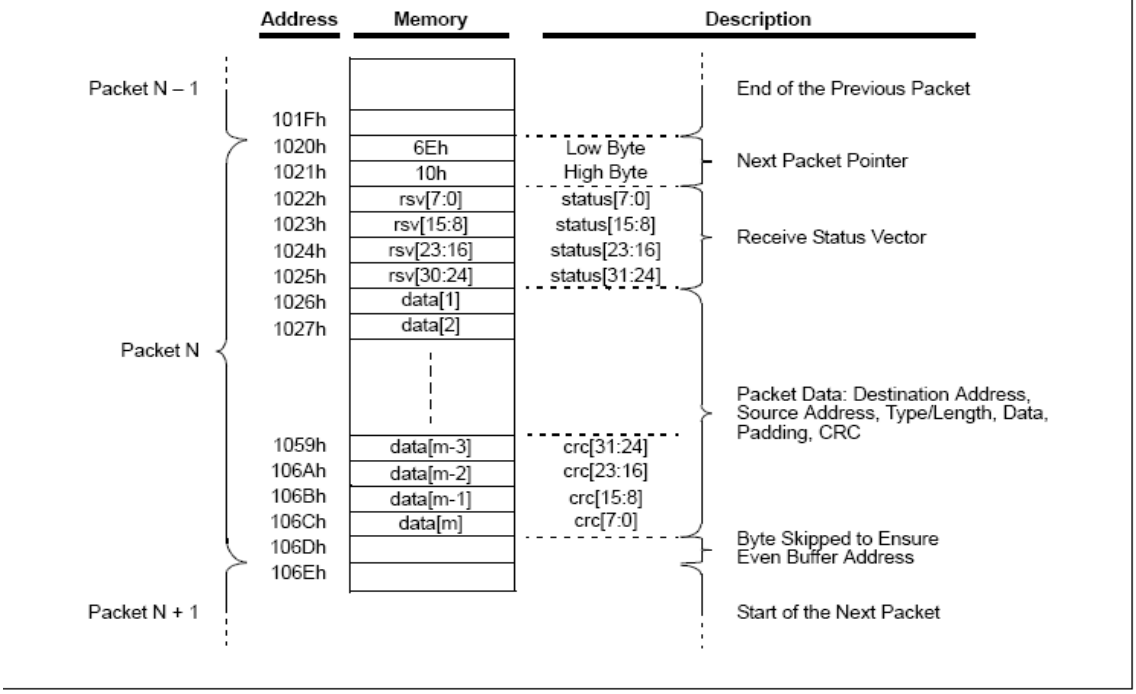
Para evitar que lleguen paquetes inesperados, se recomienda que despeje RXEN antes de modifcar la configuración del filtro de recepción (ERXFCON) y la dirección MAC (Dirección física). Después de que permitan la recepción, los paquetes que no se filtran serán guardados en el buffer.

Cualquier paquete que no resuelva los criterios necesarios del filtro, será desechado y el controlador del host (equipo anfitrión) no tendrá ningún medio de identificar que un paquete fue desestimado. Cuando un paquete se acepta y se guarda en el buffer, el registro de EPKTCNT se incrementará y el bit de EIR.PKTIF será fijado.

3.4.4 Disposición del Paquete

Cada paquete esta precedido por una cabecera de 6 bytes, adicionalmente incluye un vector de estado de recepción que contiene el registro del tamaño de cada paquete. Si el bit anterior en el paquete termina en una dirección impar del valor, el hardware agregará automáticamente un bit de padding (relleno).

SAMPLE RECEIVE PACKET LAYOUT



3.4.5 Paquetes Recibidos

Para procesar el paquete, el controlador del host (equipo anfitrión) utilizará normalmente la lectura del RBM SPI. El controlador del host (equipo anfitrión) se ahorrará el siguiente indicador de paquete, es decir, cualquier bit necesario del vector de estado de recepción; después procederá a leer el contenido real del paquete. Si se fija ECON2.AUTOINC, se podrá leer secuencialmente el paquete entero sin modificarlo en los registros de ERDPT. Es decir dada la dirección del origen del paquete, se le dará un uso lógico.

3.4.6 Liberar espacio del buffer de recepción

Después de que el controlador del host (equipo anfitrión) haya procesado un paquete (o parte del paquete) y desea liberar espacio del buffer usado por los

datos procesados, el controlador del host (equipo anfitrión) debe avanzar hacia el indicador leído en el buffer de recepción, ERXRDPT.

El ENC28J60 procura siempre sobrescribir el buffer de recepción leyendo la localización del indicador, el EIR.RXERIF será fijado y una interrupción será generada (si está permitido). De este modo, el hardware nunca sobrescribirá los paquetes sin procesar. Normalmente, el ERXRDPT será avanzado al valor señalado por el indicador siguiente del paquete que preceda el vector de estado de recepción para el paquete actual. El buffer de recepción internamente se protege para evitar que el indicador se mueva cuando solamente un bit es actualizado con el SPI. Para mover ERXRDPT, el controlador del host (equipo anfitrión) debe escribir a ERXRDPTL.

Cuando el controlador del host (equipo anfitrión) escribe a ERXRDPTH, el bit internamente estará protegido en el registro de ERXRDPTL al mismo tiempo, los bits de ERXRDPT se pueden leer en cualquier orden. Además de avanzar el buffer de recepción lee el indicador, después de que cada paquete sea tratado, el controlador del host (equipo anfitrión) debe escribir un bit de ECON2.PKTDEC.

Esto causará que el registro de EPKTCNT disminuya un bit. Después de disminuir, si EPKTCNT es 0, EIR.PKTIF estará automáticamente libre. Si no, seguirá el sistema, indicando que los paquetes adicionales están en el buffer de recepción y están esperando para ser procesados.

3.5. MODO DE CONFIGURACION Y NEGOCIACION DUPLEX

El ENC28J60 será detectado como dispositivo Half-Duplex. Para comunicarse en modo Full-Duplex, el ENC28J60 se debe configurar manualmente.

3.5.1 Half-Duplex

El ENC28J60 funciona en modo Half-Duplex cuando MACON3.FULDPX=0 y PHCON1.PDPXMD = 0.

Si solamente uno de estos dos bits se fija, el ENC28J60 estará en un estado indeterminado y no funcionará correctamente. Puesto que tendría que cambiar los modos Full y Half-Duplex que pueden dar lugar a este estado indeterminado, el controlador del host (equipo anfitrión) no debe transmitir ningún paquete (mantiene ECON1.TXRTS vacío) y la recepción del paquete debe ser terminada (ECON1.RXEN y ESTAT.RXBUSY deben estar vacíos) durante este período.

En el modo Half-Duplex, solamente un controlador de Ethernet puede transmitir en el medio físico en cualquier momento. Si el controlador del host (equipo anfitrión) es el bit de ECON1.TXRTS, si solicita que un paquete sea transmitido mientras que otro controlador de Ethernet está transmitiendo, el ENC28J60 se retrasa, después de que la transmisión pare, el ENC28J60 procurará transmitir su paquete. Si otro controlador de Ethernet comienza a transmitir en aproximadamente el mismo tiempo que el ENC28J60 comienza, los datos llegarán a ser erróneos y ocurrirá una colisión. El hardware manejará esta condición de dos maneras:

- Si la colisión ocurre antes del número de los bytes especificados. El MACLCON2 será transmitido, el bit de ECON1.TXRTS seguirá siendo parte del sistema y transcurrirá como definido por la especificación de IEEE 802.3 (Ethernet) entonces al transmitir el paquete el controlador del host (equipo anfitrión) no necesitará intervenir, podrá determinar si el paquete ha sido transmitido con éxito leyendo la ESTAT.TXABRT.

- Si la colisión ocurre después del número de los bytes especificados. El MACLCON2 será transmitido y el paquete será abortado inmediatamente sin ninguna tentativa de retransmisión. Ordinariamente, en IEEE 802.3 (Ethernet) se encontraran redes que se configuran correctamente, en esta última no ocurrirán colisiones. La intervención del usuario se puede requerir para corregir la edición. Este problema puede ocurrir como resultado de un nodo Full-Duplex que procura transmitir en el modo Half-Duplex.

Alternativamente, el ENC28J60 puede funcionar en modo Half-Duplex mientras que pueda ser conectado con una red Full-Duplex. El tamaño excesivamente largo del cable y de la red puede ser también una causa posible de colisiones.

3.5.2 Operación Full-Duplex

El ENC28J60 funciona en modo Full-Duplex cuando MACON3.FULDPX = 1 y PHCON1.PDPXMD = 1.

Half-Duplex puede dar lugar a este estado indeterminado, el controlador del host (equipo anfitrión) no debe transmitir ningún paquete (mantiene ECON1.TXRTS claramente) y la recepción del paquete debe ser terminada (ECON1.RXEN y ESTAT.RXBUSY deben estar claros) durante este período.

En modo Full-Duplex, los paquetes serán transmitidos simultáneamente mientras que son recibidos. Dado esto, es imposible causar cualquier colisión al transmitir los paquetes.

EFLOCON: ETHERNET REGISTRO DE CONTROL DE FLUJO

U-0	U-0	U-0	U-0	U-0	R-0	R/W-0	R/W-0
—	—	—	—	—	FULDPXS	FCEN1	FCEN0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Bit 7-3: Leer como '0'
Bit 2 FULDPXS: Read-Sólo MAC Full-Duplex Shadow bit
1 = MAC está configurado para el modo Full-Duplex, FULDPX (MACON3 <0>) se fija 0 = MAC está configurado para el modo Half-Duplex, FULDPX (MACON3 <0>) es clara

Bit 1-0 FCEN1: FCEN0: Habilitar los bits de control de flujo

Cuando FULDPXS = 1:

11 = Enviar una pausa marco con un '0' temporizador de valor y, a continuación, gire el control de flujo libre
10 = Enviar periódicamente los marcos de pausa
01 = Enviar un marco pausa luego gire el control de flujo libre
00 = Control de flujo libre

Cuando FULDPXS = 0:

11 = Control de flujo en
10 = Control de flujo libre

4. NIVELES DE PROGRAMACION

Microchip da acceso a unas librerías para TCP/IP Stack con las siguientes funciones:

- SPI_Ethernet_Init
- SPI_Ethernet_doPacket
- SPI_Ethernet_putByte
- SPI_Ethernet_getByte
- SPI_Ethernet_UserTCP
- SPI_Ethernet_UserUDP

La función Init inicializa el controlador, el módulo MAC y otras propiedades.

La función doPacket escucha y gestiona la recepción de paquetes de Ethernet. Si recibe un ARP (Protocolo de Resolución de Direcciones) o PING (Comprueba la conectividad de nivel IP en otro equipo TCP/IP al enviar mensajes de solicitud de eco de ICMP (Protocolo de mensajes de control Internet) responde automáticamente, y si se trata de tramas TCP (Protocolo de Control de Transmisión) o UDP (Protocolo de datagramas de usuario) las pasa al siguiente nivel.

