

Módulo [Ethernet](#) ENC28J60 para microcontroladores PIC

Introducción

El intento de este artículo es el de realizar una interfaz Ethernet para microcontroladores PIC, utilizando el controlador ENC28J60 de Microchip.

Esto no incluye solo la parte Hardware, sino también (y sobretodo) la del Software: por eso no trataremos el uso del stack TCP/IP oficial, sino la creación de un software "custom" con finalidades solo didácticas.

Software

El software de Microchip incluye muchas características interesantes, pero tiene algunas limitaciones y no es muy sencillo utilizarlo.

También Microelectrónica, ha realizado una pequeña librería para el uso del controlador (se encuentra con el compilador mikroC), pero este no soporta la fragmentación de los paquetes (y no es posible modificarla), no corresponde mucho a la realización de un Servidor HTTP y de cualquier aplicación que tenga un intercambio de datos bastante consistente (el limite es mas o menos de 1.4KB).

Realizar un propio Stack es un óptimo modo para aprender los mecanismos que permiten el funcionamiento de las redes, pero principalmente ayuda la implementación del protocolo y funcionalidad no previstos en otros Stacks.

Hardware

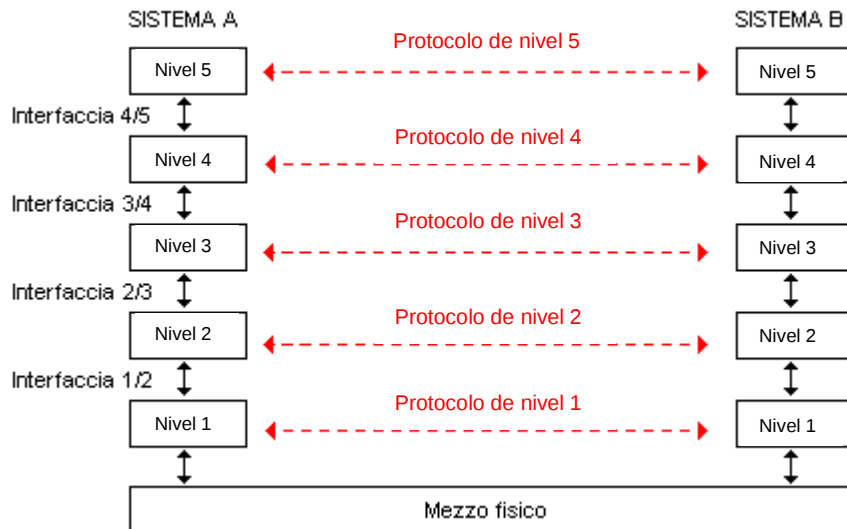
Será ilustrado el hardware necesario para el funcionamiento del controlador y veremos cuales son las funciones y como se configura el ENC28J60.

El modelo ISO/OSI

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

Está constituido por una pila (Stack) de 7 niveles (5 en la versión sencilla); por cada nivel corresponde un protocolo, por medio del cual dos niveles iguales de sistemas diferentes pueden comunicarse; esto ocurre virtualmente en manera directa o sea ignorando los otros niveles.

