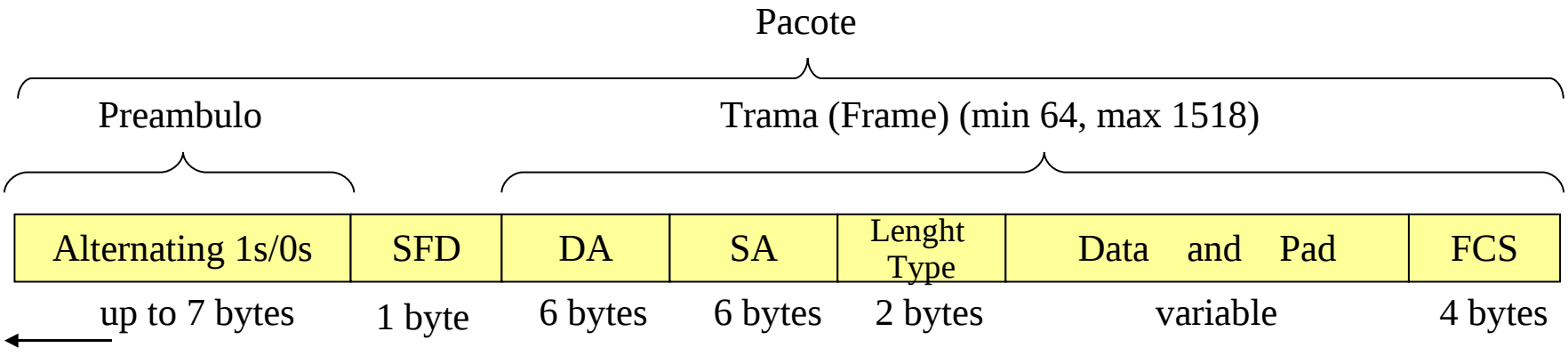
Sistemas Embebidos II

Ethernet

Ezequiel Conde

Formato de um pacote Ethernet



Preambulo (7 bytes 101010...)

SFD - Delimitador de inicio de trama (- Start Frame Delimiter) (10101011)

DA - Endereços de destino.

SA – Endereço de origem.

Lenght/Type - Dimensão ou tipo da trama.

Data – Dados.

Pad – Enchimento para garantir a dimensão mínima da trama.

FCS – Sequência de verificação de erro (Frame Check Sequence) (CRC).

Controlador Ethernet - Transmissão

- O utilizador fornece por cada trama:
  - Endereço fonte e destino;
  - Dimensão / tipo;
  - Dados.
- Forma automaticamente o pacote.
- Detecta colisões e executa retransmissões.
- Erros de transmissão assinaláveis:
  - Pacote muito grande;
  - Excesso de colisões, colisões tardias;
  - Falta de dados para transmitir (Tx Underrun)

Controlador Ethernet - Recepção

- Depois de detectado o delimitador inicial é assumido o início de uma trama.
- O endereço de destino é verificado segundo o filtro de endereços.
- A trama é recebida se verificar o endereço.
- O CRC é verificado se estiver errado a trama é descartada.
- Erros de recepção assinaláveis:
  - Erro de CRC;
  - Dimensão inválida (< 64 ou > 1518);
  - Dados para além do fim da trama (Dribble Nibble)