Las funciones putByte y getByte leen o almacenan un byte en la dirección del ENC28J60 donde esté apuntando el puntero.

Las funciones UserTCP y UserUDP son llamadas internamente por la librería y devuelven la longitud de los datos a transmitir.

Con la librería de Microchip se podrá montar de forma sencilla un micro conectado a Ethernet, que escuchara el canal y contestara a solicitudes ARP (Protocolo de Resolución de Direcciones) y PING (Comprueba la conectividad de nivel IP en otro equipo TCP/IP al enviar mensajes de solicitud de eco de ICMP (Protocolo de mensajes de control Internet) con el siguiente código:

```
program enc_ethernet
include enc_utils ' aquí es donde se debe escribir la implementación
'para UDP y HTTP
'*********************************
'* variables RAM
'*
dim myMacAddr as byte[6] ' my MAC address
myIpAddr as byte[4] ' my IP address
main:
ADPCFG = 0xFBFF ' usamos conversores ADC (canal10)
PORTB = 0
TRISB = 0xFFFF ' set PORTB como entrada de botones
PORTD = 0x00
TRISD = 0 ' set PORTD como salida
httpCounter = 0
myMacAddr[0] = 0x00myMacAddr[1] = 0x14
myMacAddr[2] = 0xA5
myMacAddr[3] = 0x76
myMacAddr[4] = 0x19
myMacAddr[5] = 0x3F
myIpAddr[0] = 192
myIpAddr[1] = 168
myIpAddr[2] = 20
myIpAddr[3] = 60
'*
'* comienza ENC28J60 con :
'* reset bit en RF0
'* CS bit en RF1
'* nuestra dirección MAC e IP
'* full duplex
'*
' para una comunicación SPI más rápida usar Spi_Init_Advanced routine
Spi2_Init()
Spi_Ethernet_Init(PORTG,    13,    PORTG,    12,    myMacAddr,    myIpAddr,
SPI_Ethernet_FULLDUPLEX)
while true ' do forever
Spi_Ethernet_doPacket() ' rutina para la llegada de paquetes
'*
'* aquí podemos añadir más cosas
'* ENC28J60_doPacket() debe llamarse tanto como se pueda
'* si no se podrían perder paquetes
'*
wend
end.
```

Pero como el interés es desarrollar un stack personalizado con fines didácticos, explorando las posibilidades del sistema y comprendiendo su funcionamiento, se hablara de los pedazos de código más importantes:

4.1 Interfaz SPI

La comunicación entre el controlador ENC28J60 y el PIC se realiza por medio de la interfaz SPI. Ésta solo soporta el modo “0,0” y el controlador en modo esclavo, por lo tanto es el PIC el que suministra la señal de reloj y maneja la transmisión. La máxima frecuencia admitida es de 10Mhz para las Rev. B1-B4, y el doble para la B5. Además, a causa de un problema en la interfaz para las Rev. B1-B4 el reloj tiene que estar necesariamente entre los 8 y los 10Mhz, o el reloj del PIC tiene que sacarse del pin CLKOUT del controlador (máx. 25Mhz). En el primer caso, entonces, el PIC tiene que trabajar a una frecuencia entre los 32 y los 40Mhz.

4.1.1 Instrucciones SPI

Por medio de la interfaz es posible enviar al chip 7 instrucciones diferentes (de 8 BIT), seguidos por un byte de datos. Estas son:

Nombre	1er byte	2° byte	Descripción
Read Control Register (RCR)	000 AAAAA		sirve para leer el registro de control A
Write Control Register (WCR)	010 AAAAA	DDDDDDDD	para escribir el byte D en el registro A (en el banco seleccionado)
Read Buffer Memory (RBM)	00111010		sirve para leer la memoria RAM del controlador a la dirección actual
Write Buffer Memory (WBM)	01111010	DDDDDDDD	escribe el byte D en la dirección actual de la memoria RAM
Bit Field Set (BFS)	100 AAAAA	DDDDDDDD	SET en el registro A (solo ETH), los BIT que hay en D están a 1 (OR)
Bit Field Set (BFS)	101 AAAAA	DDDDDDDD	resetea el registro A (solo ETH), los BIT que hay en D están a 1 (NOT AND)
System Reset Command (SRC)	11111111		resetea el controlador

4.1.2 Instrucciones SPI: implementación

La inicialización del módulo SPI será ilustrada más adelante, por el momento se nombraran algunas instrucciones básicas:

```
#define spiWrite(x)    spiRW(x)
#define spiRead()      spiRW(0)
 ....
u8 spiRW(u8 data){
        SSPBUF = data;
        while(!PIR1bits.SSPIF);
        PIR1bits.SSPIF = 0;
        return SSPBUF;
}
```

Ésta es el instrucción que permite leer/escribir en el bus SPI. Por lo que concierne a la escritura, el byte que se tiene que enviar se pone en el registro SSPBUF (forma parte del módulo SPI), después espera a que la transmisión termine observando el BIT SSPIF. Para leer un byte, en cambio, es necesario escribir un cero en el registro SSPBUF (lo que provoca la generación de 8 impulsos de reloj), se espera el fin de la operación y el byte leído se encontrará en el mismo registro SSPBUF.

En este caso se ha condensado en un único método las operaciones de lectura y escritura. Después, para una mayor claridad, se ha definido definido spiWrite y spiRead.

```
#define CS     PORTCbits.RC2          // Chip Select del ENC28J60

#define WCR    (0b01000000)           // Write Control Register
                                      // command
#define BFS    (0b10000000)           // Bit Field Set command
#define BFC    (0b10100000)           // Bit Field Clear command
#define RCR    (0b00000000)           // Read Control Register command
#define RBM    (0b00111010)           // Read Buffer Memory command
#define WBM    (0b01111010)           // Write Buffer Memory command
#define SRC    (0b11111111)           // System Reset command
 ....
void writeReg(u8 reg, u8 data){
        CS = 0;
        spiWrite(WCR | reg);
        spiWrite(data);
```

```
        CS = 1;
}

u8 readMAC(u8 reg){
        u8 b;
        CS = 0;
        spiWrite(RCR | reg);
        spiRead();
        b = spiRead();
        CS = 1;
        return b;
}

u8 readETH(u8 reg){
        u8 b;
        CS = 0;
        spiWrite(RCR | reg);
        b = spiRead();
        CS = 1;
        return b;
}
```

Con estas tres instrucciones se puede leer y escribir los registros de control (para leer los registros MII se usa readMAC).

Como se puede observar, el proceso de lectura de un registro del módulo MAC y ETH es un poco diferente, para el primero de ellos se tiene que enviar un BIT cero antes de efectuar la lectura.

```
void BFCReg(u8 reg, u8 data){
        CS = 0;
        spiWrite(BFC | reg);
        spiWrite(data);
        CS = 1;
}

void BFSReg(u8 reg, u8 data){
        CS = 0;
        spiWrite(BFS | reg);
        spiWrite(data);
        CS = 1;
}

void setBank(u8 bank){
        BFCReg(ECON1, 0b11);
        BFSReg(ECON1, bank);
}
```

Aquí se puede ver la implementación de las instrucciones Bit Set y Bit Clear; después la instrucción setBank, con la cual se logra seleccionar el banco de memoria de los registros de control, actuando sobre el registro ECON1 (común a todos los bancos).

```
u16  bufSize;
  ....
void encPut(u8 b){
        CS = 0;
        spiWrite(WBM);
        spiWrite(b);
        CS = 1;
        bufSize++;
}

u8 encGet(){
        u8 b;
        CS = 0;
        spiWrite(RBM);
        b = spiRead();
        CS = 1;
        return b;
}

void sendReset(){
        CS = 0;
        spiWrite(SRC);
        CS = 1;
}
```

Las tres instrucciones que quedan son realizadas por estas tres funciones: lectura y escritura de la RAM, y reset. La variable estática bufSize es útil para mantener una traza del número de byte de la RAM (servirá más tarde).

4.2 Registros PHY

Los registros PHY (Capa física) tienen 16 bit y son accesibles a través de los registros MII (Interfaz de medios independientes).

Para leer un registro PHY (Capa física):

- Se introduce la dirección en el registro MIREGADR
- SET del BIT MIIRD (1) del registro MICMD, empieza la lectura

- Se espera el término de la lectura observando el BIT BUSY (1) del registro MISTAT
- Se resetea el BIT MIIRD
- El dato se grabará en los registros MIRDL y MIRDH

Para escribir en un registro PHY (Capa física):

- Se introduce la dirección del registro en MIREGADR;
- Primero se escribe el byte menos significativo en MIWRL, después escribiendo el byte más significativo en MIWRH comienza la escritura.
- Se espera a que el módulo PHY (Capa Física) termine la operación.

```
u16 readPHY(u8 reg){
        setBank(2);
        writeReg(MIREGADR, reg);
        writeReg(MICMD, 0x01);

        setBank(3);
        while(readMAC(MISTAT) & 1);

        setBank(2);
        writeReg(MICMD, 0x00);

        return readMAC(MIRDL) | (readMAC(MIRDH) << 8 );
}
void writePHY(u8 reg, u16 data){
        setBank(2);
        writeReg(MIREGADR, reg);
        writeReg(MIWRL, LOW(data));
        writeReg(MIWRH, HIGH(data));

        setBank(3);
        while (readMAC(MISTAT) & 1);
}
```

4.2.1 Otros métodos

Los siguientes programas sirven para escribir/leer más byte de la RAM (Memoria de Acceso Directo) del controlador; la dirección de lectura está situada en los registros ERDPT, y la dirección de escritura está situada en los registros EWRPT (son auto-incrementales).

```
void encGetArray(u8* buf, u16 len){
        CS = 0;
        spiWrite(RBM);
        while(len--)
                *buf++ = spiRead();
        CS = 1;
}

void encPutArray(u8* buf,u16 len){
        bufSize += len;
        CS = 0;
        spiWrite(WBM);
        while(len--)
                spiWrite(*buf++);
        CS = 1;
}

void encPutString(const rom u8 *str){
        CS = 0;
        spiWrite(WBM);
        while(*str) {
                spiWrite(*str++);
                bufSize++;
        }
        CS = 1;
}
```

4.3 Inicialización

La inicialización del módulo SPI se realiza simplemente con dos instrucciones:

```
 void encInit(){
        TRISB = 0xFF;           // configuración I/O del PORTB
        TRISC = 0xD1;           // configuración I/O del PORTC
        PORTC = 0x00;

        SSPSTAT = 0x40;
```

```
        SSPCON1 = 0x20;
```

En particular el módulo MSSP viene habilitado y configurado en el modo "0,0", con reloj igual a Fosc/4.