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



Los niveles del modelo simplificado

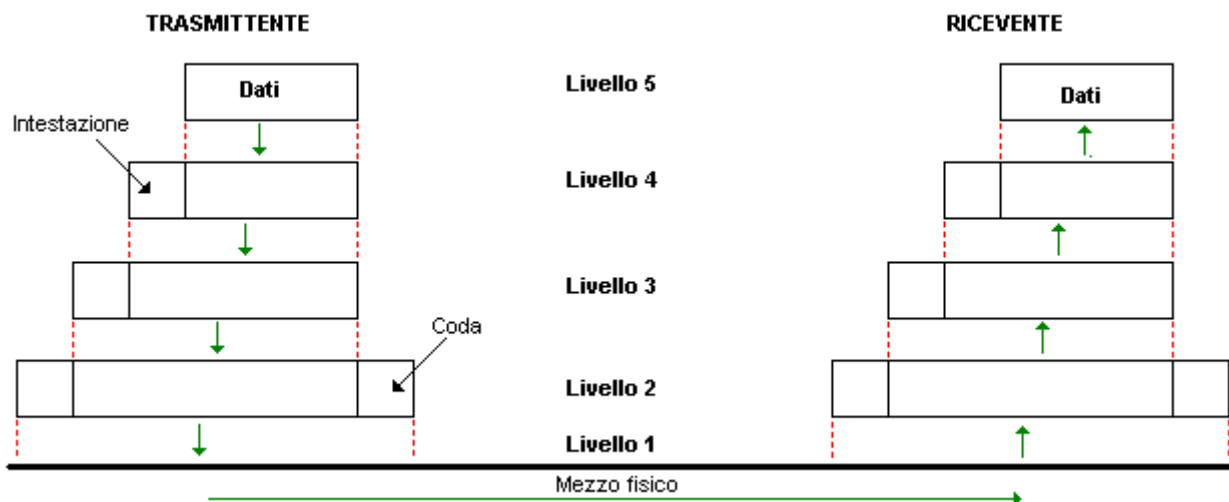
- Nivel 1: *Físico*. Se ocupa de transmitir datos en el medio físico (cable, aire, fibra óptica, etc.)
- Nivel 2: *Datalink*. Prepara los datos para que sean enviados al medio físico.
- Nivel 3: *Rete*. Sirve para "[instradare](#)" los paquetes en la red, además añade un destaco del nivel físico.
- Nivel 4: *Transporte*. Asegura la correcta recepción de los datos, se ocupa de las retransmisiones en caso de errores y permite de establecer conexiones.
- Nivel 5: *Aplicación*. Es el último nivel de la pila y como sugiere el nombre es donde están los varios servicios (HTTP, FTP, E-mail, etc.).

Encapsulamiento

En un paquete de red, los datos de los varios niveles están encapsulados uno en el otro, como en una matrioshka.

Esto significa que:

- En fase de transmisión cada nivel envía los propios datos al nivel inferior que añade el propio título (y cola) y lo envía al nivel inferior a él hasta que se llega al medio físico.
- En fase de recepción cada nivel examina su título y pasa los datos al nivel superior



Hay que notar 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, las dos transportan paquetes TCP/IP, pero sobre medios diferentes (cable y aire).

Ethernet

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

Desde esto nace el estándar IEEE 802.3 en 1985 (última revisión en el 2002) que forma 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.), por eso se pensó en dividir el segundo nivel OSI en dos sub-niveles, 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 sub-nivel LLC ofrece diferentes servicios, a menudo dejados a los niveles superiores y de todos modos no previstos desde el viejo Ethernet; por esto, la arquitectura TCP/IP utiliza el viejo framing (dicho DIX) que no usa el LLC, mientras otros protocolos usan el estándar "oficial".

Direcciones MAC 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 sucede gracias a una dirección MAC de 6 bytes, asociada a nivel mundial a cada NIC (Network Interface controlador) 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 puede ser individualizado solo por medio de su dirección MAC y entonces existe un protocolo de conversión entre estos dos (protocolo ARP).

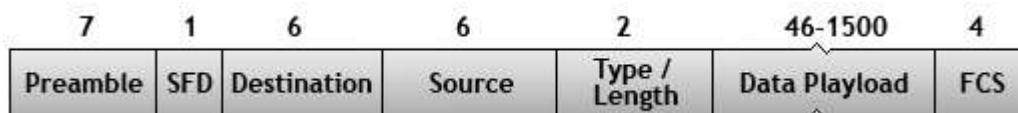
Un NIC, para enviar un paquete a un nodo de la misma red, encuentra la dirección MAC del destinatario y después procede; si en cambio el paquete está destinado a una red fuera de esta (por ejemplo Internet), será empleada la dirección MAC del gateway (por ejemplo un router ADSL) que proveerá a dirigir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

Mientras la dirección IP puede ser configurada según las exigencias, la dirección MAC está escrita en el NIC en fase de producción y generalmente no puede ser modificada.

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 la 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.

El nivel MAC (Datalink)

Ahora veamos como está constituida una trama MAC, o sea el paquete "confeccionado" desde el nivel datalink en el estándar IEEE 802.3.



- *Preamble*: Constituido por una serie de 1 y 0 para permitir al receptor que sincronizarse con el transmisor.
- *SFD*: Start-of-Frame Delimiter, señala al receptor que está por 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 sub-nivel, 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 pasados del 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 los errores (CRC).