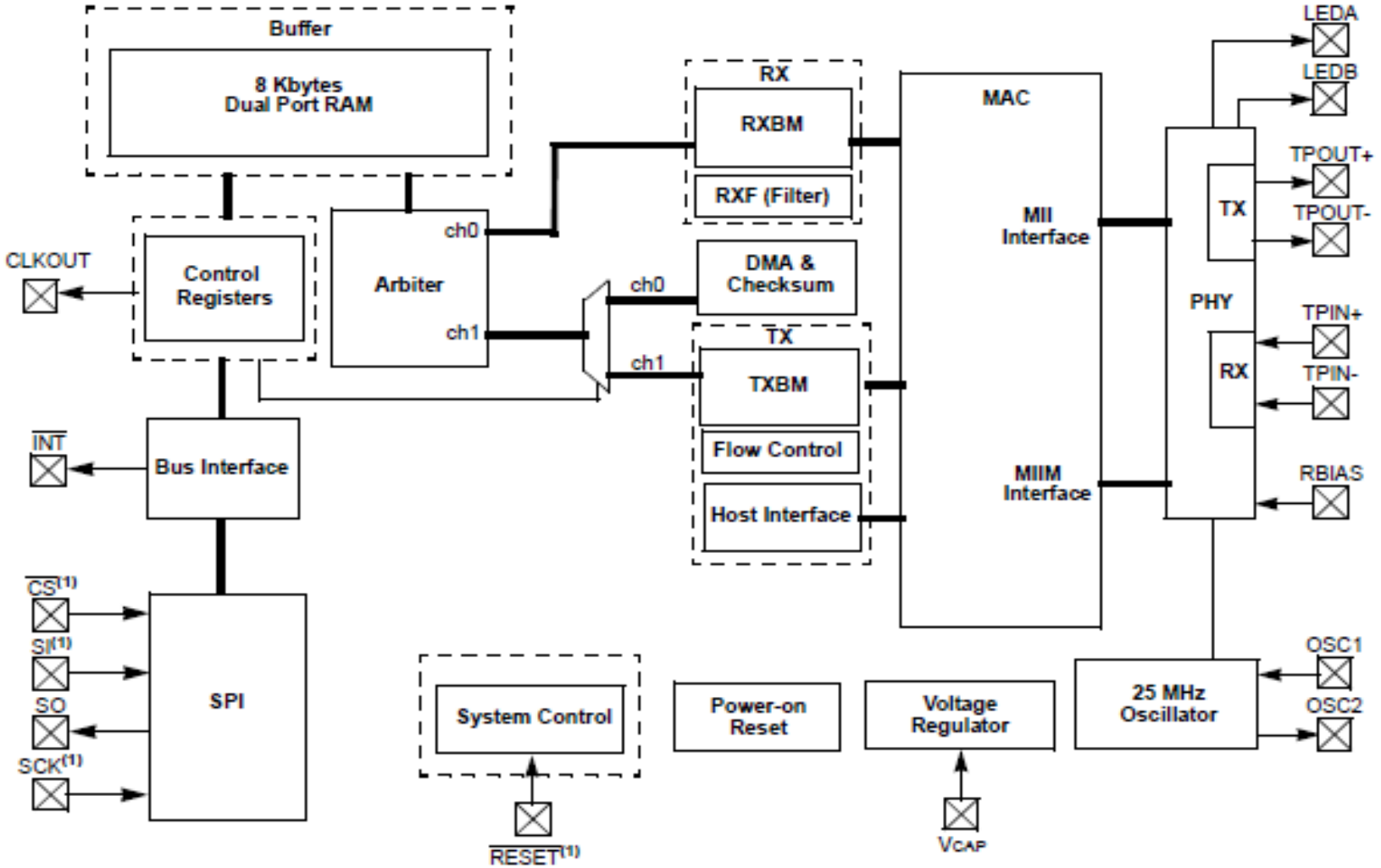
Gestão de acesso ao meio

- A topologia Ethernet baseia-se num único meio de comunicação partilhado por todos os nós.
- Dois nós podem iniciar a transmissão simultaneamente originado uma colisão.
- O controlador procura evitar colisões e recupera da sua ocorrência.
- Estratégias para evitar colisões:
  - Simple deferral: depois de transmitida uma trama é feita uma espera 9,6 uS (Inter Packet Gap).
  - Two-part deferral: O periodo IPG é dividido em duas partes 6,4uS e 3,2 uS. Se durante a primeira parte o meio for ocupado o processo é reiniciado.
- Resolução de colisões:
  - Colisões normais (detectadas antes do bit 512): É transmitida a sequência de interrupção (Jam sequence – 32 bits a 0). Ao fim de 16 tentativas desiste.
  - Colisões tardias: a trama é descartada.

Retransmissões

- Após uma colisão o transmissor aguarda um periodo antes de reiniciar nova tentativa.
- O periodo é obtido aleatóriamente e representa o número de slot times entre 0 e 2^K (um slot é o tempo de transmitir 512 bits).
- Standard backoff: K é um menor entre 10 e o número de tentativas.
- Modified backoff: K é 3 se o número de tentativas for inferior a 3, senão é igual ao standard. Este critério reduz a probabilidade de múltiplas colisões nas primeiras tentativas, mas aumenta o tempo de espera para retransmissão.