4.3.1 Inicialización ENC28J60

La inicialización del controlador necesita la configuración de ciertos registros, y la habilitación de la recepción.

```
#define RX_BUF_START   0
#define RX_BUF_END     6499
#define TX_BUF_START   6500
 ....
setBank(0);
writeReg(ERXSTL,   LOW(RX_BUF_START));        //
writeReg(ERXSTH,   HIGH(RX_BUF_START));       // inicio buffer de
                                              // lectura
writeReg(ERXRDPTL, LOW(RX_BUF_END));          //
writeReg(ERXRDPTH, HIGH(RX_BUF_END));         // puntero del buffer de
                                              // lectura
writeReg(ERXNDL,   LOW(RX_BUF_END));          //
writeReg(ERXNDH,   HIGH(RX_BUF_END));         // fin del buffer de
                                              // lectura
writeReg(ETXSTL,   LOW(TX_BUF_START));        //
writeReg(ETXSTH,   HIGH(TX_BUF_START));       // inicio buffer de
                                              // escritura
```

Como ya se ha mencionado, el buffer del ENC28J60 puede dividirse como se prefiera entre la memoria de transmisión y la de recepción. Para hacer esto se configuran los punteros del buffer de recepción, y la memoria que queda será el buffer de transmisión.

Los registros ERXST contienen la dirección del primer byte del buffer de recepción, mientras que los registros ERXND almacenan el último byte.

```
setBank(2);
writeReg(MACON1, 0b01101);              // MARXEN,  TXPAUS,  RXPAUS
writeReg(MACON3, 0b00110000);           // Half Duplex, Padding 60byte,
                                        // CRC
writeReg(MAIPGL, 0x12);                 //
writeReg(MAIPGH, 0x0C);                 //
writeReg(MABBIPG,0x12);                 // Espacio entre paquetes
```

Estos registros configuran el módulo MAC. A través del registro MACON1 se habilita el módulo MAC y la recepción/transmisión.

Con el registro MACON3 se elige la modalidad Duplex (Half o Full) del módulo MAC que tiene que ser programada de igual manera también en el módulo PHY (Capa física); además en este registro están presentes algunas configuraciones sobre el padding (relleno) automático y el cálculo del CRC (Control de errores).

Los registros MAIPG y MABBIPG contienen los valores de las pausas entre los paquetes, los presentes en el código son los valores estándar.

```
writeReg(MAMXFLL, LOW(1500));
writeReg(MAMXFLH, HIGH(1500));
```

En los registros MAXFL (Max Frame Length) se almacena el tamaño máximo de paquete; el controlador puede configurarse de forma que rechace enviar un paquete que supere este límite.

```
#define MY_MAC1     0x00
#define MY_MAC2     0x04
#define MY_MAC3     0xA3
#define MY_MAC4     0x00
#define MY_MAC5     0x30
#define MY_MAC6     0x39
  ....
#define MY_IP1      192
#define MY_IP2      168
#define MY_IP3      20
#define MY_IP4      40
  ....
setBank(3);
writeReg(MAADR1, MY_MAC1);
writeReg(MAADR2, MY_MAC2);
writeReg(MAADR3, MY_MAC3);
writeReg(MAADR4, MY_MAC4);
writeReg(MAADR5, MY_MAC5);
writeReg(MAADR6, MY_MAC6);
```

Como se está realizando un software sencillo, se asignara una IP estática al controlador, asumiendo que no se repetirá en nuestra red. Una mejora para este sistema sería programar también el protocolo DHCP (Protocolo de configuración dinámica de host) para que pudiera solicitar una IP al router.

La dirección MAC (Dirección Física) del dispositivo se guarda en los registros MACADR; éstos se utilizan por el filtro para rechazar paquetes que no vayan destinados al controlador, por lo que la dirección no será insertada automáticamente en los paquetes que se tienen que enviar.

Hay que tener en cuenta a la hora de configurar la dirección MAC (Dirección Física) que los bytes 1, 2, y 3 que corresponden con la lista de identificadores únicos (OUI) que se asignó a Microchip no debe cambiarse. Para los bytes 4, 5 y 6, el ENC28J60 viene con una pegatina que indica una dirección única asignada por fábrica. Se tendrá que convertir esta dirección a hexadecimal y utilizar este número para los últimos tres bytes de la dirección MAC (Dirección Física). Direcciones menores de 65535 o 0xFFFF sólo utilizarán los bytes 5 y 6 de la dirección MAC (Dirección Física), utilizando un byte 0x00 para la posición 4 en este caso. Por ejemplo, si el número es "12345", la dirección MAC (Dirección Física) sería 00-04-A3-00-30-39.Si esta pegatina o la dirección no puede ser localizada, se elegirá un número que pueda ser único en su red.

```
writePHY(PHCON2, 0b0000000100000000); // inhabilita el loopback
writePHY(PHCON1, 0);                  // habilita el PHY

setBank(1);
writeReg(ERXFCON, 0b10100001);        // programa los filtros de
                                      // recepción
BFSReg(ECON1, 0b100);                 // habilita la recepción
```

Con esta última parte termina la inicialización del controlador. El módulo PHY no tiene muchas opciones para configurar, las únicas opciones para configurar son la inhabilitación del LoopBack (empleado para hacer test) y la habilitación del módulo.
Los filtros están programados para aceptar sólo paquetes destinados a la dirección MAC (Dirección Física) configurada y paquetes de Broadcast.

4.4 EJEMPLO: LEER EL RevID

Llegado a este punto se tiene todo el código necesario para intercambiar datos con el controlador, se puede leer el registro EREVID (sólo como prueba) situado en el banco 3, el cual contiene el numero de revisión del Chip.

En el ejemplo, el valor del registro se ha introducido en un PORTB, por lo tanto, respecto al esquema, el pin RB0 tendría que quedarse desconectado del INT, y TRISB programado en cero.

Como alternativa, dejando **int** conectado, se puede desplazar PORTB a la izquierda.

```
void main () {
        encInit ();

        setBank (3);
        PORTB = readETH (EREVID);

        while (1);
}
```

4.5 Transmisión

El ENC28J60, en el módulo MAC, se ocupa de generar los campos Preamble, SFD, el posible padding (relleno para cumplir el tamaño de trama mínimo que exige el protocolo) y el FCS; el resto debe ser insertado por el software en el buffer de trasmisión. Además, el byte que se encuentra en la primera posición de memoria es un byte de control y no se envía realmente. Utilizando cero como valor de este byte, vienen usadas, para las transmisiones, las opciones ya programadas en MACON3.
Se ha definido el buffer de transmisión de manera que empiece en la dirección TX_BUF_START, por lo tanto las operaciones que hay que realizar para la preparación de la cabecera del módulo MAC son:

- Se programan los registros EWRPT de modo que apunten al principio del buffer transmisión.

- Enviando el comando WBM se puede empezar a escribir los datos en el buffer (por medio de spiWrite).

- Se escribe el byte de control (por ejemplo cero).
- Siguen la dirección MAC (Dirección Física) del destinatario y el remitente, finalmente el campo Type/Length (Tipo/Longitud).
- La función que desarrolla esta operación es *MACPutHeader*:

```
void MACPutHeader(MACAddr target, u16 type){
        u8 i;
        bufSize = sizeof(MAC_Header);
        setBank(0);
        writeReg(EWRPTL, LOW(TX_BUF_START));
        writeReg(EWRPTH, HIGH(TX_BUF_START));
        CS = 0;
        spiWrite(WBM);
        spiWrite(0x00);// usa MACON3

        for (i=0;i<6;i++)
                spiWrite(target.b[i]);
        spiWrite(MY_MAC1);
        spiWrite(MY_MAC2);
        spiWrite(MY_MAC3);
        spiWrite(MY_MAC4);
        spiWrite(MY_MAC5);
        spiWrite(MY_MAC6);

        spiWrite(HIGH(type));
        spiWrite(LOW(type));
        CS = 1;
}
```

Las estructuras MAC_Header y MACAddr son definidas en el archivo MAC.h:

```
#define TYPE_ARP    0x0806
#define TYPE_IP     0x0800


typedef struct _MAC_Addr {
  u8    b[6];
} MACAddr;

typedef struct _MAC_Header {
  MACAddr       destMAC;
  MACAddr       sourceMAC;
  u16           type;
} MAC_Header;
```

La cabecera del módulo MAC está lista; ahora se pueden enviar al controlador los datos del nivel superior.

Después de esta última operación, el paquete está listo para ser enviado; a este resultado se llega con pocas instrucciones:

- Se espera a que el controlador esté listo para transmitir observando el BIT TXRTS (3) del registro ECON1.

- Los registros ETXND se cargan con la dirección del último byte que se tiene que enviar. Esta dirección será TX_BUF_START + bufSize.

- Haciendo un SET del BIT TXRTS empieza la transmisión.

A causa de un problema descrito en el Errata, si se verifican errores (bit EIR.TXERIF), es necesario reposicionar la lógica de transmisión, con el BIT TXRST (7) de ECON1.

```
void MACSend(){
        setBank(0);
        if (readETH(EIR) & 0b10) {            // si se verifica un
                                              // error
                BFSReg(ECON1, 0b10000000);    //
                BFCReg(ECON1, 0b10000000);    // reposiciona el TX
        }
        while(readETH(ECON1) & 0b1000);       // espera que esté
                                              // listo para envìar
        writeReg(ETXNDL, LOW(TX_BUF_START + bufSize));
        writeReg(ETXNDH, HIGH(TX_BUF_START + bufSize));
        BFSReg(ECON1, 0b1000);                // envía
```

4.6 Recepción

Antes de todo, para comprobar si se ha recibido un paquete, se controla que el registro EPKTCNT sea diferente de cero; por eso este registro mantiene la cuenta de los paquetes recibidos, y, una vez generado el paquete, tiene que ser decrementado.

La recepción que ha tenido lugar puede verse a través del bit EIR.RXIF, pero según la Errata, el valor de este bit no es fiable. Los datos recibidos se escriben en el buffer de recepción empezando por la dirección ERXWRPT, que en fase de inicialización está automáticamente en cero.

En esta dirección los primeros seis byte no pertenecen al paquete, sino que son bytes de control: los primeros dos contienen la dirección donde se guardará el próximo paquete; los otros no son muy importantes (para profundizar, consultar el datasheet).

Definamos dos nuevas variables, la primera va inicializada a cero en *encInit*:

```
u16  RdPt;
u16  packetStart;
```

RdPt contiene la dirección del próximo paquete, mientras que packetStart almacena la dirección del paquete actual.

Dicho esto veamos el método *MACGetHeader*:

```
void MACGetHeader(MAC_Header* header){
        u8 buf[6];

        packetStart = RdPt;            // guarda RdPt en packetStart
        setBank(0);
        writeReg(ERDPTL, LOW(RdPt));
        writeReg(ERDPTH, HIGH(RdPt));  // ERDPT = RdPt
        encGetArray(&buf[0], 6);       // lee los 6 byte de control
        RdPt = (u16)((u16)buf[0] | ((u16)buf[1] << 8 )); // puntero al
                                                         // Próximo paquete

        encGetArray((u8*)header, sizeof(MAC_Header));         // lee la
                                                 // cabecera MAC

        header->type = htons(header->type);    // swap del campo type
}
```

Como se puede observar, lo primero que se carga en ERDPT es la dirección del paquete que se tiene que leer (guardado en RdPt, inicialmente cero), después se leen los bytes de control y se guarda el nuevo RdPt; finalmente se lee el módulo MAC y se guarda en la cabecera. La instrucción htons está definida en el archivo utility.c; el motivo de su utilización se aclara en el siguiente punto.