El controlador ENC28J60

El ENC28J60 de Microchip es un controlador Ethernet 10Base-T (10Mbps en cables), cercano al estándar IEEE 802.3.

Está constituido por un módulo PHY (nivel físico), un módulo MAC (sub-nivel MAC), una memoria RAM de 8kbytes para almacenar los paquetes en recepción y en transmisión, una serie de registros de configuración y un módulo para la comunicación serie SPI. El chip tiene solo 28 pines y requiere pocos componentes externos para funcionar, por eso puede ser insertado muy sencillamente en cualquier proyecto.

Revisiones y Errata

El ENC28J60, como cada chip de Microchip, tiene un número de revisión (REVID), por lo tanto cada chip con un cierto REVID puede tener algunas diferencias respecto a un chip con otro número.

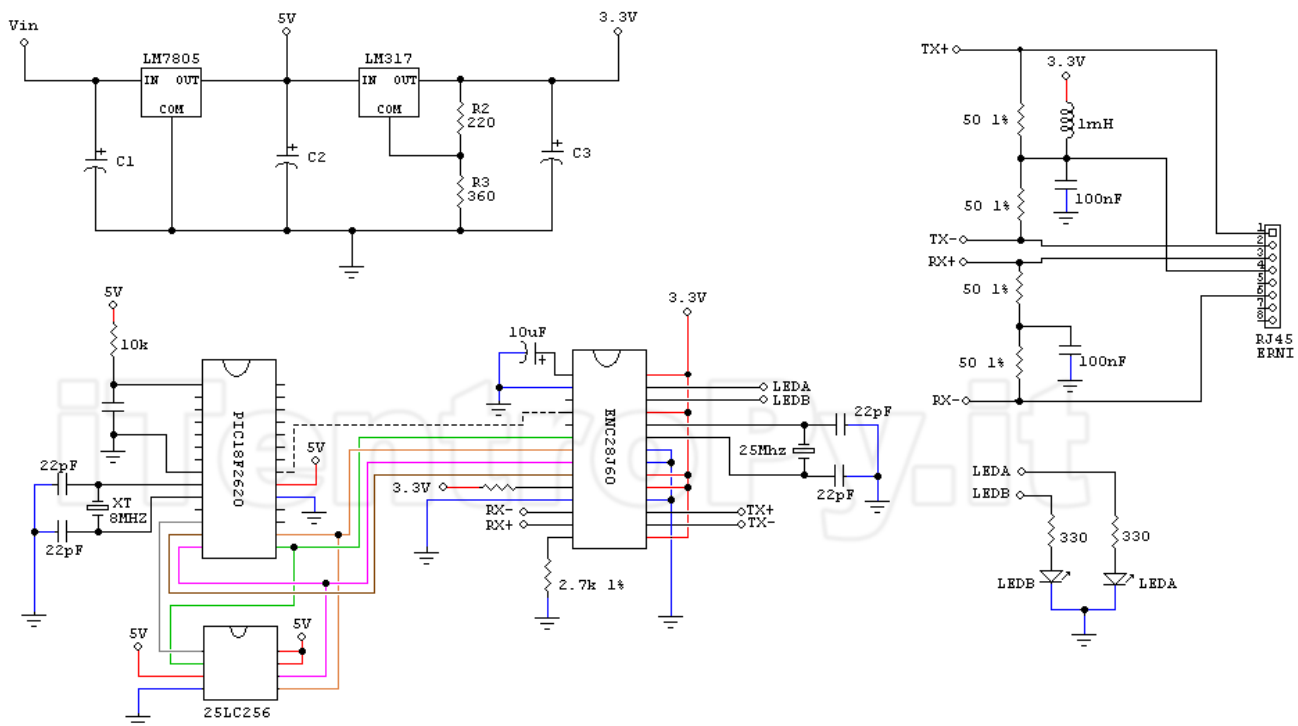
Para saber la versión del chip que se posee existen solo dos modos: el primero es el de leer un registro interno por medio de una interfaz SPI; el segundo es el de pedir a la asistencia de Microchip, si puede hallar el REVID por medio del serial de siete cifras escrito sobre el chip.

La revisión más reciente es la B5, y todavía es posible crear software y hardware sin preocuparse del número de revisión; la única cosa que varía en nivel hardware es el valor de una resistencia, que veremos dentro de poco.

