Universidade de Aveiro

Departamentos de Física e Mecânica 2006

PROJECTO DE MONITORIZAÇÃO E ELECTRO ESTIMULAÇÃO

Relatório 2: Transmissão através do protocolo ethernet

Eduardo Durana

Aveiro, 30 de Novembro de 2006

Índice

Objectivo

Este relatório é sobre a execução de um programa exemplo para transmitir dados por ethernet através do controlador ENC28J60 e do micro controlador PIC18F2550 ambos da microchip.

Breve introdução

O PIC comunica com o controlador ethernet (ENC28J60) por SPI (Serial Peripheral Interface), recorrendo a comandos definidos no datasheet do ENC28J60.

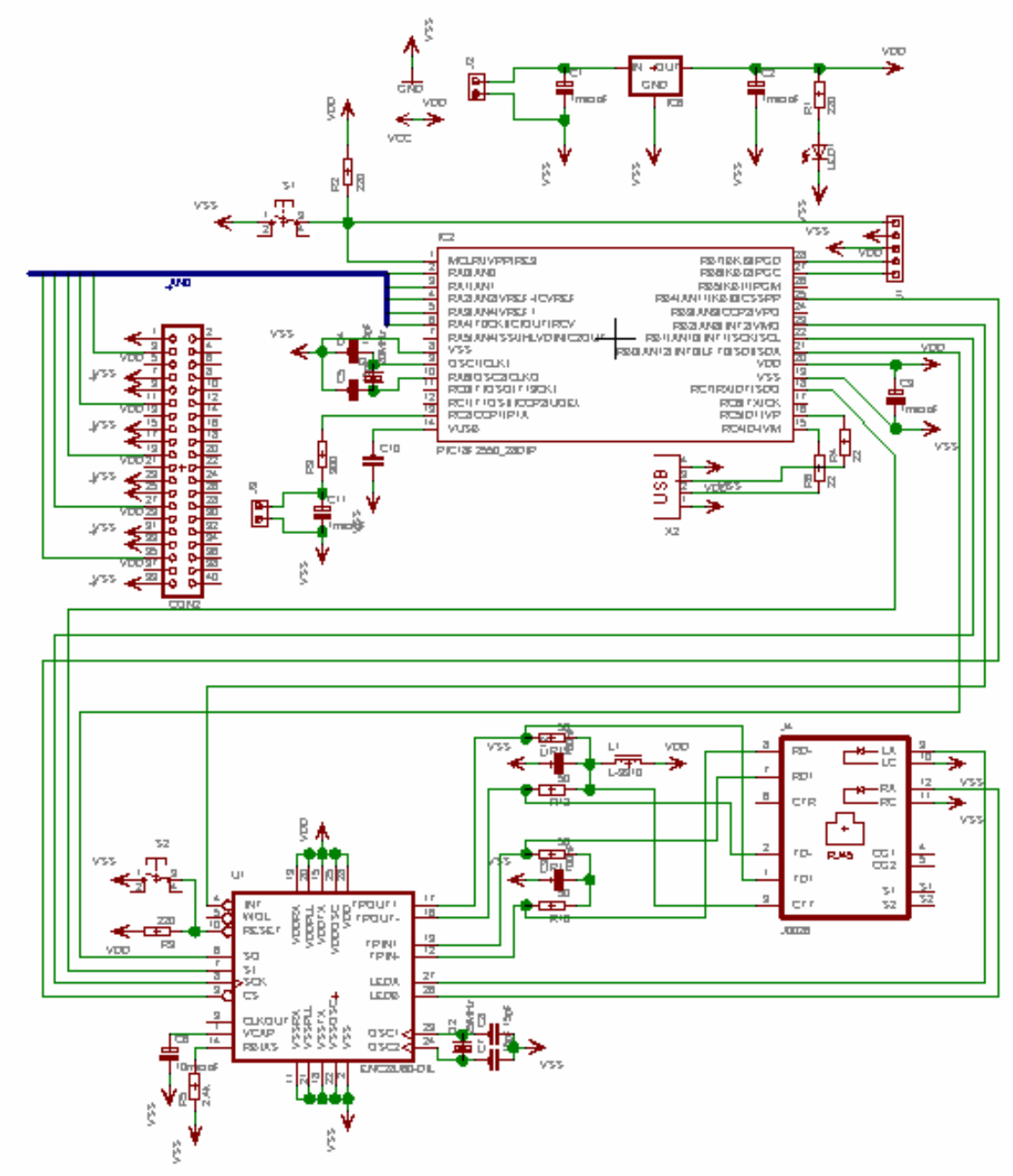
A memória do ENC28J60 encontra-se dividida em três tipos:

- Registos de controlo (constituindo por 4 bancos de 32 registos cada);
    - Ethernet (ETH);
    - Médium Access control (MAC);
    - Media independent interface management (MIIM);
- Ethernet buffer
    - 8kbytes de memória;
    - Necessita de ser definido espaço para transmissão e para recepção;
- Registos PHY (constituindo 32 endereços de registos de 2bytes);
    - Loopback;
    - Half/full duplex;
    - Codificação/ descodificação da informação na interface de pares entrelaçados;
    - LED.

Constituição do pacote ethernet;

Campo	Tamanho	Função
Preambulo	7bytes	Sincronização
Inicio de frame	1byte	Delimitação
MAC address de destino	6bytes	Identificação destinatário
Mac address de origem	6bytes	Identificação remetente
Tipo/tamanho	2bytes	Informação protocolar
Dados	46 a 1500bytes	Informação dados
FCS	4bytes	Verificação de erros

Esquema



Rotina

Este programa exemplo pretende enviar um conjunto de dados simples para teste, através do protocolo ethernet sempre que o utilizador pressiona um interruptor (com um LED indicador).

Esta é a rotina que é executada ciclicamente:

. . .

```
while(1)
      {
            if (PORTAbits.RA0 == 1)
            {
                  PORTAbits.RA1=1;

                  if (prev_RA0 == 0)
                  {
                        send=1;     //caso o botão esteja pressionado e
                                    //o seu estado anterior seja não
                                    //pressionado, a variável 'send'
                                    //assume o valor '1'.
                  }
                  else
                  {
                        send=0;
                  }
                  prev_RA0 = 1;
            }
            else if (PORTAbits.RA0 == 0)
            {
                  PORTAbits.RA1=0;
                  prev_RA0 = 0;
            }

            if (send==1)      //caso a variável 'send' tenha valor '1'
            {
                              //corre a instrução para transmitir.

//código para transmitir por ethernet

                        send=0;
                  }

            }
...
```

Em que RA0 é o pino de entrada do sinal vindo do interruptor ('1' quando pressionado) e RA1 o pino de saída para o LED que irá assinalar o estado pressionado do interruptor.

Configuração do microcontrolador

Os portos do PIC foram configurados da seguinte forma:

```
TRISA = 0b00100001;           //RA0 e RA5/SS  - entrada os outros pinos
                              //são saída
TRISB = 0b00000101;     //SDI e INT2 são entradas SCK (RB1) e RB4
                        //[escolhido para Chip Select (CS)] são saídas
TRISC = 0b00000000;     //o TRISC<7> tem de ser posto a '0' porque o
                        //SDO é uma saída
```

O pino RA5/SS (slave input) não é utilizado neste programa, mas como é um pino utilizado pelo módulo SPI, penso ser conveniente configura-lo como entrada, não vá haver conflitos depois de activar o módulo SPI. A comunicação SPI é feita entre um 'master' e um 'slave' de cada vez. Neste caso o PIC vai comunicar como 'master', razão pela qual o pino SS não é utilizado (caso o PIC fosse 'slave' quando o seu 'master' colocasse este pino a '0' estariam em comunicação, até que o seu 'master' voltasse a colocar este pino a '1').