4.7 BIG & LITTLE ENDIAN

En los ordenadores clásicos de procesador x86 y en los microcontroladores, las palabras de memoria mínimas son de 8 bits; pero cuando se tienen que representar valores de 16 bits, hay que decidir como vienen representados en memoria.

Un dato de 16 bits claramente ocupa dos palabras de memoria, por lo tanto hay dos posibilidades:

- El byte menos significativo se encuentra en la dirección más baja respecto al MSB. Viceversa, el byte menos significativo se encuentra en la dirección de memoria más alta.

Estas dos opciones son llamadas *Little Endian* y *Big Endian* respectivamente. Los ordenadores con procesadores x86 y también los PIC (Comunicadores personales de Internet), utilizan la primera notación, mientras que en los protocolos de red está difundido el uso de la segunda.

Lo que tiene que hacer la función htons es convertir un dato de 16 bits entre las dos notaciones. En el programa se empleará frecuentemente y hay que poner mucho cuidado en su utilización.

```
u16 htons(u16 val){
        return (((u16)val >> 8 ) | ((u16)val << 8 ));
}
```

Como ya se ha visto el registro ERXWRPT contiene la dirección donde el controlador está escribiendo los datos recibidos, mientras que el registro ERXRDPT apunta a la dirección donde nuestro software está leyendo los datos recibidos; la zona que se encuentra entre estas dos direcciones no está habilitada para la escritura, para impedir la pérdida de datos.

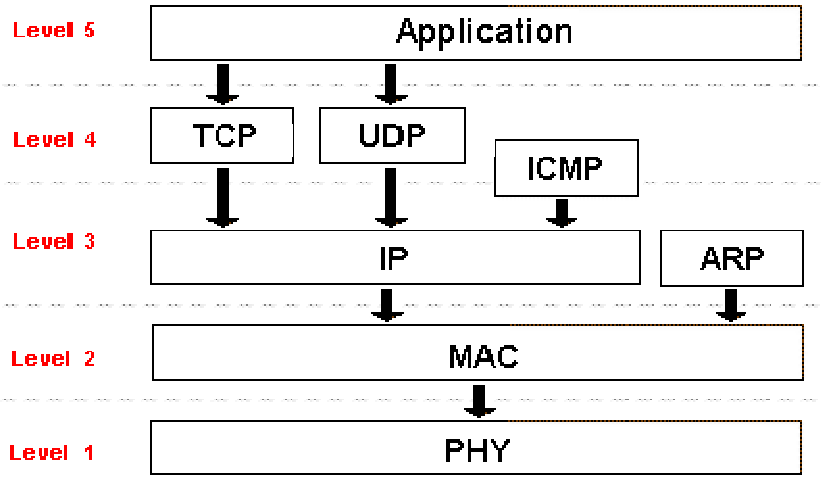
Dicho esto, es necesario, terminar de examinar el paquete, la actualización del registro ERXRDPT con la dirección del próximo paquete que se tiene que leer (RdPt).

```
Void freeRxSpace(){
        u16 NewRdPt;
        setBank(0);
        NewRdPt = RdPt -1;
        if ((NewRdPt > RX_BUF_END) || (NewRdPt < RX_BUF_START)) NewRdPt =
RX_BUF_END;
        BFSReg(ECON2, 0b01000000);                //decrementa EPKTCNT
        writeReg(ERXRDPTL, LOW(NewRdPt));
        writeReg(ERXRDPTH, HIGH(NewRdPt));        // Libera el espacio del
                                                  // Buffer
}
```

En este método viene actualizado el registro y decrementado EPKTCNT. A causa de un problema descrito en la Errata, el valor de ERXRDPT tiene que ser impar; pero sabemos que RdPt es par (ya que el controlador añade un padding (relleno) de manera que sea par). Por lo tanto es suficiente con restar uno a este valor, y después controlar que no salga fuera de los límites del buffer (RX_BUF_START y RX_BUF_END).

4.8 TCP/IP (Protocolo de Control de Transporte/Protocolo Internet)

TCP/IP es una familia de protocolos (Internet Protocol Suite) empleada en el tráfico de datos a través de Internet (y también Intranet). Ésta comprende los protocolos TCP (nivel 4) e IP (nivel 3) que son los más representativos y además los primeros a ser ideados, otros protocolos son el UDP (Protocolo de datagramas de usuario), ICMP (Protocolo de mensajes de control Internet), ARP (Protocolo de Resolución de Direcciones), RARP (Protocolo de resolución de direcciones inverso), etc.
Estos protocolos ocupan los niveles 3 y 4 del modelo TCP/IP, por lo tanto principalmente se ocupan del transporte, más o menos fiable y de los datos generados por las Aplicaciones.



En las siguientes páginas se prestara más atención a los siguientes protocolos:

- **ARP (Protocolo de Resolución de Direcciones):** Permite la conversión de una dirección IP con la correspondiente dirección MAC (Dirección Física).

- **IP (Protocolo de Internet)**: Es el protocolo de nivel 3 donde se basa principalmente la Red; es indispensable para la correcta dirección de los paquetes, por medio de la dirección IP del destinatario.

- **ICMP (Protocolo de mensajes de control Internet):** Se encuentra oficialmente en el nivel 3, pero está arriba del IP; desarrolla funciones para dirigir y diagnosticar (veremos el ping).

- **TCP (Protocolo de Control de Transmisión)**: Es un protocolo de transporte muy fiable garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron (También complejo).

- **UDP (Protocolo de datagramas de usuario):** Es otro protocolo de transporte, pero menos fiable que el TCP (Protocolo de Control de Transmisión), tiene la ventaja de ser de fácil implementación y generalmente ofrece prestaciones mejores.

4.9 ARP (Protocolo de Resolución de Direcciones)

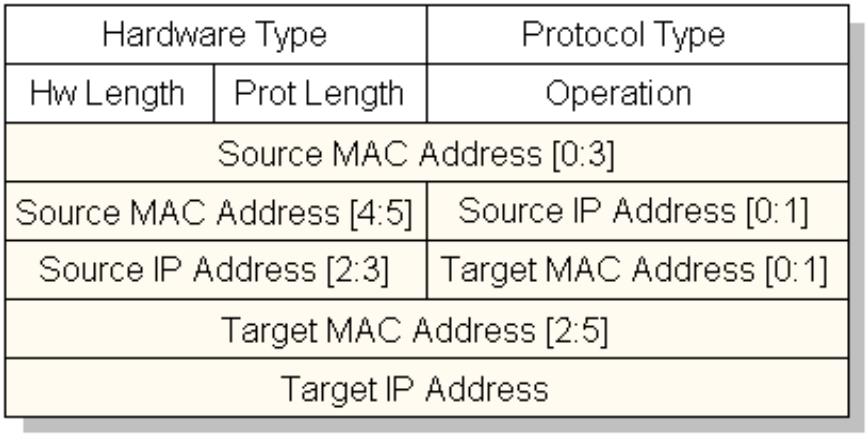
El ARP (Address Resolution Protocol) es un protocolo de tercer nivel del modelo TCP/IP, empleado dentro de redes LAN (Red de área local), para traducir direcciones IP a sus correspondientes direcciones MAC (Dirección Física).

4.9.1 El encabezado

Este protocolo no fue ideado sólo para redes Ethernet y protocolo IP, por lo tanto un paquete ARP contiene varios campos que especifican las direcciones que se están comunicando:

- ***Hardware Type*:** Indica el tipo de dirección física (para Ethernet vale 1).
- ***Protocol Type*:** Contiene el tipo de dirección de protocolo (para IP vale 0x800).
- ***Hardware Address Length*:** La longitud de la dirección física (para Ethernet es 6).
- ***Protocol Address Length*:** La longitud de la dirección de protocolo (para IPv4 4).
- ***Operation*:** Representa el mando; en nuestro caso veremos ARP Request y ARP Replay.

A esto le sigue un campo de datos de longitud variable (que depende de la cabecera), que para las operaciones nombradas tendrá 4 direcciones, como indica la siguiente ilustración:



4.9.2 Request

El paquete que pide la dirección MAC (Dirección Física) contiene, la cabecera, las direcciones IP y la dirección MAC (Dirección Física) del remitente, y como *Target IP Address*, la dirección de la cual se quiere conocer la dirección MAC (Dirección Física).

Por lo tanto, cuando llega un paquete ARP es suficiente controlar los campos *Operation* y justamente *Target IP Address* que tiene que corresponder a nuestra dirección IP, de no ser así ignoramos la petición.

4.9.3 ARP Reply

El paquete de respuesta es muy parecido al que acabamos de ver: es suficiente indicar en el campo *Operation* que se trata de una respuesta, y llenar los demás campos con las direcciones del remitente y del destinatario.

Todo esto está recogido en una única, breve, función:

```
void processARP(){
        ARPPacket packet;
        IPAddr tmp;

        encGetArray((u8*)&packet, sizeof(packet));
        packet.operation = htons(packet.operation);

        if (packet.operation == ARP_REQUEST){
                if (ipMatch(packet.TargetIP,MyIP)){
                        packet.operation = htons(ARP_REPLY);
                        tmp = packet.TargetIP;
                        packet.TargetMAC = packet.SourceMAC;
                        packet.TargetIP = packet.SourceIP;
                        packet.SourceIP = tmp;
                        packet.SourceMAC.b[0] = MY_MAC1;
                        packet.SourceMAC.b[1] = MY_MAC2;
                        packet.SourceMAC.b[2] = MY_MAC3;
                        packet.SourceMAC.b[3] = MY_MAC4;
                        packet.SourceMAC.b[4] = MY_MAC5;
                        packet.SourceMAC.b[5] = MY_MAC6;
                        MACPutHeader(packet.TargetMAC, TYPE_ARP);
                        encPutArray((u8*)&packet,sizeof(packet));
                        MACSend();
                }
        }
}
```

El paquete ARP (Protocolo de Resolución de Direcciones) está definido así:

```
typedef struct {
  u16           hardware;
  u16           protocol;
  u8            MacLen;
  u8            ProtocolLen;
  u16           operation;
  MACAddr       SourceMAC;
  IPAddr        SourceIP;
  MACAddr       TargetMAC;
  IPAddr        TargetIP;
} ARPPacket;
```

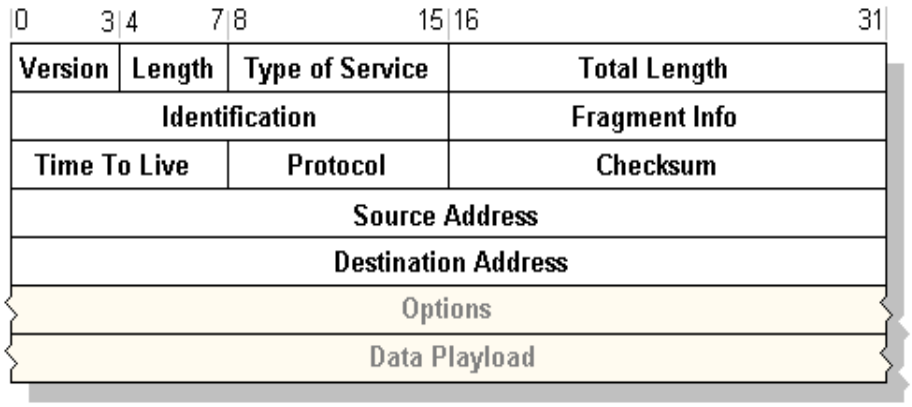
4.10 PROTOCOLO IP (PROTOCOLO DE INTERNET)

El protocolo IP es la base de la comunicación a través de Internet, precisamente por ello es la abreviatura de Protocolo de Internet, y su papel es el de permitir el

enrutamiento de los paquetes a través de la red, para que puedan llegar a su destinatario.

