Sistema d'Interrupcions del MSP432P401

Per fer servir les interrupcions de qualsevol recurs, hem de seguir una sèrie de passes:

- Habilitar les interrupcions. Al nostre processador es fa a 3 nivells:
  1. **A nivell de dispositiu** (s'ha de consultar el *Technical reference* per a cada dispositiu).
  2. **A nivell del controlador d'interrupcions del processador**, anomenat **NVIC** (*Nested Vectored Interrupt Controller*).
  3. **A nivell global** al registre de "status" del processador, fent servir la instrucció: **__enable_interrupt();**
- De fet, al primer pas que hem indicat, a nivell de dispositiu, podem emmascarar (inhabilitar) algunes interrupcions i mantenir altres actives.
- El controlador d'interrupcions també gestiona un sistema de prioritats, per aquelles situacions en que més d'una interrupció s'activi simultàniament.

Exemple d'Interrupció: vector amb diverses fonts

Si el port 2 el tinguéssim configurat com GPIOs d'entrada que poden generar interrupcions:

```
void PORT2_IRQHandler (void)  //interrupcions GPIO del port 2.
{
    uint8_t flag = P2IV
    P2IE &= 0x00;    //Inhabilitem temporalment les interrupcions de tot el port 2

    switch (flag) {
        case 0x02:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.0
                    break;
        case 0x04:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.1
                    break;
        case 0x06:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.2
                    break;
        case 0x08:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.3
                    break;
        case 0x0A:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.4
                    break;
        case 0x0C:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.5
                    break;
        case 0x0E:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.6
                    break;
        case 0x10:
    // Aquí posem el que volem fer si s'ha generat una interrupció al P2.7
                    break;
        default:    break;
                    }
    P2IE |= 0xFF;    //Tornem a habilitar les interrupcions del port 2
}
```

Figure 10-1. PxIV Register

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved			PxIV				
r0	r0	r0	r-0	r-0	r-0	r-0	r0

Table 10-4. PxIV Register Description

Bit	Field	Type	Reset	Description
15-5	Reserved	R	0h	Reserved. Reads return 0h
4-0	PxIV	R	0h	Port x interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Port x.0 interrupt; Interrupt Flag: PxIFG.0; Interrupt Priority: Highest 04h = Interrupt Source: Port x.1 interrupt; Interrupt Flag: PxIFG.1 06h = Interrupt Source: Port x.2 interrupt; Interrupt Flag: PxIFG.2 08h = Interrupt Source: Port x.3 interrupt; Interrupt Flag: PxIFG.3 0Ah = Interrupt Source: Port x.4 interrupt; Interrupt Flag: PxIFG.4 0Ch = Interrupt Source: Port x.5 interrupt; Interrupt Flag: PxIFG.5 0Eh = Interrupt Source: Port x.6 interrupt; Interrupt Flag: PxIFG.6 10h = Interrupt Source: Port x.7 interrupt; Interrupt Flag: PxIFG.7; Interrupt Priority: Lowest

Interrupcions als *Timers* de 16 bits

*** Cas del *timer* A0, si és el A1, A2 o A3 això canvia pel nom corresponent**

Com les fem servir:

- Cas *Timer A0, Timer A1, Timer A2 i Timer A3* (associats a CCR0): **Directes**.

```
void TA0_0_IRQHandler (void)*        /Cas del TA0. Aquest és el nom important
{
   TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;  //Hem de netejar el flag de la
                                                interrupció
   //Codi de la IRQ
}
```

- Cas *Timer A0, Timer A1, Timer A2 i Timer A3 (CCR1-CCR6 i TAxIFG):* **Indirectes.**

```
void TA0_N_IRQHandler (void)*        /Cas del TA0. Aquest és el nom important
{
      /* Analitzar els flags CCIFGx i TxIFG per saber qui ha demanat la interrupció,
         llavors fer que el corresponent a cadascuna.
         Com al cas dels Ports, si fem servir TAxIV, l'hem de guardar a una variable
         ja que quan accedim al registre es neteja automàticament */
}
```

Exemple timer

Disposem de dos fonts de rellotge: SMCLK a 24 MHz i ACLK a 32768 Hz.

Període SMCLK = (1/24e6) ≈ 0.042 µs.

Període ACLK = (1/32768) ≈ 30.518 µs.