Os pinos SDI e SDO são respectivamente para ‘data input’ e ‘data output’ do módulo SPII. Escolhi utilizar o pino INT2 para as interrupções SPI (que não são utilizadas nesta fase mas serão posteriormente para melhorar o desempenho do programa) e o pino RB4 como ‘chip select’ para poder seleccionar qual o periférico para comunicar por SPI.

O pino SCK é a saída ‘serial clock’ cuja frequência é definida pelo PIC (‘master’) e será definida mais á frente na configuração do módulo SPI. A razão de existir deste pino é o facto de a comunicação SPI ser síncrona.

Uma vez que não vou utilizar entradas analógicas configuro todos os pinos como digitais, através da seguinte configuração do conversor analógico digital:

```
ADCON1=0b00001111;            //todos os pinos digitais
```

Configuração do SPI para a comunicação entre o PIC e o microcontrolador:

```
...
//configuração do SPI
//vem especificado que o ENC28J60 suporta apenas o SPI modo 0,0
//(que implica: CKE=1;CKP=0)

SSPCON1bits.SSPEN = 0;  //reset

SSPSTATbits.SMP = 0;    //dados de input 'sampled' no meio do tempo de
                        //dados de output. Na duvida optei por colocar
                        //único estado permitido para o modo de
                        //funcionamento ‘slave’.
SSPSTATbits.CKE = 1;    //a transmissão ocorre na transição do estado
                        //do 'clock' de activo para inactivo

SSPCON1bits.CKP = 0;    //o estado 'idle' para o 'clock' é um nível
                        //'low'

//estes quatro bits configuram o 'Synchronous Serial Port Mode' como
//'master' com 'clock=Fosc/16'
SSPCON1bits.SSPM3 = 0;
SSPCON1bits.SSPM2 = 0;
SSPCON1bits.SSPM1 = 0;
SSPCON1bits.SSPM0 = 1;

SSPCON1bits.SSPOV = 0;  //em modo 'master' o SSPOV não é activado.

ChipSelect=1;           //para entretanto não haver comunicação por SPI
SSPCON1bits.SSPEN = 1;  //activa o serial port e configura os pinos
                        //SCK, SDO, SDI e SS como pinos de serial port
...
```

Configuração do controlador ethernet

A primeira instrução sobre o ENC28J60 é no sentido de auscultar quando o ‘clock’ estabiliza com os parâmetros normais de funcionamento.

```
//esperar que o ENC28J60 Fique pronto para trabalhar
```

```
//espera que o LSb do ESTAT transferido para o SPItemp fique a '1'
//CLKRDY
      while(!LerRegETH((byte)ESTAT) & 0b00000001);
```

nota: para aceder a este registo (ESTAT), assim como aos registos EIE, EIR, ECON1 e ECON2, não existe a necessidade de mudar de banco, uma vez que estes registos existem nos quatro bancos de registos de controlo do ENC28J60 com o mesmo endereço.

A função LerRegETH serve para ler a informação contida em registos ethernet, i.e. iniciados por 'E' e está definida da seguinte forma:

Ler registo ethernet

```
byte LerRegETH(byte address)
{
      byte r;
      ChipSelect=0;     //coloca o CS do ENC28J60 a '0' para que este
                        //fique à escuta
//    PIR1bits.SSPIF=0;
      SSPBUF = RCR | address;       //opcode para ler registo de
                                    //controlo seguido do endereço
      while(!SSPSTATbits.BF);       //espera que o SSPBUF seja
                                    //transmitido
      r = SSPBUF;                   //atribui à variável r o valor
                                //contido em SSPBUF recebido (dummy)
//    PIR1bits.SSPIF=0;
      SSPBUF = 0;               //envio de um byte 'dummy' para poder
                                //receber em seguida
while(!SSPSTATbits.BF);         //espera que o byte 'dummy' seja
                                //transmitido
      r = SSPBUF;               //atribui à variável r o valor contido em
                                //SSPBUF recebido
//    PIR1bits.SSPIF=0;
      ChipSelect=1;             //coloca o CS do ENC28J60 a '1' para
                                //fechar a comunicação
      return r;
}
```

Onde o argumento é o endereço do registo cujo valor se pretende ler e que irá ser devolvido.