El elemento fundamental es la dirección IP, a través de la cual se puede identificar al remitente y destinatario de manera singular dentro de una misma red. ¡Atención! ¡No a nivel global, como ocurre con la dirección MAC! Solo dentro de una red privada, la dirección IP puede ser elegida a voluntad.

4.10.1 Cabecera IP



Como se puede observar en la figura la cabecera IP (IP Header) contiene la dirección IP del remitente y el destinatario. Además, tiene otro tipo de información, más o menos importante, veamos alguna:

- **Versión:** La versión del protocolo IP en uso, sólo usaremos la versión 4.
- **Longitud de cabecera:** Longitud cabecera en palabras de 4 bytes.
- **Longitud total:** Longitud total del paquete IP (cabecera + datos).
- **Identificación:** Número de secuencia utilizado para identificar de forma exclusiva el paquete IP durante una comunicación.

- **Time to Live (TTL):** contiene el número de saltos (a través de routers) antes de que el paquete se elimine (devolviendo error de destino inalcanzable).

- **Protocolo:** Indica el tipo de protocolo de cuarto nivel de contenido en el campo de datos (TCP **(**Protocolo de Control de Transmisión), UDP (Protocolo de datagramas de usuario)…)

- **Checksum:** Campo para controlar la integridad de la cabecera únicamente, no la de datos.

Hay otras áreas, pero no las utilizaremos. Entonces la programación en C sería de la siguiente manera:

```
typedef struct {
   u8    b[4];
} IPAddr;

typedef struct {
        u8      verlen;
        u8      typeOfService;
        u16     totalLength;
        u16     id;
        u16     fragmentInfo;
        u8      TTL;
        u8      protocol;
        u16     checksum;
        IPAddr  sourceIP;
        IPAddr  destIP;
} IP_Header;
```

4.11 IP RECEPCIÓN Y ENVÍO

4.11.1 Recepción

A la recepción de un paquete IP, las operaciones a llevar a cabo son sencillas, y se implementarán en el programa processIP:

• Control del destinatario.

• Control de errores.

• Pasar al nivel superior.

Hay que asegurarse de que el receptor del paquete es la dirección IP que tenemos asignada, y este control lo llevaremos a cabo comparando la variable con MyIP,

Esta variable la definimos en el archivo ip.c y se inicializa en la instrucción encInit:

```
// en ip.c
IPAddr MyIP;

// in define.h

#define MY_IP1      192
#define MY_IP2      168
#define MY_IP3      1
#define MY_IP4      2

// en enc28j60.c, encInit()
MyIP.b[0] = MY_IP1;
MyIP.b[1] = MY_IP2;
MyIP.b[2] = MY_IP3;
MyIP.b[3] = MY_IP4;
```

El control de integridad de la cabecera de la dirección IP se controla con la verificación de Checksum (Control de redundancia) de dicho campo, esta operación se realiza con la función DMAChecksum que veremos en otro punto. Ahora veremos la función processIP:

```
void processIP(){
        IP_Header header;
        u16 chksum;
        u8 optlen;

        encGetArray((u8*)&header, sizeof(header));

        if (!ipMatch(header.destIP,MyIP)) return;

        // controla el checksum
        if (DMAChecksum(0, (header.verlen & 0x0F)*4, FALSE)) return;

        // descarta eventualmente el campo opciones
```

```
        optlen = (header.verlen & 0x0F)*4 - 20;
        if (optlen)
                encDiscard(optlen);

        // big endian --> little endian
        swapIPHeader(&header);

        switch (header.protocol){
                case IPPROTO_ICMP: processICMP(header);
                        break;
        /*      case IPPROTO_UDP :
                        break;
                case IPPROTO_TCP:
                        break;  */
                default: break;
        }
}
```

La función IpMatch compara dos direcciones IP:

```
u8 ipMatch(IPAddr ip1, IPAddr ip2){
        return (ip1.b[0] == ip2.b[0] && ip1.b[1] == ip2.b[1] && ip1.b[2]
== ip2.b[2] && ip1.b[3] == ip2.b[3]);
}
```

4.11.2 Envíos

Para enviar un paquete IP se necesita generar la cabecera de la dirección MAC (Dirección Física) y la cabecera de la dirección IP, a lo que le seguirá el campo datos con el paquete de cuarto nivel.

De esto se ocupará la función IPPutHeader a la que se le pasan los siguientes parámetros:

• **Target**: Dirección IP de destino

• **Protocol**: Protocolo de cuarto nivel que hay en el campo de datos.

• **Data**: Puntero a la posición de memoria que contiene los datos que serán incluidos en el paquete.

• **Datalen**: Tamaño del campo data.

• **Total Length**: Tamaño total del paquete, excluyendo la cabecera (no siempre coincide con dataLen)

Y la función sería:

```
void IPPutHeader(IPAddr target, u8 protocol, u8* data, u16 datalen, u16
totalLength){
        IP_Header header;
        u16 tmp;
        header.verlen = 0x45;
        header.typeOfService = 0x00;
        header.totalLength = 20+totalLength;
        header.id = ++id;
        header.fragmentInfo = 0x00;
        header.TTL = 128;
        header.protocol = protocol;
        header.checksum = 0x00;;
        header.sourceIP = MyIP;
        header.destIP = target;

        swapIPHeader(&header);

        MACPutHeader(remoteAddr, TYPE_IP);
        encPutArray((u8*)&header, sizeof(header));
        encPutArray(data, datalen);
        putChecksum(IP_Offset+10,  DMAChecksum(IP_Offset,  sizeof(header),
TRUE));
}
```

Primero se prepara la cabecera en la variable *header*, usando otros parámetros y constantes, después se llama a la función MACPutHeader que escribe en el buffer la cabecera de la dirección MAC (Dirección Física) y prepara la operación de escritura y envío. Seguidamente se envía la cabecera de la dirección IP y los datos al controlador, y se calcula el checksum (Control de redundancia) del paquete, que también se escribe en el buffer (en la posición IPOffset+10).

Para trabajar con el campo de checksum (Control de redundancia) en el paquete, primero debe estar a cero, con el método DMAChecksum se calcula su valor directamente del buffer del ENC28J60.

4.12 CHECKSUM (Control de Redundancia)

El checksum es un método de control de redundancia del paquete, que permite determinar si contiene errores (y por tanto se debe descartar). La palabra Checksum no identifica un algoritmo en particular pero se utiliza un sistema en concreto para IP y otros protocolos de comunicaciones. Se trata de hallar el complemento a 1 de la suma del complemento a 1 de la palabra de 16 bits de los datos que analizamos, y compararlo con el valor que hay en el campo de Checksum (Control de redundancia).

Si los valores son iguales, suponemos que los datos son correctos y que no han sufrido ninguna interferencia. También se pueden sumar todas las palabras de 16 bits, sumando el posible overflow que pudiera surgir y después sacar el complemento a 1. Para esta verificación, debemos guardar temporalmente el campo Checksum (Control de redundancia) y poner en su lugar un cero, después hacemos los cálculos pertinentes y comprobamos el valor obtenido con el que hay guardado.

El ENC28J60 integra, en el módulo DMA (Acceso directo a la memoria), una función bastante rápida para el cálculo del checksum (Control de redundancia) y es la que utilizaremos. Desafortunadamente, para la revisión B5 del chip, se encuentran serios problemas para recibir paquetes, por lo que se deberá realizar a través de software.

Primero, veremos el prototipo de la función DMAChecksum:

```
u16 DMAChecksum(u16 start, u16 len, BOOL rx)
```

El primer parámetro indica la dirección de comienzo de los datos, *Len* es el tamaño en bytes de estos datos, y en recepción se utiliza para indicar si la dirección se refiere a memoria de RX (Recepción) o de TX (Transmisión) (La dirección es un offset relativo, respecto al inicio del campo de datos del paquete MAC).

Como dijimos antes, este método tiene dos implementaciones diferentes para las revisiones B1-B4 y para la revisión B5, veremos el primer caso

```
u16 DMAChecksum(u16 start, u16 len, BOOL rx){
        // calcula el checksum utilizando la función hardware del chip
```

```
        u16 tmp;
        u8 L,H;
        if (rx) {
                tmp = TX_BUF_START + 1 + sizeof(MAC_Header) + start;
        } else {
                tmp = packetStart + 6 + sizeof(MAC_Header) + start;
                if (tmp > RX_BUF_END)
                        tmp = tmp - RX_BUF_END + RX_BUF_START - 1;
}
        setBank(0);
        writeReg(EDMASTL, LOW(tmp));
        writeReg(EDMASTH, HIGH(tmp));

        tmp = tmp+len-1;                        // fin del paquete
        if (!rx && tmp > RX_BUF_END)
                tmp = tmp - RX_BUF_END + RX_BUF_START - 1;
        writeReg(EDMANDL, LOW(tmp));
        writeReg(EDMANDH, HIGH(tmp));

        BFSReg(ECON1, 0b00110000);              // inicio calculo
        while(readETH(ECON1) & 0b00100000);     // espera el fin del
                                                // calculo
        tmp = (u16)readETH(EDMACSL) << 8;
        tmp = tmp | readETH(EDMACSH);
        return tmp;                             // devuelve el checksum
                                                // calculado
}
```

Primero se calcula y se escribe en los registros apropiados las direcciones absolutas de inicio y fin de los datos sobre los que se van a efectuar la operación.