Necessitem generar una interrupció continuament cada 333 µs:

Comptes amb ACLK = 333 µs * 32768 Hz ≈ 10.9 = 11

Període real amb ACLK amb 11 = 335.7 µs (pot no ser acceptable)

Comptes amb SMCLK = 333 µs * 24 MHz = 7992.0

7992 -> Necessitem un comptador de 13 bits (acceptable, dels timers MSP432 són de 16)

Configuració:

Divider 1 (ID) a 8 i dividir 2 (IDEX) a 1: Freqüència real 24 MHz / 8 = 3 MHz

TAxCCR0 = (333 µs * 3 MHz) – 1 = 998

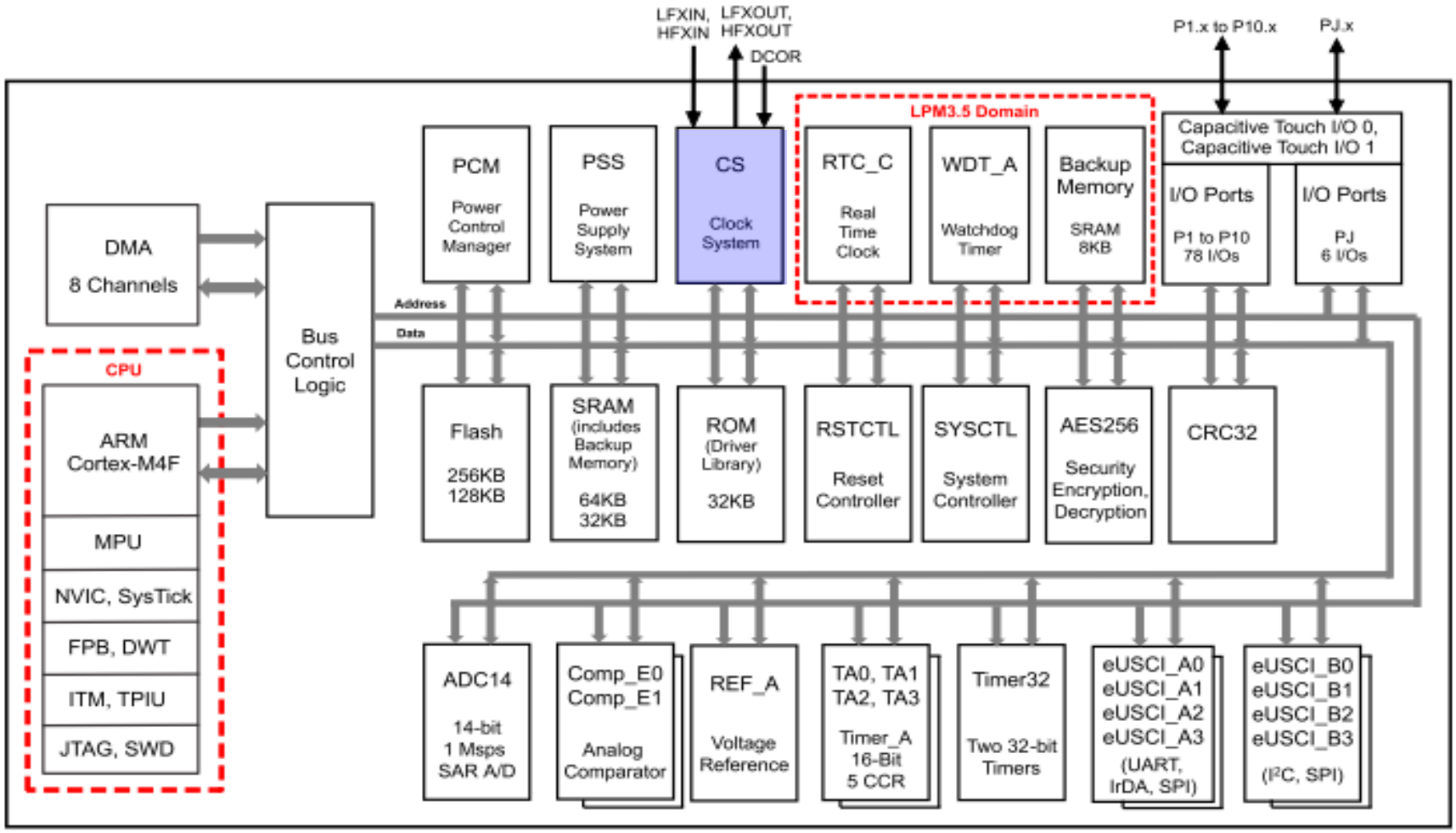
Up mode timer: MC = 1

PROGRAMACIÓ D'ARQUITECTURES ENCASTADES

UART

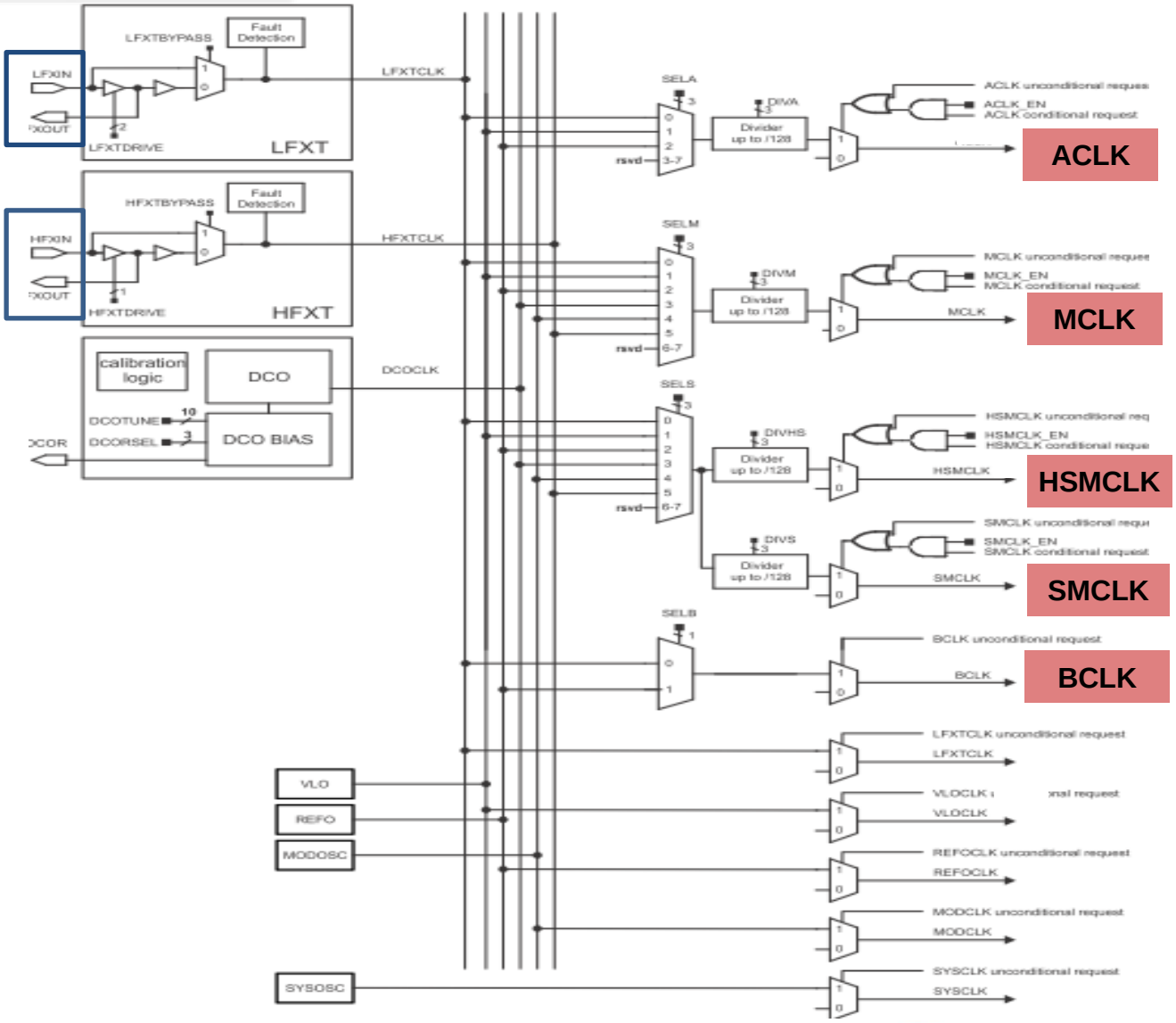
Classe 4

Bloc de Rellotge del MSP432P401