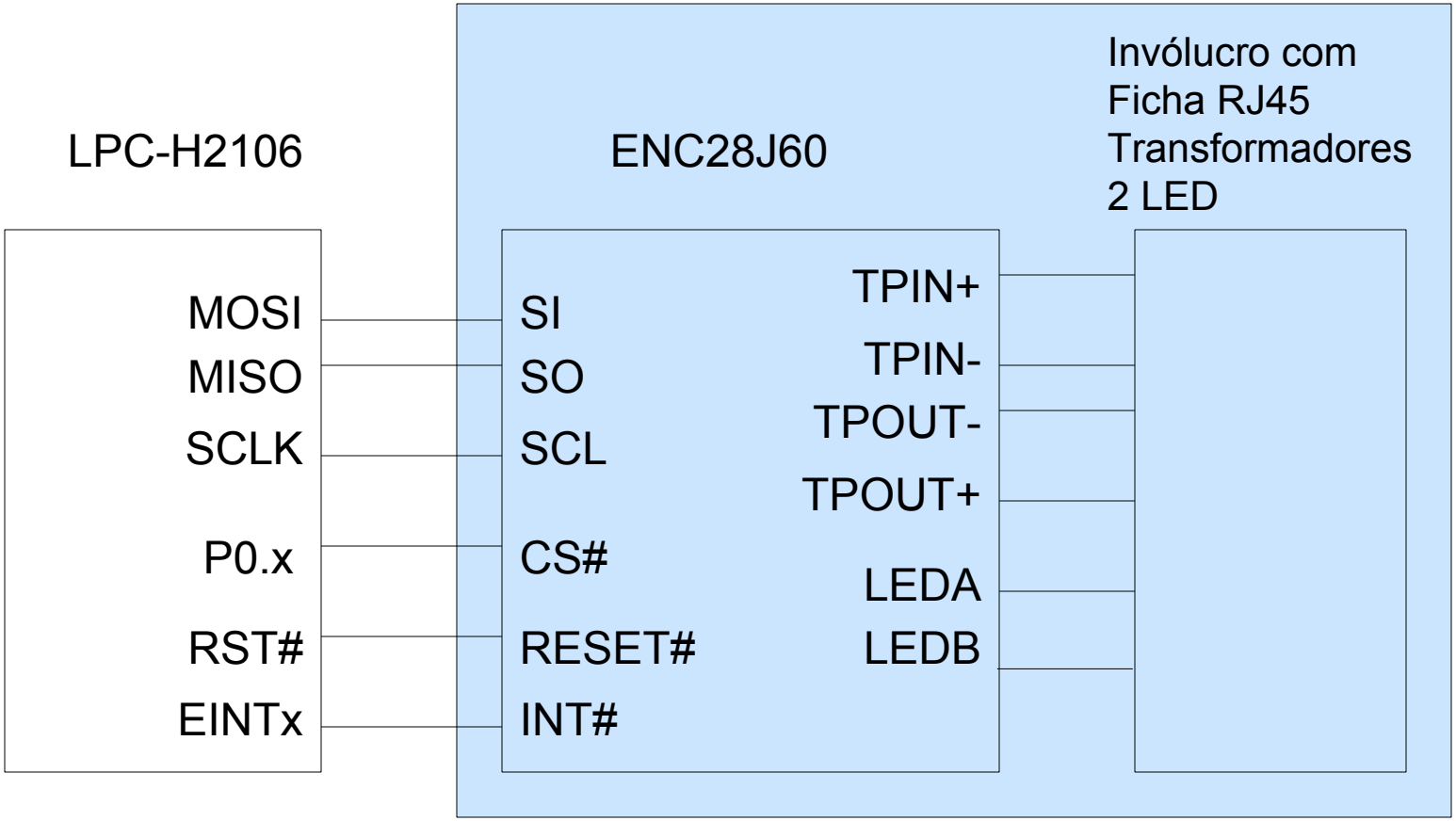
ENC28J60 - Diagrama de blocos



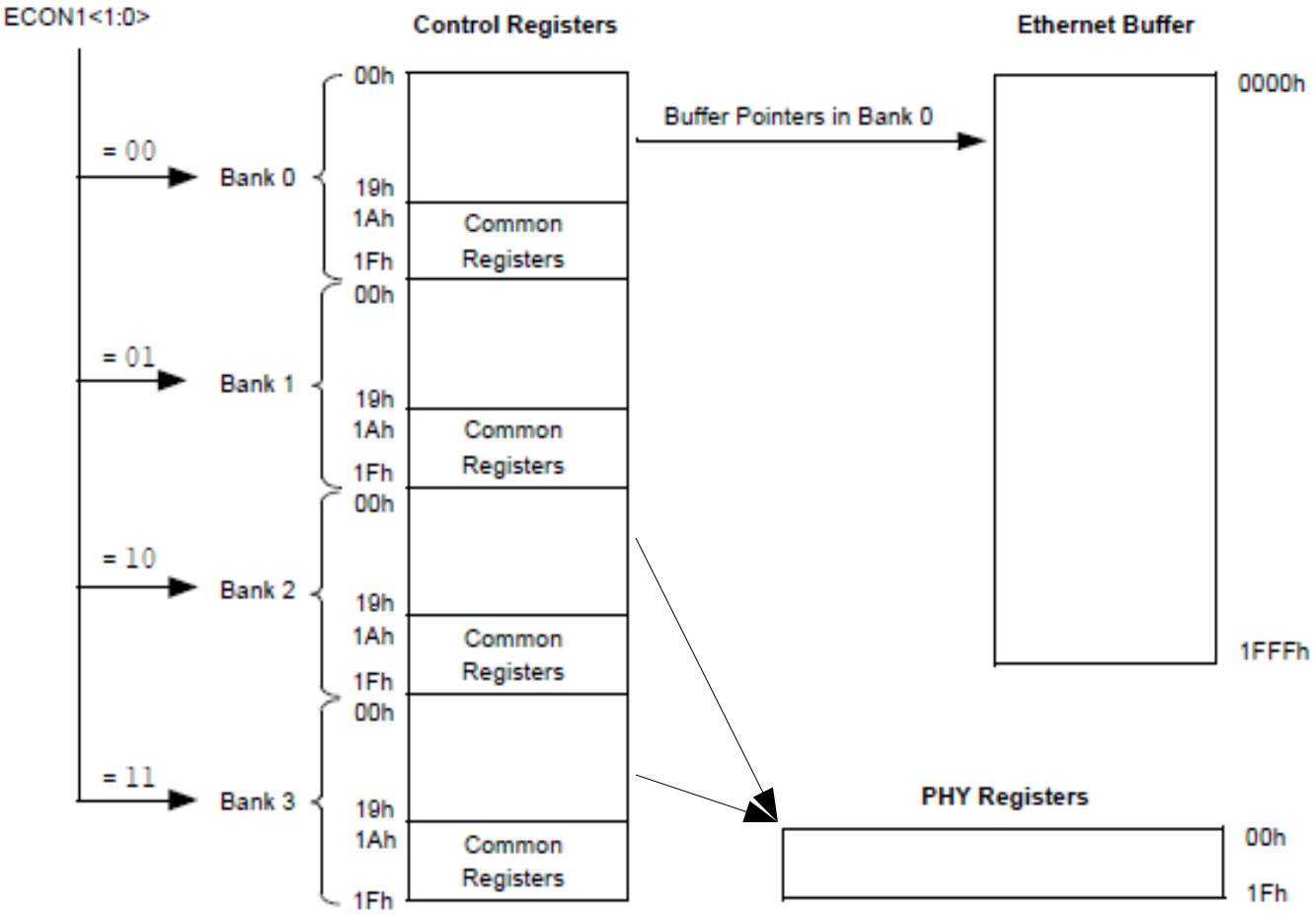
ENC28J60 - Características

- Interface SPI até 20 Mhz.
- Memória interna de 8K para transmissão e recepção.
- Integra MAC e PHY 10BASE-T.
- Half ou full-duplex.
- Geração e verificação de CRC.
- Retransmissão automática em caso de colisões.
- Modo Loopback.
- Filtro de recepção versátil.