A partir deste momento o controlador está pronto a funcionar em conformidade com as especificações tendo o oscilador estabilizado.

Configuração da memória de 'buffer' para recepção

A instrução de inicialização seguinte é a selecção da memória de buffer que irá corresponder à recepção. Para tal tem de se atribuir o endereço de início e de fim aos registos ERXST e ERXND respectivamente, que se encontram no banco0.

```
//programar a memória de buffer para recepção (start: 0h0000,
// stop: 0h0fff). Estes são escolhidos pelo programador
//definir inicio da memória de recepção

//ATENSÃO: este procedimento tem de ser efectuado quando o bit de
permissão de recepção (RXEN do reg.ECON1) está a '0'.
```

```
BFCReg((byte)ECON1,0b00000100);


BankSel(0);       //mudar para o banco0
                  //onde estão os registos a operar de seguida

//atribuir o valor '0x0000' ao registo ERXST. no datasheet recomendam
//que este valor seja par.

EscreverReg((byte)ERXSTH,0x00);
EscreverReg((byte)ERXSTL,0x00);

//definir fim da memória de recepção
//atribuindo o valor '0x0fff' ao registo ERXND

EscreverReg((byte)ERXNDH,0x0f);
EscreverReg((byte)ERXNDL,0xff);
```

Foram utilizadas as funções ‘BankSel’ para mudar para o banco correcto e ‘EscreverReg’ que permite escrever um valor em qualquer registo de controlo, com excepção dos registos PHY (que têm de ser acedidos indirectamente).

Selecção de banco

A função ‘BankSel’ está definida da seguinte forma:

```
void BankSel(byte banco)
{
//função que serve para mudar de banco. (colocar um valor entre 0 e 3)
BFCReg((byte)ECON1, 3);       //apaga os bits BSEL(1:0) que são os LSb
            //do registo ECON1 responsáveis pela selecção do banco

BFSReg((byte)ECON1, banco);   //coloca a '1' os bits correspondentes ao
            //banco pretendido
}
```

Que por sua vez utiliza as funções de ‘bit field clear’ e ‘bit field set’, mostradas de seguida:

Apagar e elevar a ‘1’ bits de um registo

```
void BFCReg(byte address, byte dados)
{
//o argumento address é o endereço do registo a ser operado e os bits
//a '1' do byte dados seram apagados
      byte dummy;
      ChipSelect=0;     //coloca o CS do ENC28J60 a '0' para que este
                        //fique à escuta
//    PIR1bits.SSPIF=0;
      SSPBUF = BFC | address;       //opcode para bit field clear
                                    //seguido do endereço

      while(!SSPSTATbits.BF);       //espera que o SSPBUF receba (BF
                                    //fica a '1')
      dummy = SSPBUF;         //lê o dummy byte, para o BF voltar a '0'
//    PIR1bits.SSPIF=0;
      SSPBUF = dados;         //envia os bits a apagar (os que
                              //estiverem a '1')
      while(!SSPSTATbits.BF);
      dummy = SSPBUF;
//    PIR1bits.SSPIF=0;
```

```
ChipSelect=1;           //coloca o CS do ENC28J60 a '1' para fechar a
                        //comunicação

}
```

A função de BFSReg é análoga com a diferença do ‘opcode’ e da acção sobre os bits a ‘1’ enviados no segundo argumento, que em vez de serem apagados são agora elevados.

Escrever num registo de controlo

A função ‘EscreverReg’ vem definida da seguinte forma:

```
void EscreverReg(byte address, byte dados)
{
      byte dummy;
      ChipSelect=0;                              //coloca o CS do
ENC28J60 a '0' para que este fique à escuta
//    PIR1bits.SSPIF=0;
      SSPBUF = WCR | address;       //opcode para escrever no registo
                              //de controlo seguido do endereço
      while(!SSPSTATbits.BF);       //espera que o SSPBUF seja
                  //transmitido verificando que BF fique a '1'
      dummy = SSPBUF;                            //lê o byte dummy
//    PIR1bits.SSPIF=0;
      SSPBUF = dados;               //envio de dados
      while(!SSPSTATbits.BF);       //espera que o valor seja
                                    //transmitido
      dummy = SSPBUF;               //lê o byte dummy
//    PIR1bits.SSPIF=0;
      ChipSelect=1;                 //coloca o CS do ENC28J60 a '1'
                                    //para fechar a comunicação

}
```

Onde o primeiro argumento é o endereço do registo onde se pretende escrever o valor a ser introduzido no segundo argumento.