En el segundo caso, el cálculo se hace por software, lo que hace la operación muy lenta:

```
u16 DMAChecksum(u16 start, u16 len, BOOL rx){
        // calculo de checksum vía software
        u16     tmp;
        u16 reg[2];
        u32 sum;
        u16 len2;
        int i;

        if (rx) {
        tmp = TX_BUF_START + 1 + sizeof(MAC_Header) + start;
        } else {
                tmp = packetStart + 6 + sizeof(MAC_Header) + start;
                if (tmp > RX_BUF_END)
                        tmp = tmp - RX_BUF_END + RX_BUF_START - 1;
        }
```

```
        // guarda ERDPT
        setBank(0);
        reg[0] = readETH(ERDPTL);
        reg[1] = readETH(ERDPTH);

        writeReg(ERDPTL,LOW(tmp));
        writeReg(ERDPTH,HIGH(tmp));

        sum = 0;
        len2 = len & 0xFE;
        CS = 0;
        spiWrite(RBM);
        for (i=0; i<len2; i=i+2){
                tmp = ((u16)spiRead()) << 8 | spiRead();
                sum = sum + (u32) tmp;
        }
        if (len2!=len) sum += ((u32)spiRead()) << 8;
        // si el paquete tiene longitud impar
        CS = 1;

        while (sum >> 16)
          sum = (sum & 0xFFFF) + (sum >>  16);

        tmp = ~sum;

        // repara ERDPT;
        setBank(0);
        writeReg(ERDPTL,reg[0]);
        writeReg(ERDPTH,reg[1]);

        return htons(tmp);     // devuelve el checksum calculado
}
```

Antes de efectuar esta operación, hay que guardar el registro ERDPT (restaurado al final), del que se leerán todos los bytes necesarios para el cálculo.

Veamos ahora la función putChecksum que hemos usado anteriormente:

```
void putChecksum(u16 offset, u16 sum){
// escribe el checksum «sum» en la dirección “offset” en el
// buffer de escritura
        u16 tmp;
        u16 addr[2];
        setBank(0);
        addr[0] = readETH(EWRPTL);                    // guarda
                                        // temporalmente el puntero
        addr[1] = readETH(EWRPTH);                    // escritura
        tmp = 1+sizeof(MAC_Header)+offset;            // nueva
                                                      // dirección
        writeReg(EWRPTL,LOW(TX_BUF_START+tmp));
        writeReg(EWRPTH,HIGH(TX_BUF_START+tmp));      // carga el
```

```
                                                    // puntero
        encPut(LOW(sum));
        encPut(HIGH(sum));                          // escribe el
                                                    // checksum
        writeReg(EWRPTL,addr[0]);
        writeReg(EWRPTH,addr[1]);                   // representa el
                                                    // Vector puntero
}
```

4.13 ICMP (Protocolo de Mensajes de Control de Internet)

El ICMP (Protocolo de Mensajes de Control de Internet) es un protocolo de servicio (descrito por la RFC 792), empleado para efectuar controles y señalizaciones dentro de una red. Un paquete ICMP puede llevar diferentes tipos de mensajes, pero el más conocido y del cual hablaremos aquí, es la petición de Eco, conocido también como Ping (Comprueba la conectividad de nivel IP en otro equipo TCP/IP al enviar mensajes de solicitud de eco de ICMP (Protocolo de mensajes de control Internet). Se muestra la recepción de los mensajes de solicitud de eco correspondiente, junto con sus tiempos de ida y vuelta Ping es el principal comando de TCP/IP que se utiliza para solucionar problemas de conectividad, accesibilidad y resolución de nombres).

En el modelo ISO/OSI este protocolo se inserta casi siempre en el tercer nivel, no siendo un verdadero protocolo de transporte, pero de todos modos se encuentra arriba del protocolo IP, dentro del cual vendrá encapsulado.

4.13.1 Estructura del paquete

La estructura de un paquete ICMP (Protocolo de mensajes de control Internet) cambia según el mensaje transportado, excepto un membrete común a todos los tipos. En el caso particular del ping:

- **Type**: Adopta el valor 8 para la petición, y el valor 0 para la respuesta.
- **Code**: No se utiliza.
- **Checksum**: El ya conocido campo de control. El algoritmo empleado es el mismo que el del protocolo IP.

- **Identifier** y **Sequence Number**: Vienen generados por el solicitante para reconocer la respuesta. Quien responde los deja tal y como llegaron.

- **Data**: Contiene un número variable de datos que también se devolverán con el mismo valor.

4.13.2 El código

Veamos el contenido del archivo ICMP.h el cual contiene algunos define, pero sobretodo la definición del paquete ICMP:

```
#define ICMP_ECHO 8
#define ICMP_ECHO_REPLY 0
#define MAX_ICMP_DATA 32       // dimensión máxima del campo de datos

typedef struct {
        u8 type;
        u8 code;
        u16 checksum;
        u16 id;
        u16 sn;
        u8 data[MAX_ICMP_DATA];
} ICMPPacket;

void processICMP(IP_Header ipHeader);
```

Como se puede ver están presentes los campos descritos en el párrafo anterior. El archivo ICMP.c contiene el código que permite responder a un ping.

```
#include "stack.h"
#define ICMP_Offset sizeof(IP_Header)

void processICMP(IP_Header ipHeader){
        ICMPPacket packet;
        u8 size;
        size = ipHeader.totalLength - (ipHeader.verlen & 0x0F)*4;
        if (size > sizeof(packet)) size = sizeof(packet);
        encGetArray((u8*)&packet, size);
        if (packet.type == ICMP_ECHO){
                packet.type = ICMP_ECHO_REPLY;
                packet.checksum = 0;
```

```
                IPPutHeader(ipHeader.sourceIP, IPPROTO_ICMP, (u8*)&packet,
size, size);
                putChecksum(ICMP_Offset+2,
DMAChecksum(ICMP_Offset,size,TRUE));
                MACSend();

        }
}
```

Por medio del método encGetArray el paquete se lee y se guarda en la variable packet. Luego, una vez examinado el campo type (Tipo), si este resulta ser una petición de ping, se montará el paquete de respuesta: el campo type (Tipo) estará a 0 (Echo Reply), el checksum (Control de redundancia) se pondrá a cero y será calculado de nuevo solo cuando el paquete esté escrito en el buffer. Finalmente se enviará todo con MACSend.

4.13.3 ProcessPacket

Para hacer que todo funcione bien, hay que implementar un método que examine los paquetes que llegan y los envíe a los gestores (IP y ARP para el tercer nivel).

Este método se llama processPacket y está mencionado dentro de un bucle infinito en el método main:

```
void main() {
        encInit();
        while (1) {
                processPacket();
        }
}
```

Observe el archivo stack.c

```
#include "stack.h"
MACAddr remoteAddr;

void processPacket(){
        MAC_Header header;
        setBank(1);
        while (readETH(EPKTCNT)) { // si hay por lo menos un paquete
                MACGetHeader(&header);
                if (header.type == TYPE_IP){
                        remoteAddr = header.sourceMAC;
                        processIP();
```

```
                } else
                if (header.type == TYPE_ARP)
                        processARP();
                freeRxSpace(); // deja el espacio en el buffer RX
        }
}
```

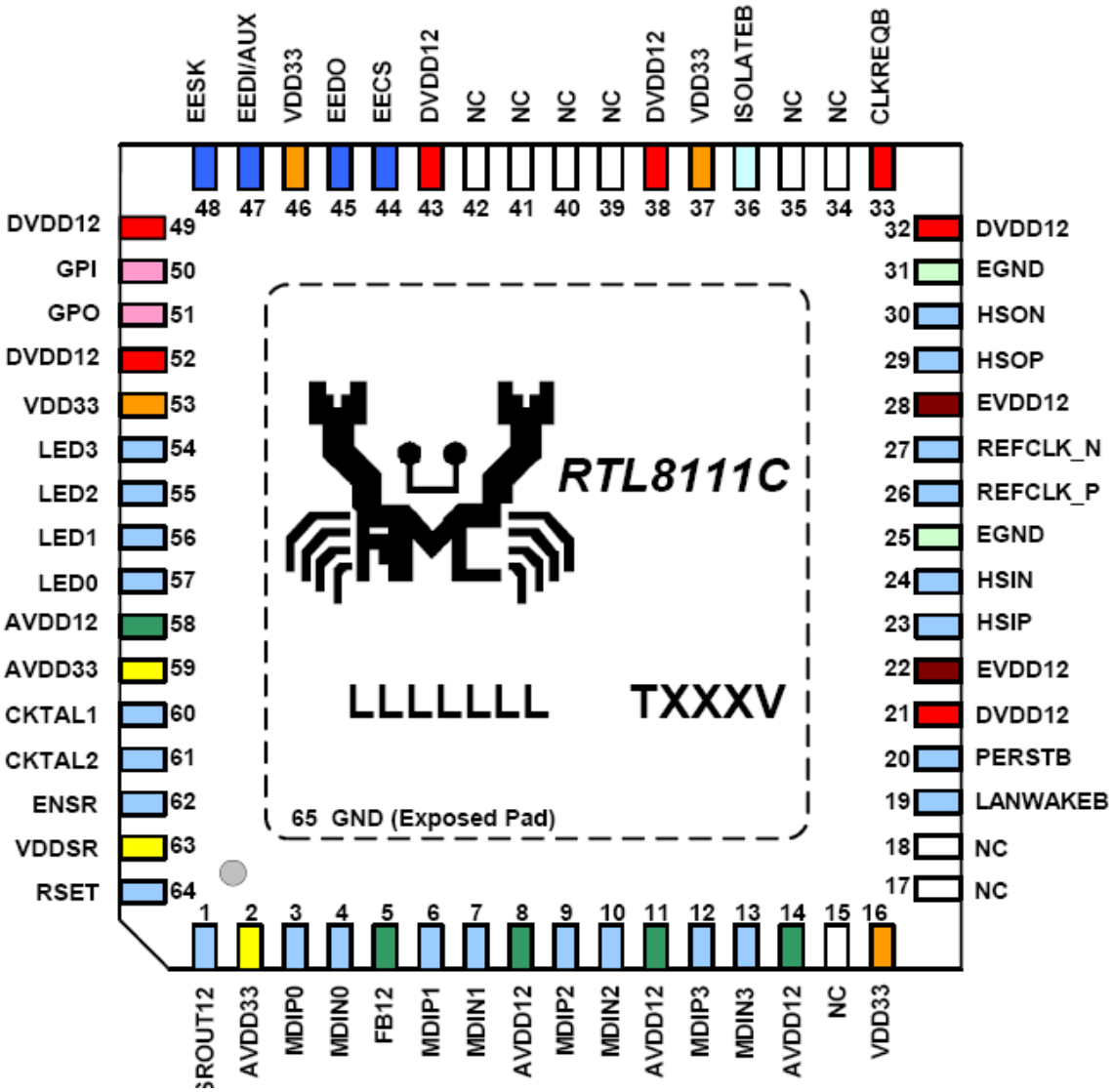
Leyendo el registro EPKCNT se verifica que haya por lo menos un paquete por leer todavía, después con MACGetHeader se lee la cabecera de la dirección MAC (Dirección Física) y por medio del campo type (Tipo) se decide llamar a processIP o processARP (o ninguno de ellos).

Estos métodos ya han sido tratados en las páginas anteriores. En el programa que se adjunta, se han añadido también las funciones para generar un ping, además de la misma rutina que permanece a la escucha y responde a las solicitudes que le llegan.