ENC28J60 - H


LPC-H2106
ENC28J60
Invólucro com
Ficha RJ45
Transformadores
2 LED
MOSI
MISO
SCLK
P0.x
RST#
EINTx
SI
SO
SCL
CS#
RESET#
INT#
TPIN+
TPIN-
TPOUT-
TPOUT+
LEDA
LEDB

ENC28J60 – Memória interna



ENC28J60 – Registos de controlo

Bank 0 Address	Name
00h	ERDPTL
01h	ERDPTH
02h	EWRPTL
03h	EWRPTH
04h	ETXSTL
05h	ETXSTH
06h	ETXNDL
07h	ETXNDH
08h	ERXSTL
09h	ERXSTH
0Ah	ERXNDL
0Bh	ERXNDH
0Ch	ERXRDPTL
0Dh	ERXRDPTH
0Eh	ERXWRPTL
0Fh	ERXWRPTH
10h	EDMASTL
11h	EDMASTH
12h	EDMANDL
13h	EDMANDH
14h	EDMADSTL
15h	EDMADSTH
16h	EDMACSL
17h	EDMACSH
18h	—
19h	—
1Ah	Reserved
1Bh	EIE
1Ch	EIR
1Dh	ESTAT
1Eh	ECON2
1Fh	ECON1

Bank 1 Address	Name
00h	EHT0
01h	EHT1
02h	EHT2
03h	EHT3
04h	EHT4
05h	EHT5
06h	EHT6
07h	EHT7
08h	EPMM0
09h	EPMM1
0Ah	EPMM2
0Bh	EPMM3
0Ch	EPMM4
0Dh	EPMM5
0Eh	EPMM6
0Fh	EPMM7
10h	EPMCSL
11h	EPMCSH
12h	—
13h	—
14h	EPMOL
15h	EPMOH
16h	Reserved
17h	Reserved
18h	ERXFCON
19h	EPKTCNT
1Ah	Reserved
1Bh	EIE
1Ch	EIR
1Dh	ESTAT
1Eh	ECON2
1Fh	ECON1

Bank 2 Address	Name
00h	MACON1
01h	Reserved
02h	MACON3
03h	MACON4
04h	MABBIPG
05h	—
06h	MAIPGL
07h	MAIPGH
08h	MACLCON1
09h	MACLCON2
0Ah	MAMXFLL
0Bh	MAMXFLH
0Ch	Reserved
0Dh	Reserved
0Eh	Reserved
0Fh	—
10h	Reserved
11h	Reserved
12h	MICMD
13h	—
14h	MIREGADR
15h	Reserved
16h	MIWRL
17h	MIWRH
18h	MIRDL
19h	MIRDH
1Ah	Reserved
1Bh	EIE
1Ch	EIR
1Dh	ESTAT
1Eh	ECON2
1Fh	ECON1

Bank 3 Address	Name
00h	MAADR5
01h	MAADR6
02h	MAADR3
03h	MAADR4
04h	MAADR1
05h	MAADR2
06h	EBSTSD
07h	EBSTCON
08h	EBSTCSL
09h	EBSTCSH
0Ah	MISTAT
0Bh	—
0Ch	—
0Dh	—
0Eh	—
0Fh	—
10h	—
11h	—
12h	EREVID
13h	—
14h	—
15h	ECOCON
16h	Reserved
17h	EFLOCON
18h	EPAUSL
19h	EPAUSH
1Ah	Reserved
1Bh	EIE
1Ch	EIR
1Dh	ESTAT
1Eh	ECON2
1Fh	ECON1

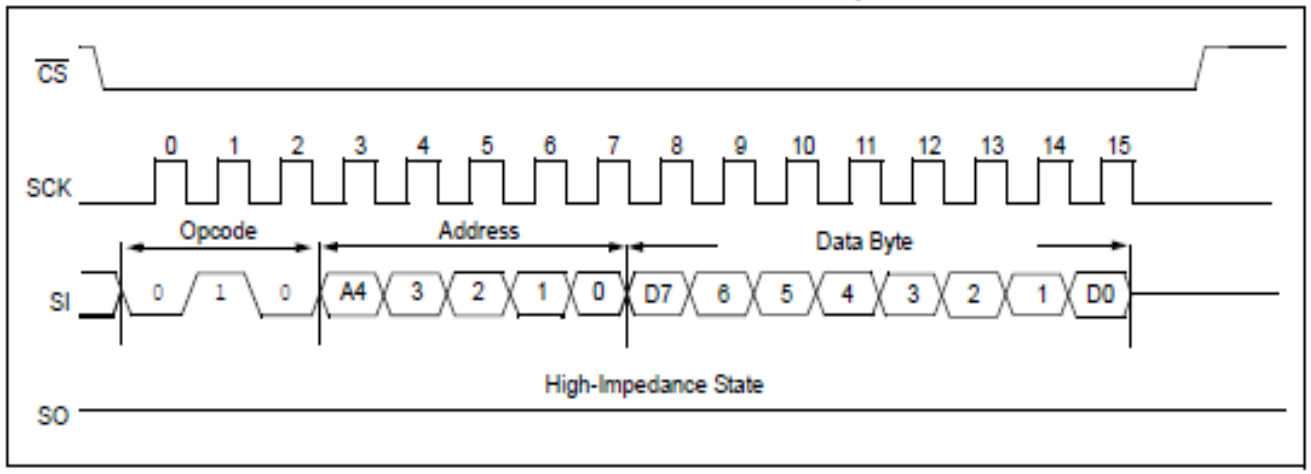
Operações básicas de input/output

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.