Configuração de registos de controlo ethernet

```
...
//registos de controlo ethernet
      //CSUMEN é activado permitindo que a DMA calcule checksums
      //activa RXEN, permite escrever no buffer de recepção pacotes
consoante a configuração dos filtros.
      //ATENSÃO: este bit ECON1(2) só deve ser activado após se ter
determibado a memória de bufer de recepção.
BFSReg((byte)ECON1,0b00010100);

      //activa o bit AUTOINC permitindo um incremento automatico nos
ponteiros ERDPT e EWRPT
      //durante a leitura e escrita de dados no buffer
respectivamente.
BFSReg((byte)ECON2,0b10000000);

//configuração do registo do filtro de recepção
      //descarta pacotes que não tenham o MAC address do ENC28J60 como
endereço de destino
      //descarta pacotes que não cumpram todos os filtros activados
      //descarta pacotes com CRC inválido
BFSReg((byte)ERXFCON,0b11100000);
```

```
...
```

Configuração de registos MAC

O próximo passo é a configuração dos registos MAC (medium access controller).

```
      //mudar para banco2

BankSel(2);

//para activar a recepção de frames pelo MAC é preciso colocar a '1' o
//MACON1.MARXEN='1'

BFSReg((byte)MACON1,0b00000001);

//configurar o byte MACON3 para ter Padding (802.3 exige packets
//superiores a 64bytes) e CRC automáticos e frame length reporting
//status (aconselhado).
//no protocolo ethernet para que o CRC seja criado correctamente, o
//padding é até completar 60bytes (minimo de acordo com 802.3 64bytes
//a contar com o proprio CRC)

EscreverReg((byte)MACON3,0b10110010);


//activar o bit DEFER do registo MACON4 de acordo com as
//especificações IEEE 802.3

BFSReg((byte)MACON4,0b01000000);


//colocar nos registos MAMXFL o comprimento maximo permitido para os
//frames a ser transmitidos ou recebidos
//valor escolhido 1518 (05EEh)

EscreverReg((byte)MAMXFLH,0x05);
EscreverReg((byte)MAMXFLL,0xee);

//os valores introduzidos no 3 registos seguintes estão de acordo com
//o aconselhado no datasheet
//colocar o valor 12h no byte MABBIPG por ser half duplex

EscreverReg((byte)MABBIPG,0x12);

//colocar o valor 12h no byte MAIPGL por ser half duplex

EscreverReg((byte)MAIPGL,0x12);

//colocar o valor 0Ch no byte MAIPGH por ser half duplex

EscreverReg((byte)MAIPGH,0x0c);

//Configuração dos registos do source MAC address
//utilizados apenas na recepção

BankSel(3);
EscreverReg((byte)MAADR1,MAC1);
EscreverReg((byte)MAADR2,MAC2);
EscreverReg((byte)MAADR3,MAC3);
EscreverReg((byte)MAADR4,MAC4);
EscreverReg((byte)MAADR5,MAC5);
EscreverReg((byte)MAADR6,MAC6);
```

Configuração dos registos PHY

Ao contrário dos registos de controlo ETH,MAC e MII ou da memória de buffer, os registos PHY não são directamente acessíveis por SPI. Os registos têm de ser acedidos por meio de um conjunto especial de registos de controlo MAC, que implementam 'media independent interface managment' (MIIM).