5. OTRAS OPCIONES DE CONTROLADORES ETHERNET

5.1 RTL8111C

Controlador Gigabit Ethernet con SPI y PCI Express™ integrado



5.1.1 Descripción General

El GR-Realtek RTL8111C Gigabit Ethernet combina un controlador de triple velocidad compatible con IEEE 802,3 (Ethernet), MAC (Control Acceso al Medio) con un triple de la velocidad de Ethernet transceptor, un controlador de bus PCI (Interconexión de Componentes Periféricos) Express, memoria y embebidos. Con el estado de la técnica y la tecnología DSP (Procesamiento Digital de Señales) de modo mixto señal de la tecnología, el GR-RTL8111C ofrece alta velocidad de transmisión de más de CAT 5 UTP o cable UTP CAT 3 (sólo 10 Mbps) de cable.

Funciones tales como detección de cruce y Auto-Corrección, corrección de polaridad, la igualdad de adaptación, cancelación de eco, tiempo de recuperación, y corrección de errores se aplican para proporcionar una sólida capacidad de transmisión y recepción a altas velocidades.

El RTL8111C-GR es compatible con la especificación IEEE 802.3 (Ethernet) para 10/100Mbps Ethernet IEEE 802.3ab y de la especificación para Ethernet de 1000Mbps. También apoya un auxiliar de energía auto-detectar la función, y se auto-configura relacionados bits PCI de los registros de la administración de energía en el espacio de configuración PCI (Interconexión de Componentes Periféricos).

Configuración Avanzada de interfaz de gestión de energía (ACPI) de la administración de energía para los sistemas operativos modernos que son capaces de Sistema Operativo dirigido por la Administración de Energía (OSPM)-cuenta con el apoyo de lograr la más eficiente administración de energía posible. MSI PCI (Mensaje Signaled de interrupción) y MSI-X también están soportados.

Además de la característica de ACPI, wake-up remoto (incluyendo AMD Magic Packet™ y Microsoft® Wake-up frame) se apoya en los dos ACPI y APM (Advanced Power Management) entornos. Para apoyar WOL de un profundo estado de apagado (por ejemplo, D3cold, es decir, el principal de alimentación

está desactivada y sólo existe auxiliares), la fuente de energía auxiliar debe ser capaz de proporcionar la energía necesaria para la RTL8111C-GR.

El RTL8111C-GR es totalmente compatible con Microsoft® NDIS5, NDIS6 (IPv4, IPv6, TCP (Protocolo de Control de Transmisión), UDP (Protocolo de datagramas de usuario) y la segmentación de tareas de control de descargar (Large enviar y enviar Giant) características, y soporta IEEE 802 de la capa IP y la codificación de prioridad 2 IEEE 802.1Q Virtual salvarse red de área local (VLAN). Las anteriores características contribuyen a la reducción de utilización de la CPU, especialmente en beneficio de rendimiento cuando en la operación de un servidor de red.

Soporta Reception Side Scaling (RSS) de las conexiones TCP (Protocolo de Control de Transmisión) hash (Función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc.) y equilibrio de carga de procesamiento de datos recibidos a través de múltiples CPUs. RSS mejora el número de transacciones por segundo y el número de conexiones por segundo, por el aumento de rendimiento de red.

El dispositivo también cuenta de la conexión entre la tecnología PCI (Interconexión de Componentes Periféricos) Express. PCI (Interconexión de Componentes Periféricos) Express es un gran ancho de banda, baja pin cuentan, de serie, la tecnología de interconexión que ofrece mejoras significativas de rendimiento en más de PCI (Interconexión de Componentes Periféricos) convencionales y también mantiene la compatibilidad con el software PCI (Interconexión de Componentes Periféricos) de la infraestructura existente. El dispositivo que lleva incorporado un ecualizador de adaptación en el PCIe PHY (Capa física) para facilitar la integración de sistemas y la excelente relación calidad. El ecualizador permite a la duración de las trazas de PCB (Bloque de control del proceso) para llegar a 40 pulgadas.

El RTL8111C-GR es adecuado para múltiples segmentos de mercado y las nuevas aplicaciones, como equipos de sobremesa, portátiles, estación de trabajo, servidores, plataformas de comunicaciones, y aplicaciones embebidas.

5.1.2 Características

- Soporta estándares 10/100/1000.
- Auto-Negociación con capacidad “Next Page”.
- Soporta PCI Express ™ 1,1
- Crossover Detección & Auto-Corrección.
- Wake-on-LAN y wake-up remoto de apoyo.
- Microsoft ® NDIS5, NDIS6 Checksum Offload (IPv4, IPv6, TCP (Protocolo de Control de Transmisión), UDP (Protocolo de datagramas de usuario) y el Task-Segmentation
- Soporta Full-Dúplex y control de flujo (IEEE 802.3x)
- Totalmente compatible con IEEE 802,3, IEEE 802.3u, IEEE 802.3ab
- Soporta IEEE 802.1P, prioridad codificación de la capa 2
- Soporta IEEE 802.1Q tagged VLAN
- Serial EEPROM
- Buffer de apoyo para Transmisión / Recepción
- Soporta apagado / suspensión de ahorro de energía
- Soporta PCI (Interconexión de Componentes Periféricos) MSI (Mensaje Signaled de interrupción) y MSI-X
- Soporta Reception-Side Scaling (RSS) 64-pin QFN
- Incorpora un ecualizador adaptable en PCI Express PHY (PCB hellas para llegar a 40 pulgadas)

5.2 WIZ W5100 (CONTROLADOR ETHERNET CON INTERFAZ SPI)

Controlador Ethernet con interfaz SPI (modo 0,3), fácil implementación TCP/IP sin necesidad de Sistema Operativo, con capa MAC (Control Acceso al Medio) y PHY (Capa Física) de 10BaseT/100BaseTX incorporadas.

- Soporta los protocolos TCP (Protocolo de Control de Transmisión), UDP (Protocolo de datagramas de usuario), ICMP (Protocolo de mensajes de control Internet), ARP (Protocolo de Resolución de Direcciones), IPv4, IGMP (Protocolo de Dirección Grupo de Internet), PPPoE (Protocolo Punto a Punto sobre Ethernet).

- Soporta Auto negociación (Full-dúplex o Half dúplex).

- Soporta Auto MDI/MDIX.

- Soporta conexiones ADSL (Línea de Abonado Digital Asimétrica) con soporte de protocolo PPPoE (Protocolo Punto a Punto sobre Ethernet) con sistemas de autenticación PAP/CHAP.

- Soporta hasta 4 sockets (Medio por el cual dos programas (posiblemente situados en computadores distintos) pueden intercambiarse cualquier flujo

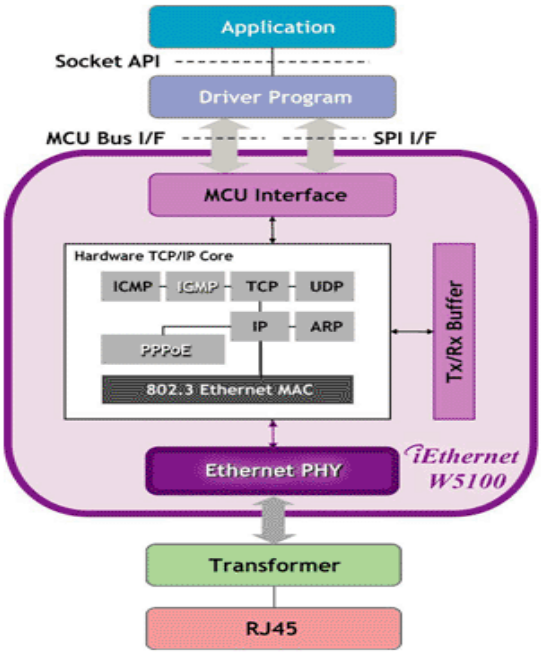
de datos, generalmente de manera fiable y ordenada.) independientes simultáneamente.

- Buffer interno de Tx/Rx de 16Kbytes.

- Tecnología CMOS (Semiconductor Complementario de Óxido Metálico) de 0.18 µm.

- Opera a 3.3V con tolerancia a señanes I/O de 5V.

- Pequeño encapsulado LQFP (Encapsulado Cuadrado Plano de Perfil Bajo) de 80 pines.



6. Bibliografía

http://www.microchip.com

http://www.farnell.com

http://www.wikipedia.org

http://www.todopic.com.ar/

http://www.realtek.com.tw/

http://www.micropic.es/

http://dev.emcelettronica.com/

http://todopic.mforos.com

http://www.datasheetcatalog.net

http://www.wiznet.co.kr

6. LISTA DE ANEXOS

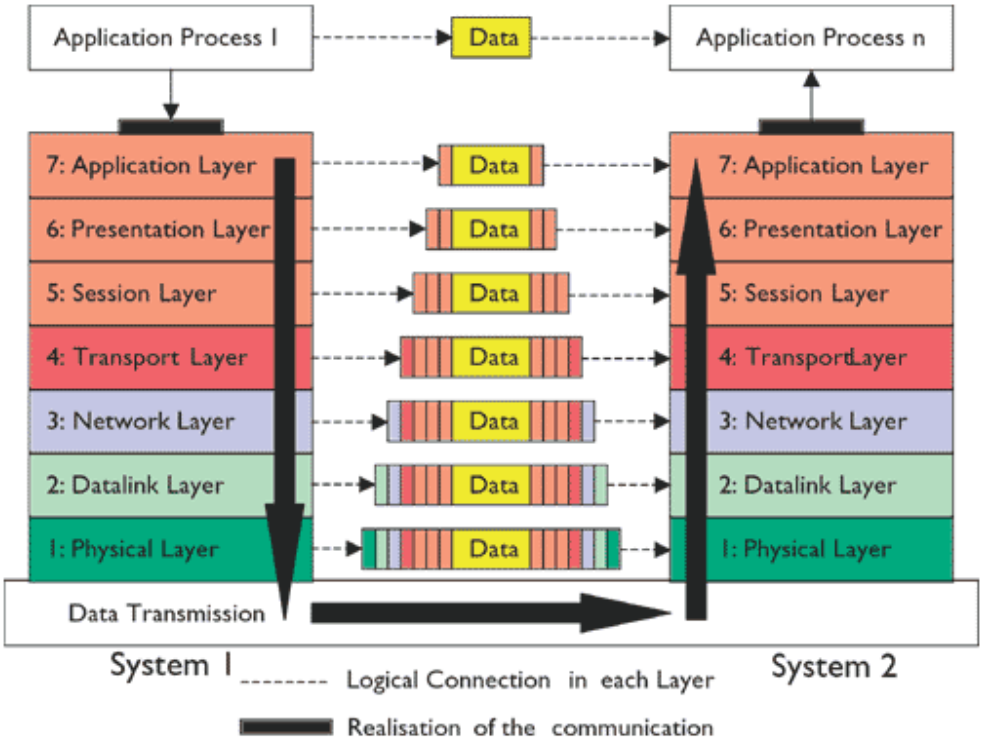
Página

Anexo 1. El Modelo ISO/OSI

El modelo OSI (Open System Interconnection) fue creado en el 1978 por ISO (International Organization Standardization) con el fin de crear un estándar para las comunicaciones entre ordenadores.