Además del [datasheet](#) del controlador, un documento muy importante es la [Errata](#) (relativa a cada revisión): en esta están presentes algunos problemas del chip y relativas soluciones, y además las diferencias entre una revisión y otra.

El Hardware

Este es el esquema del circuito que yo realicé para escribir este artículo.



El ENC28J60 está alimentado por una tensión de 3.3V (máx. 180mA), por lo tanto existen diferentes configuraciones posibles:

- Para alimentar todo el circuito (PIC incluido) a 3.3V, puede ser utilizado un PIC18F25J10 o similar, que es capaz de trabajar a 40Mhz; otros PIC también trabajan con esta tensión, en cambio, funcionan a frecuencias menores.
- Alimentando el PIC a 5V, son necesarios unos adaptadores de nivel (puertas lógicas sencillas CMOS, tipo 74HC08), para las salidas del ENC que van al PIC (SO, CLKOUT, INT, WOL); las entradas del controlador (CS, SCK, SI, RESET) son en cambio tolerantes a los 5V.
- Los PIC, a menudo, tienen como valor de umbral para distinguir un valor lógico 1 en entrada de 2V, por eso estos adaptadores de nivel no parecen indispensables, como se describe en el datasheet del ENC28J60, por esto en mi esquema no están presentes.

El controlador está diseñado para trabajar a 25 MHz, por lo tanto es necesario un cristal de esta frecuencia entre los polos OSC1 e OSC2, mas dos condensadores cerámicos conectados a masa. El valor de estos condensadores no está especificado en el datasheet, pero presumo que sean mas o menos de 15pF.

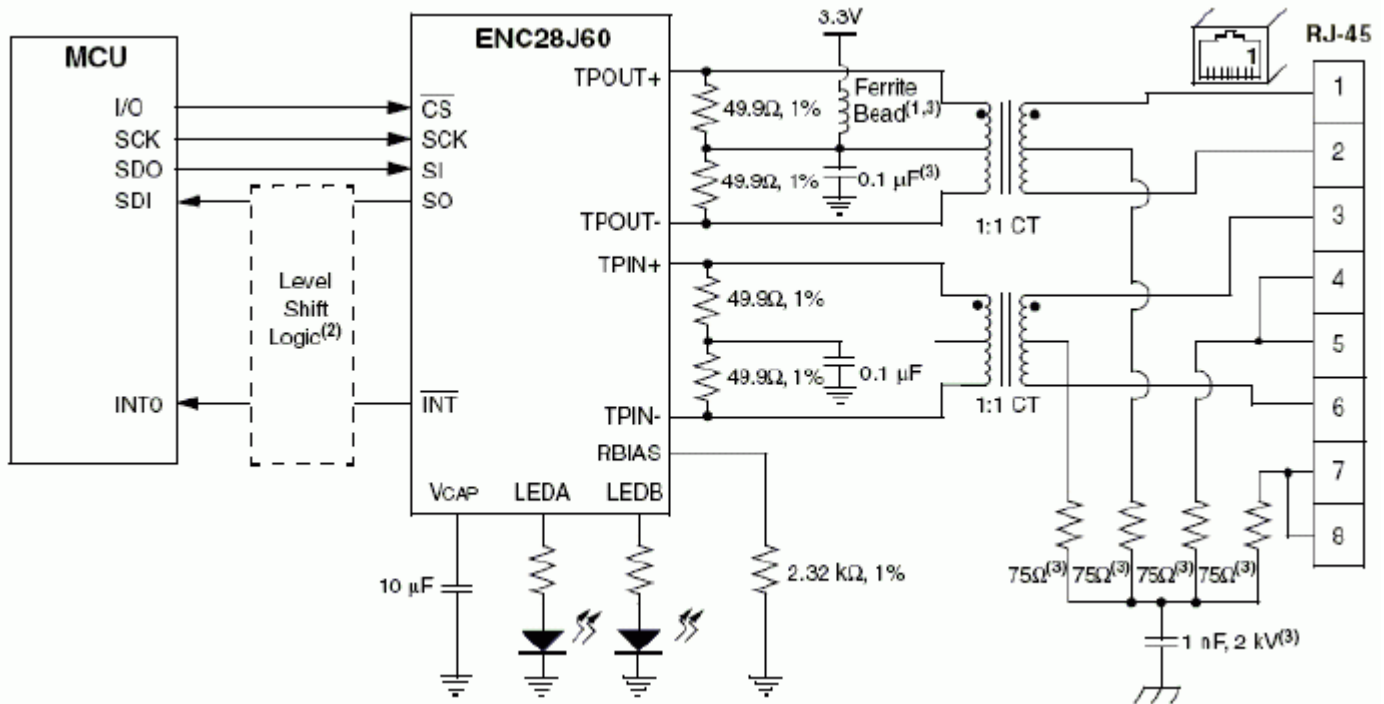
Puede también ser utilizado un clock externo, conectado al pic OSC1, pero tiene que ser muy preciso (+/- 50ppm).