Para escrever num registo PHY é necessário escrever o seu endereço no registo MIREGADR e colocar o valor em MIWRL (em primeiro) e só depois em MIWRH, isto porque logo após se escrever neste registo se inicia automaticamente a transacção MII. Por fim basta esperar que o bit BUSY, bit 0 do registo MISTAT volte a '0' para ter a certeza que a operação está completa.

```
//o bit PHCON1<PDPXMD> tem de estar a '0' para estar de acordo com o
//funcionamento em half duplex
EscreverRegPHY((byte)PHCON1, 0b00000000, 0b00000000);

//o bit PHCON2<HDLDIS> tem de estar a '1' em funcionamento em half
//duplex para prevenir LoopBack
EscreverRegPHY((byte)PHCON2, 0b00000000, 0b00000001);

//configuração dos LEDs
//LEDA (verde) mostra Link status e LEDB (amarelo) mostra a actividade
//de transmissão
EscreverRegPHY((byte)PHLCON, 0b00010010, 0b00110100);
```

Escrever num registo PHY

Para escrever no registo PHY foi utilizada a função ‘EscreverRegPHY’, mostrada de seguida.

```
void EscreverRegPHY(byte address, byte low, byte high)

{
BankSel(2);                    //mudar para o banco2, onde se encontra o
registo MIREGADR
EscreverReg((byte)MIREGADR, address);
EscreverReg((byte)MIWRL, low);
EscreverReg((byte)MIWRH, high);

BankSel(3);                    //mudar para o banco3, onde se encontra o
registo MISTAT

while (LerRegMAC((byte)MISTAT) & 0b00000001);

}
```

Esta função por sua vez utiliza a função ‘LerRegMAC’ para avaliar o bit BUSY, por este bit estar num registo MAC. A única diferença em relação a ler um registo ethernet é que o ENC28J60, antes de enviar a leitura que se pretende envia primeiro um byte ‘dummy’ sem informação relevante, mas que tem de ser lido antes de se obter o valor que está no registo desejado.

Função para transmitir

Esta função tem as seguintes variáveis locais:

```
int txBufLength;        //para guardar a dimenção do frame a ser
                        //escrito no buffer de transmissão
int TXST = 0x1a00;      //endereço dentro da memória de buffer
                        //para o inicio da memória de transmissão
int TXND;               //onde vai ser colocado o valor do endereço
                        //do fim de memoria de transmissão
byte TXNDL;             //LSB do TXND
byte TXNDH;             //MSB do TXND

      BankSel(0);
//Programar o ponteiro ETXST
      EscreverReg((byte)ETXSTL,(byte)((TXST<<8)>>8));
      EscreverReg((byte)ETXSTH,(byte)(TXST>>8));
```

Programar o início da memória de buffer de transmissão

É necessário indicar onde se inicia o ponteiro de escrita configurando EWRPT. O valor a atribuir será o início da memória de transmissão.

```
//Programar o ponteiro EWRPT
      EscreverReg((byte)EWRPTL,(byte)((TXST<<8)>>8));
      EscreverReg((byte)EWRPTH,(byte)(TXST>>8));
```

Escrita do pacote a enviar

Byte de controlo

```
//byte de controlo
      EscreverBuffer(controlo);
```

Onde controlo está definido com o valor ‘0b00000110’. O byte de controlo permite que se alterem algumas configurações de transmissão em relação ao que está definido no registo ‘MACON3’. Neste caso o byte de controlo está configurado para não existir alteração.

Escrever um byte na memória de ‘buffer’

Foi utilizada a função ‘EscreverBuffer’ para escrever um byte na memória de ‘buffer’ que é definida da seguinte forma:

```
            byte dummy;
            ChipSelect=0;                           //coloca o CS do
ENC28J60 a '0' para que este fique à escuta
//          PIR1bits.SSPIF=0;
            SSPBUF = WBM;                           //opcode para bit
field clear seguido argumento correspondente
```

```
                  while(!SSPSTATbits.BF);       //espera que o SSPBUF
receba (BF fica a '1')
            dummy = SSPBUF;                           //lê o dummy
byte, para o BF voltar a '0'
//          PIR1bits.SSPIF=0;
            SSPBUF = dados;                           //envia o byte a
ser escrito
                  while(!SSPSTATbits.BF);
            dummy = SSPBUF;
//          PIR1bits.SSPIF=0;

            ChipSelect=1;                             //coloca o CS do
ENC28J60 a '1' para fechar a comunicação
```