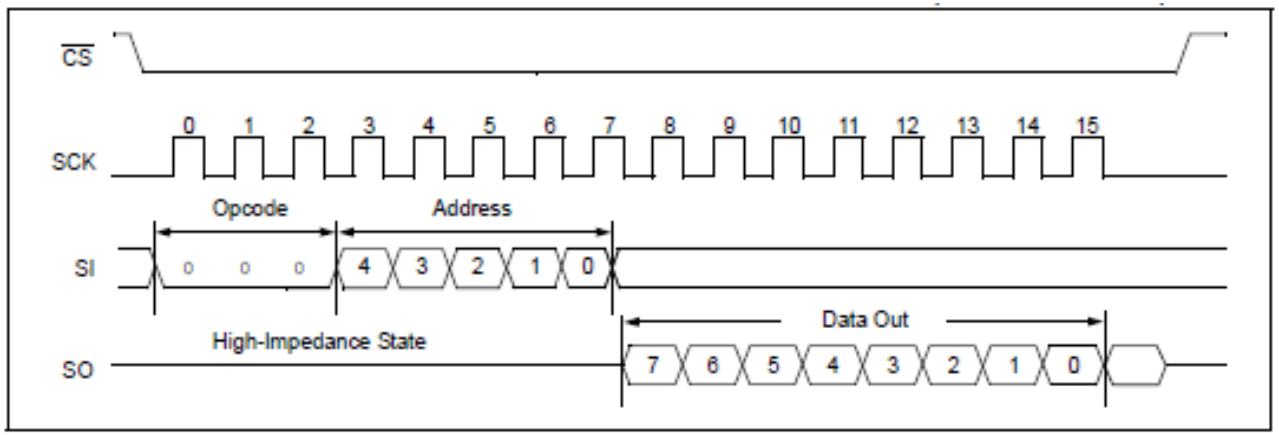
- As operações básicas de acesso ao controlador começam com o envio de um bytes indicando a operação.
- Segue-se uma sequência de zero ou mais bytes, num ou noutro sentido, dependendo da operação.

Operação básica de escrita em registo




```
static void write_control_register(int reg, U8 data) {
    U8 tx_buf[] = { 0x40 | (reg & ADDR_MASK), data };
    spi_transaction_begin(spi);
    spi_transaction_transfer(spi, sizeof(tx_buf), tx_buf, tx_buf);
    spi_transaction_end(spi);
}
```

Operação básica de leitura de registo




```
static U8 read_control_register(int reg) {
    int len = (reg & SPRD_MASK) ? 3 : 2;     /* dummy byte ? */
    U8 tx_buf[3] = { 0x00 | (reg & ADDR_MASK)};
    U8 rx_buf[3];
    spi_transaction_begin(spi);
    spi_transaction_transfer(spi , len, tx_buf, rx_buf);
    spi_transaction_end(spi);
    return rx_buf[len - 1];
}
```

(Na leitura de registos MAC e MII o controlador devolve dois bytes de dados. Os dados úteis são os do segundo byte.

Operações compostas de acesso a registos

REGISTER 3-1: ECON1: ETHERNET CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	BSEL1	BSEL0
bit 7							bit 0

- Os registos de controlo são cerca de 128 mas ocupam apenas 32 endereços.
- Os registos são divididos por 4 bancos de 32 registos. Antes de aceder a um registo é preciso seleccionar o banco a que ele pertence.

```
static void bank_select(int address) {
    U8 aux = read_control_register(ECON1);
    aux &= ~ECON1_BSEL;
    aux |= ((address >> 5) & ECON1_BSEL);
    write_control_register(ECON1, aux);
}

static U8 reg_read8(int addr) {
    bank_select(addr);
    return read_control_register(addr);
}

static U8 reg_write16(int addr, U16 data) {
    bank_select(addr);
    write_control_register(addr, data);
    write_control_register(addr + 1, data >> 8);
}
```

enc28j60.h

```
#ifndef ENC28J60_H
#define ENC28J60_H

/* Bank 0 registers */
#define ERDPTL      0x00
#define ERDPTH      0x01
...
#define ECON1       0x1F
#define ECON1_BSEL  (3 << 0)

/* Bank 1 registers */
#define EHT0        (0x00 | 0x20)
#define EHT1        (0x01 | 0x20)
...

/* Bank 2 registers */
#define MACON1      (0x00 | 0x40 | SPRD_MASK)
#define MACON3      (0x02 | 0x40 | SPRD_MASK)
...

/* Bank 3 registers */
#define MAADR5      (0x00 | 0x60 | SPRD_MASK)
#define MAADR6      (0x01 | 0x60 | SPRD_MASK)
...
#endif
```

Outras operações básicas

Colocar bits a zero

```
static void bit_field_clear(int reg, U8 data) {
    U8 tx_buf[] = { 0xa0 | (reg & ADDR_MASK), data };
    spi_transaction_begin(spi);
    spi_transaction_transfer(spi,  sizeof(tx_buf), tx_buf, tx_buf);
    spi_transaction_end(spi);
}
```