Un componente importante es la resistencia RBIAS, su valor cambia según la revisión del chip: para las revisiones B1-B4 es de 2.7Kohm 1%, mientras para la B5 es de 2.32K 1%; su valor es importante para respetar las especificas IEEE, por lo tanto es mejor ser seguros de la revisión que se posee (veremos después como hacer).

El datasheet menciona una "ferrite bead", pero no especifica el valor: en mi esquema la sustituí con una inductancia de 1mH.

Los últimos componentes de relieve son el conector RJ45 y relativos filtros (Magnetics): existen conectores con filtros integrados, o conector y filtros a parte. En mi caso usé un conector [ERNI](#) con filtros integrados (cuidado al esquema, la polarización podría ser diferente de otro tipo de conector).

De todos modos es indispensable seguir con cuidado el esquema presente en el datasheet, que pongo aquí seguido (los transformadores 1:1 son los Filtros):



Los Magnetics son cajas negras que se pueden recuperar (junto al conector) desde las tarjetas de red.

Notas sobre el hardware:

- El pin INT (interrupt) no se utiliza por el software, por lo tanto si no se quieren utilizar las interrupciones puede no ser conectado.
- El ENC28J60 tiene entre sus características la *auto-polarity*, o sea los pines TPIN+ y TPIN-, aunque si invertidos, funcionan lo mismo; esto en realidad es falso (el problema está descrito en el Errata), por lo tanto es aconsejable un control preciso de estos polos en fase de proyecto/montaje del circuito.
- El datasheet aconseja insertar condensadores cerámicos de 0.1µF, para cada pareja de polos Vcc-Vss.

Organización de la Memoria

La memoria del chip está dividida en dos partes: el buffer Rx/Tx y los registros de control. A los dos se accede por medio de la interfaz SPI.

El buffer es una dual port RAM, configurable en modo que se pueda dividir, como quieras, entre la memoria de recepción y la de transmisión.

La estructura de los registros de control es igual a la presente en los PIC: ella está constituida por una serie de registros a 8bit, divididos en bancos, por medio de los cuales es posible configurar el dispositivo.

Estos se dividen en registros ETH, MAC, MII y PHY; los primeros tres se encuentran en los bancos de registros, mientras a los registros PHY se accede por medio de los registros MII.

Bank 0		Bank 1		Bank 2		Bank 3	
Address	Name	Address	Name	Address	Name	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	ERXRPTL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRPTH	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
1Eh	ECON2	1Eh	ECON2	1Eh	ECON2	1Eh	ECON2
1Fh	ECON1	1Fh	ECON1	1Fh	ECON1	1Fh	ECON1

Veremos la función de los registros cada vez que los usemos.

La interfaz SPI

La comunicación entre el ENC28J60 y el PIC ocurre por medio de la [interfaz SPI](#); esta soporta solo la modalidad 0,0 y el controlador es esclavo, por lo tanto es el PIC que suministra el clock y maneja la transmisión.

La máxima frecuencia admitida es de 10Mhz para las Rev. B1-B4, mientras que es el doble para la B5. Además, a causa de un problema en la interfaz (descrito en la Errata), para las Rev. B1-B4 el clock tiene que estar necesariamente entre los 8 y los 10Mhz, o el clock del PIC tiene que ser sacado 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.