Éste está constituido por una pila (stack) de 7 niveles (5 en la versión sencilla); a cada nivel le corresponde un protocolo, por medio del cual dos niveles iguales de sistemas diferentes pueden comunicarse; esto ocurre virtualmente de manera directa, es decir, ignorando los otros niveles.

Dentro del mismo sistema, cada nivel puede comunicar solo con los niveles adyacentes, por medio de una interfaz.



- **Nivel 1**: *físico*. Se ocupa de transmitir datos en el medio físico (cable, aire, fibra óptica, etc.)

- **Nivel 2:** *Enlace de Datos*. Se encarga del direccionamiento físico (MAC y LLC).

- **Nivel 3:** *Red*. Se encarga del direccionamiento lógico, (selección de rutas) y control de congestión.

- **Nivel 4:** *Transporte*. Asegura la correcta recepción de los datos, se ocupa de las retransmisiones en caso de errores y permite establecer conexiones.

- **Nivel 5:** *Sesión*. Esta capa establece, gestiona y finaliza las conexiones entre usuarios (procesos o aplicaciones) finales.

- **Nivel 6:** *Presentación*. El objetivo de la capa de presentación es encargarse de la representación de la información, de manera que aunque distintos equipos puedan tener diferentes representaciones internas de caracteres (ASCII, Unicode, EBCDIC), números (little-endian tipo Intel, big-endian tipo Motorola), sonido o imágenes, los datos lleguen de manera reconocible

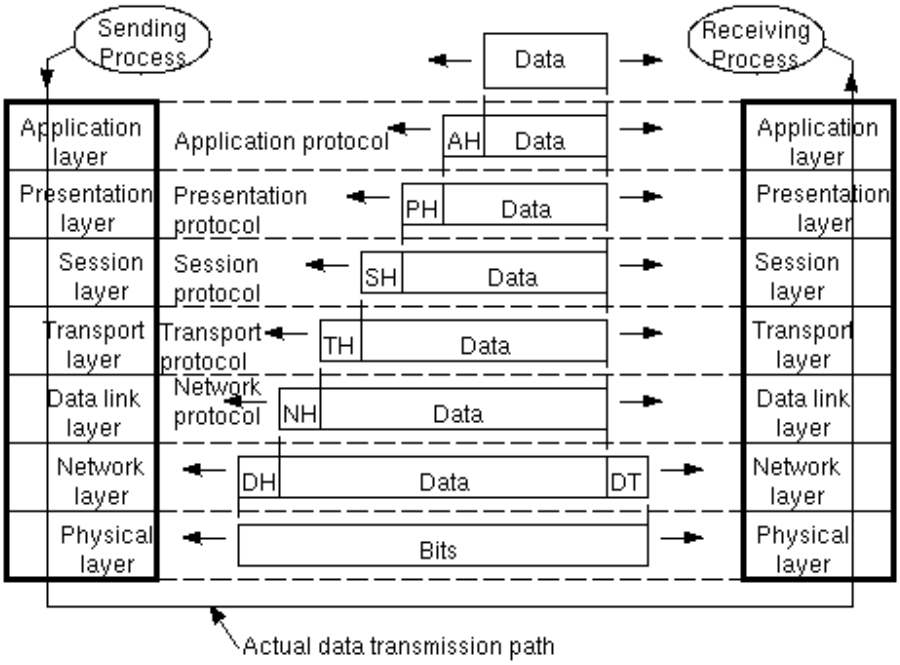
- **Nivel 5:** *Aplicación*. Es el último nivel de la pila y como sugiere el nombre es donde están los servicios (HTTP, FTP, E-mail, etc.).

Anexo 2. Encapsulamiento

En un paquete de red, los datos de los distintos niveles están encapsulados uno en el otro, como en una matrioshka (muñeca rusa). Esto significa que:

- En fase de transmisión cada nivel envía sus datos al nivel inferior que añade su cabecera (y cola) secuencialmente hasta que se llega al medio físico

- En fase de recepción cada nivel examina su cabecera y pasa los datos al nivel superior



Hay que tener en cuenta que la comunicación no concierne necesariamente el último nivel, sino ocurre siempre entre dos niveles iguales.

Esta modularización permite, por ejemplo, transportar los mismos datos en soportes físicos diferentes: por ejemplo las redes Ethernet y WiFi que transportan las dos paquetes TCP/IP, pero sobre medios diferentes (cable y aire).

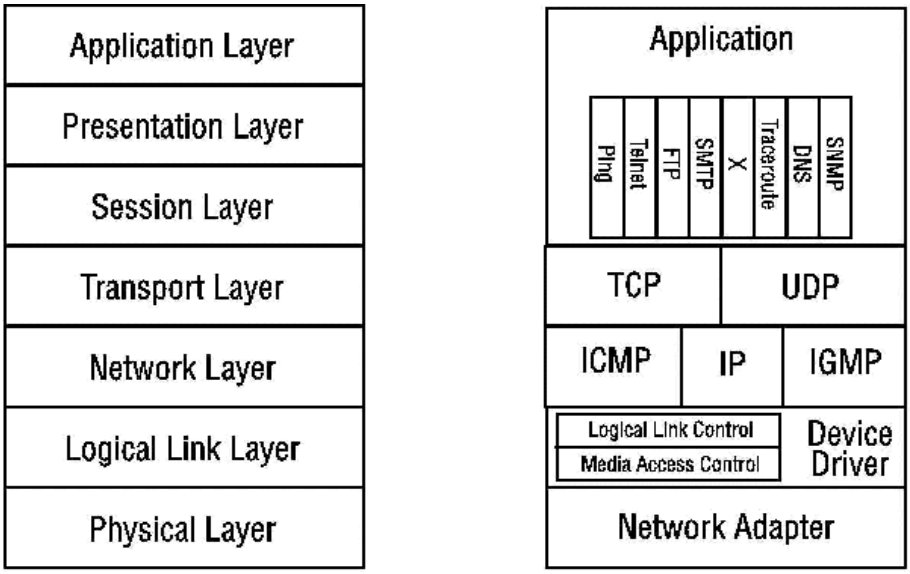
Anexo 3. Ethernet

Ethernet es un protocolo de tipo CSMA/CD (Carrier Sense Multiple Access / Collision Detect) desarrollado en el 1973, con el fin de conseguir transmisiones por cable fiables en condiciones de tráfico moderado.

De ahí nace el estándar IEEE 802.3 de 1985 (última revisión en el 2002) que es parte de la gran familia de protocolos IEEE 802.

Esta familia establece estándares para numerosas topologías de red (como Token Ring, Token Bus, WiFi, etc.). Se puede dividir el segundo nivel OSI en dos subniveles, el superior, LLC (Logical Link Control), es común a todos los estándares, mientras la parte inferior, el MAC (Medium Access Control), está unido al nivel físico.

El subnivel LLC ofrece diferentes servicios a menudo pertenecientes a los niveles superiores y de todos modos no previstos desde el primer Ethernet; por esto, la arquitectura TCP/IP utiliza el viejo framming (dicho DIX) que no usa el LLC, mientras otros protocolos usan el estándar "oficial".



Anexo 4. Direcciones MAC (Dirección Física) e IP

En una red Ethernet, cuando un paquete se envía, cualquier sistema conectado a la misma red lo recibe; es entonces necesario identificar de manera unívoca el destinatario y el remitente (para la respuesta). Esto se consigue gracias a una dirección MAC de 6 bytes, asociada a nivel mundial con cada NIC (Network Interface Controller), o sea, cada dispositivo (tarjeta de red etc.) que pueda transmitir y recibir datos en una LAN (Local Area Network)..

Pero como ya sabemos, un ordenador está identificado también por una dirección IP, pero en las redes locales un sistema sólo puede ser individualizado por medio de su dirección MAC. Existe un protocolo de conversión entre estas dos direcciones (física y lógica), llamado protocolo ARP.

Un NIC (Network Interface Controller), para enviar un paquete a un nodo de la misma red, resuelve la dirección MAC del destinatario y después procede; si en cambio el paquete está destinado a una red fuera de la suya (por ejemplo Internet), será empleada la dirección MAC del gateway (por ejemplo un router ADSL) para poder dirigir los datos.

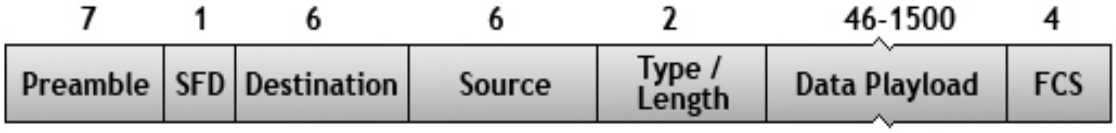
Mientras la dirección IP puede ser configurada según las exigencias, la dirección MAC se graba en el NIC en fase de producción y casi nunca puede ser modificada, otorgando unicidad al dispositivo.

La dirección MAC está dividida en dos partes de tres bytes cada una: la primera está asignada por el IEEE a cada sociedad que hace la petición; la segunda se usa para generar direcciones diferentes para cada tarjeta producida desde la misma sociedad.

Con fin didáctico podemos utilizar las direcciones asignadas a Microchip (de 00:04:A3:00:00:00 a 00:04:A3:FF:FF:FF), pero con finalidades comerciales, las direcciones tienen que ser compradas.

Anexo 5. El Nivel MAC (DATALINK)

Ahora veamos como está constituida una trama MAC, es decir, el paquete "confeccionado" desde el nivel Enlace de Datos en el estándar IEEE 802.3.



- ***Preamble*:** Constituido por una serie de 1 y 0 para permitir a el receptor de sincronizarse con el transmisor.
- ***SFD*:** Start-of-Frame Delimiter, señala a el receptor que está listo para empezar la trama verdadera.
- ***Destination*:** Contiene la dirección MAC del destinatario.
- ***Source*:** Contiene la dirección MAC del remitente.
- ***Length/Type*:** Según el estándar 802.3 este campo puede asumir dos significados diferentes: si el valor es menor o igual a 1500, indica la longitud del campo datos donde se cree que sea presente un paquete LLC que será procesado desde el mismo subnivel, si no indica el protocolo de tercer nivel contenido en el campo datos; en este último caso la trama MAC es una trama DIX, por eso no sigue un paquete LLC, sino los datos vienen pasados directamente a el nivel 3.
- ***Data*:** Aquí están contenidos los datos recibido por el nivel superior; la longitud mínima es de 46 bytes, si este límite no se respeta, en fase de transmisión el nivel MAC añade un campo de padding para llenar el espacio que queda.
- ***FCS:*** Frame Check Sequence, constituido de 4 bytes para el control de errores (CRC).