Colocar bits a um

```
static void bit_field_set(int reg, U8 data) {
    U8 tx_buf[] = { 0x80 | (reg & ADDR_MASK), data };
    spi_transaction_begin(spi);
    spi_transaction_transfer(spi,  sizeof(tx_buf), tx_buf, tx_buf);
    spi_transaction_end(spi);
}
```

Reset por software

```
static void system_reset() {
    U8 tx_buf[] = { 0xff };
    spi_transaction_begin(spi);
    spi_transaction_transfer(spi, sizeof(tx_buf), tx_buf, tx_buf);
    spi_transaction_end(spi);
}
```

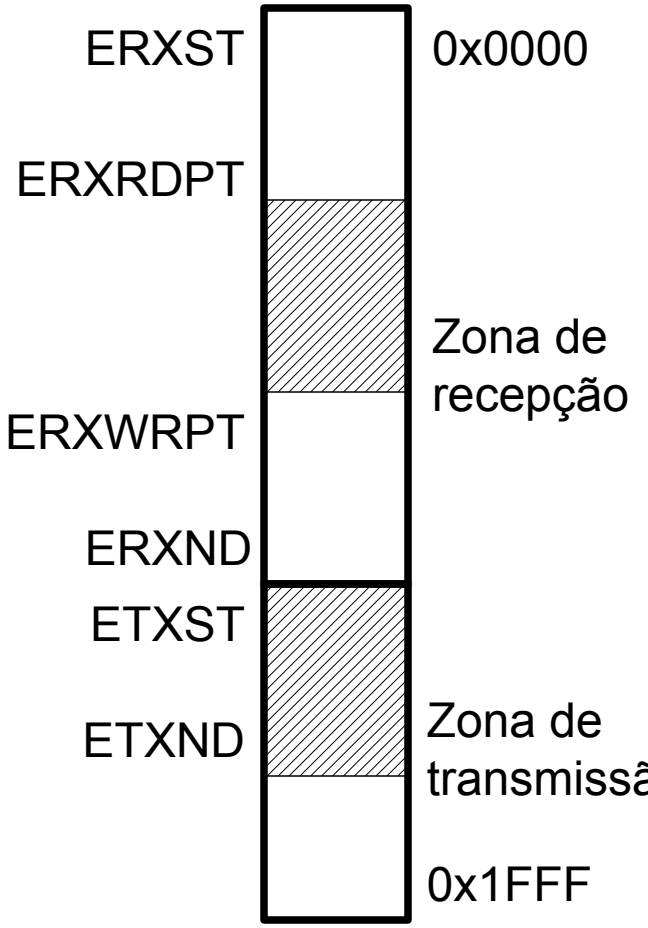
ENC28J60 – Memória de pacotes

Transmissão

- Cada trama a transmitir é previamente escrita na memória e assinalada pelos ponteiros ETXST e ETXND.
- Não é necessário haver mais do que uma trama na memória.

Recepção

- A zona de memória para recepção é delimitada pelos ponteiros ERXST e ERXND.
- Esta zona é usada como um ring buffer de tramas.
- O ponteiro ERXWRPT indica memória livre e o ponteiro ERXRDPT indica memória com tramas recebidas.



EWRPT – ponteiro para escrita

ERDPT – ponteiro para leitura

Operações de acesso à memória interna

- Leitura de um bloco de dados a partir da memória interna. Antes é preciso definir o endereço no registo ERDPT. Este registo incrementa automaticamente.

```
static void read_buffer_memory(U8 * buffer, size_t size) {
   U8 command[] = { 0x3a };
   spi_transaction_begin(spi);
   spi_transaction_transfer(spi, 1, command, command);
   spi_transaction_transfer(spi, size, buffer, buffer);
   spi_transaction_end(spi);
}
```

- Escrita e um bloco de dados na memória interna. Antes é preciso definir o endereço no registo EWRPT.

```
static void write_buffer_memory(U8 * data, size_t size) {
   U8 command[] = { 0x7a };
   spi_transaction_begin(spi);
   spi_transaction_transfer(spi, 1, command, command);
   spi_transaction_transfer(spi, size, data, data);
   spi_transaction_end(spi);
}
```