MAC address de destino

```
//MAC address de destino (braodcast: FF-FF-FF-FF-FF-FF)

      EscreverArrayBuffer(destMACaddr,sizeof(destMACaddr));
```

Nesta fase, o ‘MAC address’ de destino é FF-FF-FF-FF-FF-FF que corresponde a um endereço de ‘broadcast’, que se destina a todos os MACs ligados à mesma rede.

Escrever um ‘array’ na memória de buffer

A função ‘EscreverArrayBuffer’ foi utilizada para automaticamente escrever um ‘array’ na memória de buffer. O primeiro argumento é um ponteiro para o ‘array’, o segundo argumento é o tamanho do referido ‘array’ e está escrita da forma a seguir descrita:

```
void EscreverArrayBuffer(byte *val, unsigned int len)
{
            byte dummy;
                  ChipSelect=0;     //coloca o CS do ENC28J60 a '0'
                                    //para que este fique à escuta
//          PIR1bits.SSPIF=0;
            SSPBUF = WBM;           //opcode para bit field clear
                                    //seguido argumento correspondente
            while(!SSPSTATbits.BF);       //espera que o SSPBUF receba
                                          //(BF fica a '1')
            dummy = SSPBUF;               //lê o dummy byte, para o BF
                                          //voltar a '0'
//          PIR1bits.SSPIF=0;

// envio de dados
      while(len)  //enquanto existirem bites a enviar, corre o ciclo
      {
            SSPBUF = *val;    // inicia o envio do bite
            val++;            // incremento do ponteiro que aponta para
                              //o byte a enviar
            len--;            // decremento do numero de bytes a enviar
                  while(!SSPSTATbits.BF); //espera que o SSPBUF
                                          //receba (BF fica a '1')
            dummy = SSPBUF;
//          PIR1bits.SSPIF=0;
      };

      ChipSelect=1;     //coloca o CS a '1' fecha a comunicação
}
```

MAC address de origem

```
//MAC address da fonte

      EscreverArrayBuffer(sorceMACaddr,sizeof(sorceMACaddr));
```

Aqui o 'array' sourceMACaddr está configurado com o valor de defeito da placa de demonstração da microchip: 00-04-a3-00-00-00, em hexadecimal.

Campo Type/Length

```
//type length (2 bytes).
//quando length, deve conter o numero de bytes (de dados) no campo de
//dados, sem contar com o padding
//quando type, deve ter um valor de acordo com o protocolo. exemplo:
//protocolo IP - type 0x0800

      EscreverArrayBuffer(typeLength,sizeof(typeLength));
```

Campo de dados

```
      EscreverArrayBuffer(dados,sizeof(dados));
```

Programar o fim da memória de buffer de transmissão

Antes de enviar o pacote é necessário definir o fim da memória de buffer de transmissão através da configuração do ponteiro ETXND.

```
//Programar o ponteiro ETXND
txBufLength = sizeof(destMACaddr)+sizeof(sorceMACaddr)+
              sizeof(typeLength)+sizeof(dados);
      BankSel(0);

TXND = TXST+txBufLength;

      EscreverReg((byte)ETXNDL,(byte)((TXND<<8)>>8));
      EscreverReg((byte)ETXNDH,(byte)(TXST>>8));
```

Transmitir o pacote

Para se iniciar activa-se o bit ECON1.TXRTS

```
//iniciar o processo de transmissão

      BFSReg((byte)ECON1, 0b00001000);
      while(LerRegETH((byte)ECON1) & 0b00001000);
```

Devendo-se por fim aguardar que o mesmo bit accionado para dar inicio á transmissão fique a ‘0’, assinalando o final do processo de envio. Período durante o qual nenhum dos registos associados á transmissão de pacotes por ethernet deve ser alterado.