Mandos SPI

Por medio de la interfaz es posible enviar al chip 7 comandos diferentes (de 8 bit), seguidos por un byte de datos; estos son:

Nombre	1° 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 dentro del registro A (en el banco seleccionado)
Read Buffer Memory (RBM)	001	11010	Sirve para leer la memoria RAM del controlador a la dirección corriente.
Write Buffer Memory (WBM)	011	11010 DDDDDDDD	Escribe el byte D a la dirección corriente de la memoria RAM.
Bit Field Set (BFS)	100	AAAAA DDDDDDDD	Programa en el registro A (solo ETH), los bits que en D son a 1 (OR).
Bit Field Set (BFS)	101	AAAAA DDDDDDDD	Reposiciona en el registro A (solo ETH), los bit que en D son a 1 (NOT AND).
System Reset Command (SRC)	111	11111	Reposiciona el controlador.

Mandos SPI: implementación

La inicialización del módulo SPI será ilustrada mas adelante, por el momento vemos algunos métodos de base.

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

Esto es el método que nos permite de leer/escribir en el bus SPI. Por lo que concierne la escritura, el byte que se tiene que enviar (data) se pone en el registro SSPBUF (que hace parte del módulo SPI), después se espera que la transmisión termine observando el bit SSPIF; para leer un byte, en cambio, es necesario escribir uno cero en el registro SSPBUF (así vienen generados 8 impulsos de clock), se espera el fin de la operación y el byte leído se encontrará en el mismo registro SSPBUF.

En este caso he condensado en un único método las operaciones de lectura y escritura, después, para claridad mental, he 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 estos tres métodos podemos leer y escribir los registros de control (para leer los registros MII se usa readMAC).

Como se nota el procedimiento de lectura de un registro MAC y ETH es un poco diferente, en cuanto por el primer tipo se tiene que enviar un byte 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 vemos la implementación de los comandos Bit Set y Bit Clear; luego el método setBank, con el cual podemos 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;
}

```

Los tres comandos que quedan son realizados por estos tres métodos: lectura y escritura de la RAM, y reset. La variable estática bufSize sirve a tener traza del número de byte escritos en la RAM (servirá después).

Los registros PHY

Los registros PHY son de 16 bits y accesibles por medio de los registros MII. Para leer un registro PHY:

- Se introduce la dirección en el registro MIREGADR.
- Programando el bit MIIRD (1) del registro MICMD, empieza la lectura.
- Se espera el fin de la lectura observando el bit BUSY (1) del registro MISTAT.
- Se reposiciona el bit MIIRD.
- El dato será presente en los registros MIRDH e MIRDH.

Para escribir en un registro PHY:

- Se introduce la dirección del registro en MIREGADR.
- Antes se escribe el byte menos significativo en MIWRL, después escribiendo el byte mas significativo en MIWRH empieza la escritura.
- Se espera que el módulo PHY 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(MIRDH) | (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);
}

```

Otros métodos

Los siguientes sirven para escribir/leer mas bytes de la RAM del controlador; la dirección desde la cual vienen leídos está situada en los registros ERDPT, mientras la dirección de escritura está situada en los registros EWRPT (son auto-incrementantes).

```
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;
}
```

Inicialización SPI

La inicialización del módulo SPI tiene lugar con estas dos simples instrucciones:

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

    SSPSTAT = 0x40;
    SSPCON1 = 0x20;
```

En particular el módulo MSSP viene habilitado y configurado en modalidad "0,0", con clock igual a $F_{osc}/4$.

Inicialización del ENC28J60

La inicialización del controlador necesita de la configuración de diferentes registros, y de la habilitación a 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));              // apuntador del buffer de lectura
writeReg(ERXNDL,    LOW(RX_BUF_END));              //
writeReg(ERXNDH,    HIGH(RX_BUF_END));              // termino buffer de lectura
writeReg(ETXSTL,    LOW(TX_BUF_START));            //
writeReg(ETXSTH,    HIGH(TX_BUF_START));            // inicio buffer de escritura
```

Como ya hemos dicho antes, el buffer del ENC28J60 puede ser dividido como se prefiere entre la memoria de transmisión y la de recepción.

Para hacer esto se configuran los punteros del buffer de recepción; la memoria que queda será el buffer de transmisión.