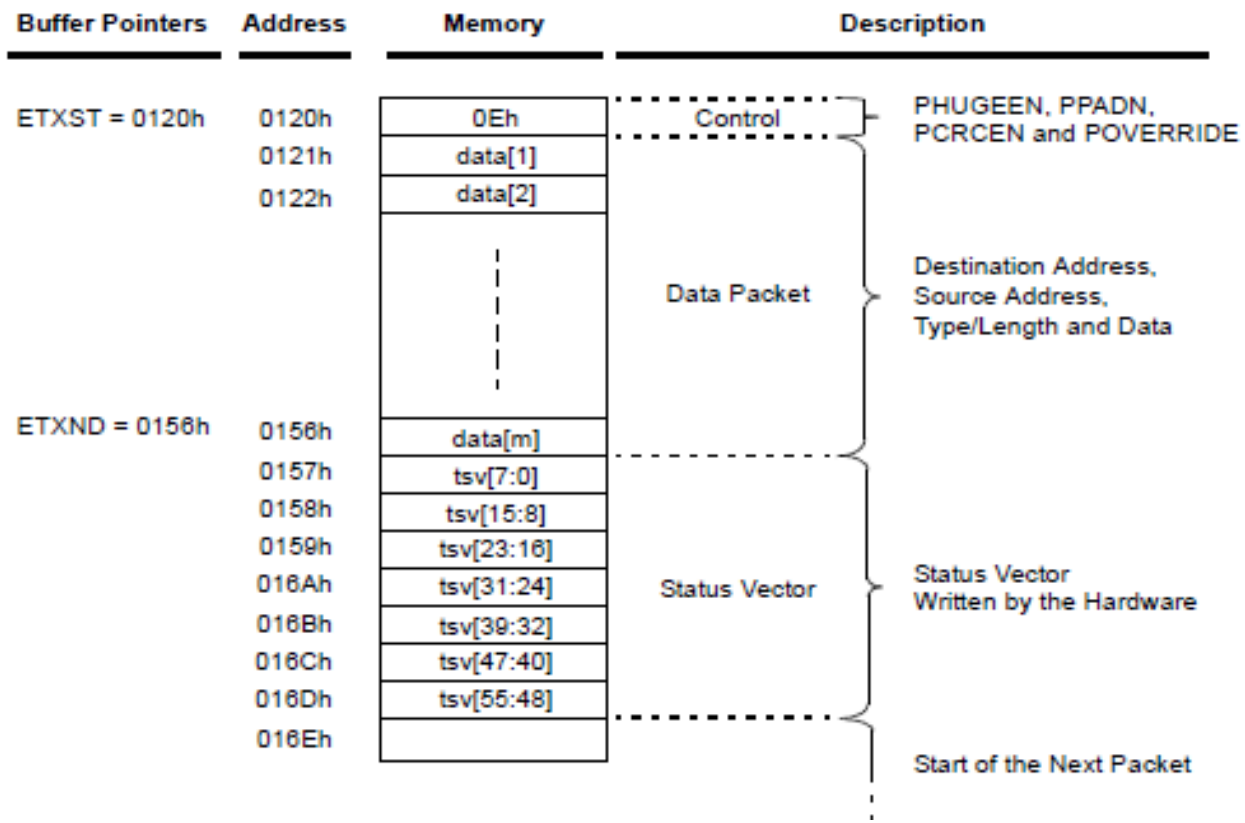
Acesso aos registos PHY

- O acesso aos registos PHY faz-se através dos registos MIREGADR e dos registos MIWR e MIRD.
- Em MIREGADR define-se o registo, escreve-se através do registo MIWR e lê-se através do registo MIRD.
- Nos registos MICMD e MISTAT controla-se o acesso.

```
void write_physical_register(int addr, U16 data) {
    reg_write8(MIREGADR, addr);
    reg_write16(MIWRL, data);
}

U16 read_physical_register(int addr) {
    U8 aux;
    reg_write8(MIREGADR, addr);
    reg_bit_field_set(MICMD, MICMD_MIIRD);
    do {
          aux = reg_read8(MISTAT);
    } while ((aux & MISTAT_BUSY) != 0);
    reg_bit_field_clear(MICMD, MICMD_MIIRD);
    return reg_read16(MIRDL);
}
```

Transmissão de pacotes (1)



- A trama a transmitir é disposta na memória após o byte de controlo e é seguida pelo array tsv (transmition status vector) com o resultado da transmissão.

Transmissão de pacotes (2)

```
void ethernet_send(U8 * packet, size_t packet_size) {
    U8 tsv[7], aux, control = 0x00;

    reg_write16(ETXSTL, TX_START);
    reg_write16(ETXNDL, TX_START + packet_size);
    reg_write16(EWRPTL, TX_START);
    write_buffer_memory(&control, 1);
    write_buffer_memory(packet, packet_size);

        int retry = 0;
    do {
        reg_bit_field_set(ECON1, ECON1_TXRST);
        reg_bit_field_clear(ECON1, ECON1_TXRST);
        reg_bit_field_clear(EIR, EIR_TXERIF | EIR_TXIF);
        reg_bit_field_set(ECON1, ECON1_TXRTS);

        do {
            aux = reg_read8(EIR);
        } while ((aux & EIR_TXIF) == 0);
        reg_bit_field_clear(ECON1, ECON1_TXRTS);

        reg_write16(ERDPTL, TX_START + packet_size + 1);
        read_buffer_memory(tsv, 7);

        aux = reg_read8(EIR);
    } while ((aux & EIR_TXERIF) && (tsv[29 / 8] & (1 << (29 % 8))) && retry++ < 10);
}
```