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

En ERXRDP, en cambio, está el puntero de lectura de la memoria RX, o sea contraseña una zona (junto a ERXWRPT) que tiene que ser todavía elaborada del PIC y por supuesto no puede ser escrita; al principio el valor de esta dirección está igual a ERXND (tiene que ser impar según un problema descrito en el Errata). El registro ERXWRPT vale cero al reset y se actualiza automáticamente a la recepción de un paquete.

```
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);              // Inter-Packet Gap
```

Estos registros configuran el módulo MAC. A través del registro MACON1 se habilitan el módulo MAC y la recepción/transmisión de tramas de pausa.

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

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

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

En los registros MAXFL (máx. Frame Length) se guarda la máxima dimensión admitida para un paquete; el controlador puede ser configurado de manera que no envíe un paquete que supera este límite.

```
#define MY_MAC1      0x00
#define MY_MAC2      0x04
#define MY_MAC3      0xA3
#define MY_MAC4      0x00
#define MY_MAC5      0x00
#define MY_MAC6      0x00
....
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);
```

La dirección MAC de nuestro dispositivo se guarda en los registros MACADR; estos son utilizados solo por el filtro para rechazar paquetes que no sean destinados al controlador, por lo tanto la dirección no se inserta automáticamente en los paquetes que se tienen que enviar.

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

setBank(1);
writeReg(ERXFCN, 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 operaciones que vienen efectuadas son la inhabilitación del LoopBack (empleado para hacer test) y la habilitación del módulo. Los filtros son programados de manera que acepten solo paquetes destinados a la dirección MAC configurada y paquetes de broadcast.

Leer el RevID

Llegados a este punto ya tenemos todo el código necesario para cambiar datos con el controlador, podemos entonces leer el registro EREVID, situado en el banco 3, el cual contiene el número de revisiones del Chip.

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

En alternativa, dejando int conectado, se puede shiftar PORTB a la izquierda.

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

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

    while (1);
}
```

El código completo está anexo a este artículo.

Fuentes

[\[C18\] 01 - inicialización](#)

[\[MikroC\] 01 - inicialización](#)

Transmisión

El ENC28J60, en nivel MAC, se ocupa de generar los campos Preamble, SFD, el eventual padding y el FCS; todo lo que queda tiene que ser insertado por el software en el buffer de transmisión. Además, el byte que se encuentra en la primera ubicación de memoria de tal buffer, es un byte de control y no viene efectivamente enviado. Utilizando cero como valor de este byte, se usan, para las transmisiones, las opciones ya programadas en MACON3.

Hemos definido el buffer de transmisión de manera que empiece en la dirección TX_BUF_START, por lo tanto las operaciones que se tienen que seguir para la preparación de la transmisión MAC son:

- Se programan los registros EWRPT de modo que punten en el principio del buffer TX.
- Enviando el mando 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 del destinatario, el del remitente, y por fin el campo Type/Length.

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 file 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;

```

Entonces la intestación 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 que el controlador este listo para transmitir observando el bit TXRTS (3) del registro ECON1.
- Los registros ETXND vienen cargados con la dirección del último byte que se tiene que enviar. Esta dirección será TX_BUF_START + bufSize.
- Programar el bit TXRTS empieza la transmisión.

Por 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 verificó un error
        BFSReg(ECON1, 0b10000000);           //
        BFCReg(ECON1, 0b10000000);           // reposiciona el TX
    }
    while(readETH(ECON1) & 0b1000);             // espera que sea listo para enviar
    writeReg(ETXNDL, LOW(TX_BUF_START + bufSize));
    writeReg(ETXNDH, HIGH(TX_BUF_START + bufSize));
    BFSReg(ECON1, 0b1000);                     // envía
}

```

Recepción

Antes de todo, para verificar si ha sido 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 elaborado el paquete, tiene que ser decrementado. La recepción que tuvo lugar puede ser vista a través del bit EIR.RXIF, pero según la Errata, el valor de este bit no está fiable.

Los datos recibidos vienen escritos por el controlador en el buffer de recepción empezando por la dirección ERXWRPT, que en fase de inicialización está automáticamente programada en cero.

En esta dirección los primeros seis bytes no pertenecen al paquete, sino son bytes de control: los primeros dos contienen la dirección donde será escrito el próximo paquete; los otros no son muy importantes (para ahondamientos, consultar el datasheet). definimos dos nuevas variables, la primera va inicializada a cero en *encInit*:

u16 RdPt;

u16 packetStart;

RdPt contiene la dirección del próximo paquete, mientras packetStart la dirección del paquete corriente.

Dicho esto vemos el método *MACGetHeader*:

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

    packetStart = RdPt; // salva RdPt in
    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)); // apuntador próximo
    paquete

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

    header->type = htons(header->type); // swappa el campo type
}
```