Definir os limites do pacote a transmitir

Transferir o pacote para a memória interna

Reset à lógica de transmissão

Reset flags de status

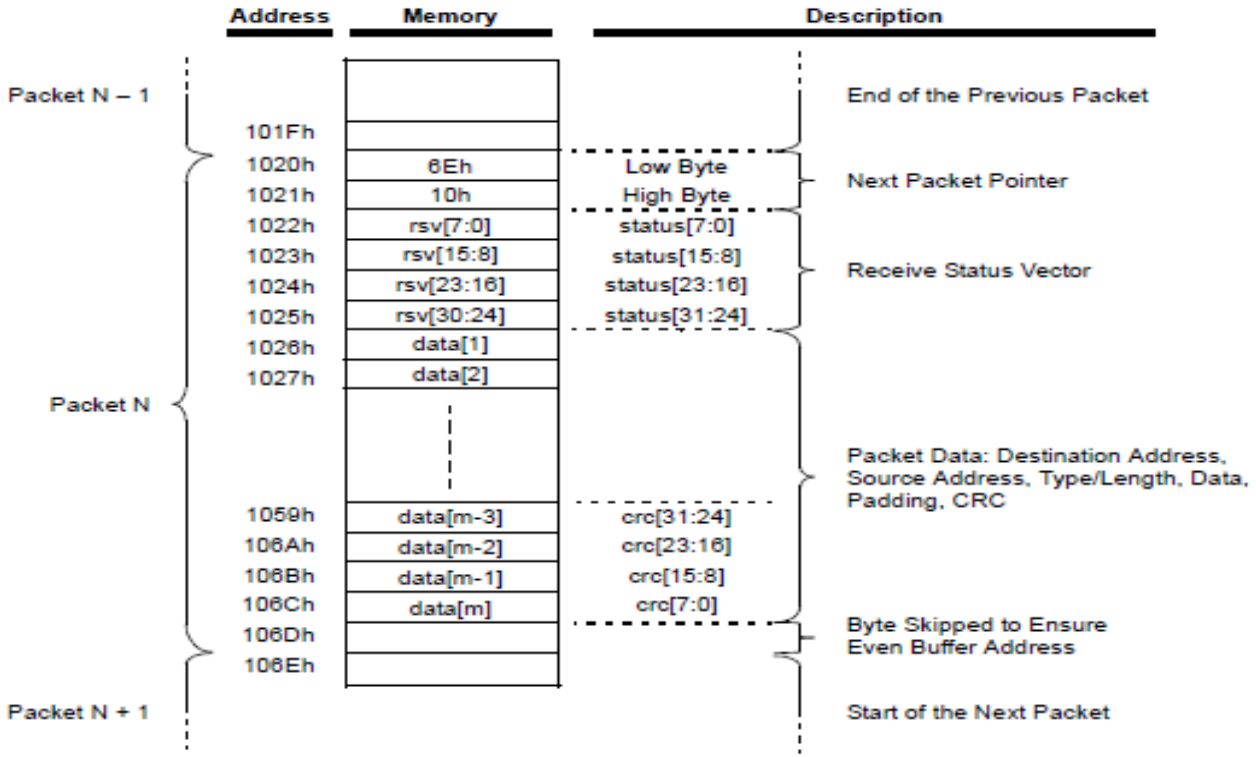
Activa pedido de transmissão

Esperar que a transmissão esteja concluída

Ler tsv

Repetir transmissão caso seja necessário (de acordo com errata, ponto13)

Recepção de pacotes (1)



- Os pacotes recebidos são dispostos na memória interna entre os ponteiros ERXRDPT e ERXWRPT.
- Cada pacote recebido (endereço destino, endereço fonte, tipo, dados e CRC) é antecedido por um ponteiro que indica o inicio do próximo pacote e pelo array rsv (receive status vector) com estado da recepção.

Recepção de pacotes (2)

```
size_t ethernet_recv(U8 * buffer, size_t buffer_size) {
    U8 rsv[4];
    static U16 rx_next_packet = RX_START;

    int n_packets = reg_read8(EPKTCNT);
    if (n_packets > 0) {

        reg_write16(ERDPTL, rx_next_packet);
        read_buffer_memory((U8*)&rx_next_packet, 2);
        read_buffer_memory(rsv, 4);
        size_t packet_size = ((U16)rsv[1] << 8) + rsv[0];
        read_buffer_memory(buffer, min(buffer_size, packet_size));

        reg_write16(ERXRDPTL, rx_next_packet);

        reg_bit_field_set(ECON2, ECON2_PKTDEC);

        return packet_size;
    }
    return 0;
}
```

Verificar se há pacotes recebidos

Preparar a transferencia do pacote

Ler rsv

Aqui começa o próximo pacote

Cálcular dimensão do pacote

Transferir o pacote

Sinalizar pacote consumido

Iniciação (1)

Regra geral, são utilizadas as definições por omissão (ver manual).
Estas correspondem às situações de operação mais comuns.

```
void ethernet_init(U8 * mac) {
   reg_write16(ERXSTL, RX_START);
   reg_write16(ERXNDL, RX_END);
   reg_write16(ERXRDPTL, RX_END));

   reg_write8(ERXFCON, 0);

   reg_write8(MAADR1, mac[0]);
   reg_write8(MAADR2, mac[1]);
   reg_write8(MAADR3, mac[2]);
   reg_write8(MAADR4, mac[3]);
   reg_write8(MAADR5, mac[4]);
   reg_write8(MAADR6, mac[5]);
```

Preparação da memória interna para a recepção de pacotes

Modo promíscuo: aceita todas as tramas (só para teste)

Definir endereço MAC para filtragem de pacotes

Iniciação (2)

```
    reg_write8(MACON1, MACON1_MARXEN);

    reg_write8(MACON3, MACON3_FRMLNEN
        + MACON3_TXCRCEN + MACON3_PADCFG0);

    reg_write8(MACON4, MACON4_DEFER)

    reg_write16(MAMXFLL, MAX_FRAME_LEN);

    reg_write8(MABBIPG, 0x12);
    reg_write16(MAIPGL, 0x0C12);

    reg_write8(ECON1, ECON1_RXEN);
}
```

Habilitar a recepção

Verificar campo Type, gerar CRC, inserir padding

Não desistir ao atingir máximo de colisões

Dimensão máxima de um pacote

Temporizações

Activa a recepção

Referências

- Microchip – ENC28J60 Data Sheet
- http://www.olimex.com/dev/enc28j60-h.html
- IEEE Standards – 802.3