Como se puede ver, como primera cosa se carga en ERDPT la dirección del paquete que se tiene que leer (salvado en RdPt, al principio cero), luego vienen leídos los byte de control y se salva el nuevo RdPt; en fin se lee la intestación MAC y se salva en header. La instrucción htons está definida en el file utility.c; el motivo de su utilizzo está aclarado a continuación:

Pequeños y grandes indios

En los comunes ordenadores y microcontroladores las locaciones de memoria mínimas son comúnmente de 8 bit; pero cuando se tienen que representar valores de 16 bit, hay que decidir como vienen salvados en memoria. Un dato de 16bit claramente ocupa dos locaciones de memoria, por lo tanto hay dos posibilidades:

- el byte menos significativo se encuentra en la dirección mas baja respecto al MSB.
- viceversa, el byte menos significativo se encuentra en la dirección de memoria mayor de los dos.

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

Lo que tiene que hacer la función `htons` es de convertir un dato de 16 bit entre las dos notaciones. En el artículo será empleado frecuentemente y hay que poner mucho cuidado en su utilización.

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

Como hemos ya visto el registro `ERXWRPT` contiene la dirección donde el controlador está escribiendo los datos recibidos, mientras 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 a la escritura, para impedir la pérdida de los datos durante su elaboración.

Dicho esto, es necesario, terminada la examinación del 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); // decremento EPKTCNT
    writeReg(ERXRDPTL, LOW(NewRdPt));
    writeReg(ERXRDPTH, HIGH(NewRdPt)); // libera el espacio del buffer
}
```

En este método viene actualizado el registro citado aquí arriba, y viene decrementado `EPKTCNT`.

Por causa de un problema descrito en la Errata, el valor de `ERXRDPT` tiene que ser impar; pero sabemos que `RdPt` está seguramente par (el controlador añade un padding de manera que sea par), por supuesto es suficiente restar uno a este valor, y después controlar que no vaya fuera de los límites del buffer (`RX_BUF_START` e `RX_BUF_END`).

ARP

El ARP ([Address Resolution Protocol](#)) es un protocolo de tercer nivel, empleado dentro de redes LAN, para traducir direcciones IP en las correspondientes direcciones físicas (MAC). Hemos ya analizado en [esta página](#) el mecanismo que une las dos direcciones.

La itestación

Este protocolo no fue pensado solo para redes ethernet y protocolo IP, por lo tanto un paquete ARP contiene varios campos que especifican de cuales direcciones se está hablando:

- *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 es 4).
- *Operation*: representa el mando; en nuestro caso veremos ARP Request y ARP Replay.

A estos sigue un campo de datos de longitud variable (que depende de la itestación), que para las operaciones nombradas tendrá 4 direcciones, como indicado por la siguiente ilustración:

Hardware Type		Protocol Type
Hw Length	Prot Length	Operation
Source MAC Address [0:3]		
Source MAC Address [4:5]		Source IP Address [0:1]
Source IP Address [2:3]		Target MAC Address [0:1]
Target MAC Address [2:5]		
Target IP Address		

ARP Request

El paquete que pide la dirección MAC contiene, la intestación, las direcciones IP y MAC del remitente, y como *Target IP Address*, la dirección de la cual se quiere conocer la 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 nuestro IP, en otro lugar ignoramos la petición.

ARP Reply

El paquete de respuesta es muy parecido al que hemos visto ahora: es suficiente indicar en el campo *Operation* que se trata de una respuesta, y llenar los varios 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 está definido así:

```
typedef struct {
    u16      hardware;
    u16      protocol;
    u8       MacLen;
    u8       ProtocolLen;
```

```
    u16      operation;  
    MACAddr  SourceMAC;  
    IPAddr   SourceIP;  
    MACAddr  TargetMAC;  
    IPAddr   TargetIP;  
} ARPPacket;
```