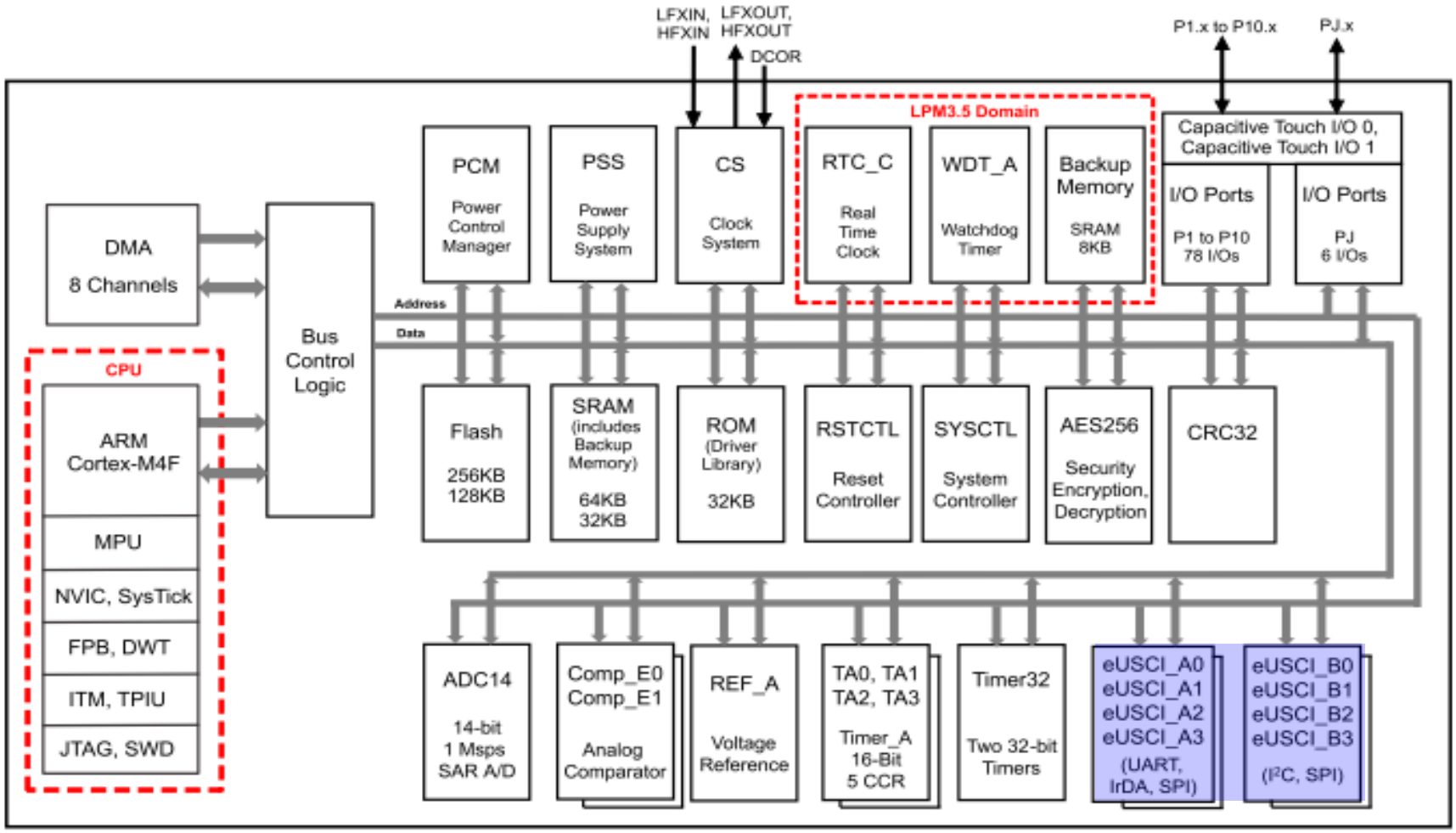
Sistema de Rellotge CS*

- **Diverses fonts de rellotge "independents":**
  - ***Low Frequency***
    - **LFXTCLK Low Freq (Extern rang de 32kHz o menys)**
    - **VLOCLK 10 kHz (Intern, molt baixa potència)**
    - **REFOCLK 32768 Hz (Intern LF)**
  - ***High Frequency***
    - **HFXTCLK High-Freq (Extern 1MHz-48MHz)**
    - **DCOCLK Oscil·lador intern programable (3MHz per defecte)**
    - **MODCLK intern 25MHz**
    - **SYSOSC intern 5MHz**
- **Fonts de rellotge disponibles pels dispositius:**
  - **ACLK Auxiliary Clock. Es pot seleccionar a partir de les fonts *LowFreq* anteriors.**
  - **MCLK Master clock (CPU)**
  - **HSMCLK Subsystem master clock**
  - **SMCLK Secondary master clock**
  - **BCLK Low-speed backup clock.**
- **Els Rellotges funcionen quan es necessiten.**



** CS = Clock System*

Bloc de Comunicacions Sèrie del MSP432P401



Comunicacions Sèrie al MSP432P401

El MSP432P401 té 8 interfícies eUSCI (enhanced *Universal Serial Communication Interface*). Els mòduls eUSCI es fan servir per les comunicacions sèrie, i poden treballar amb protocols síncrons (SPI i I^2C) i asíncrons (UART i IrDA).

Cada eUSCI consta de dos tipus d'interfícies diferents: A i B.

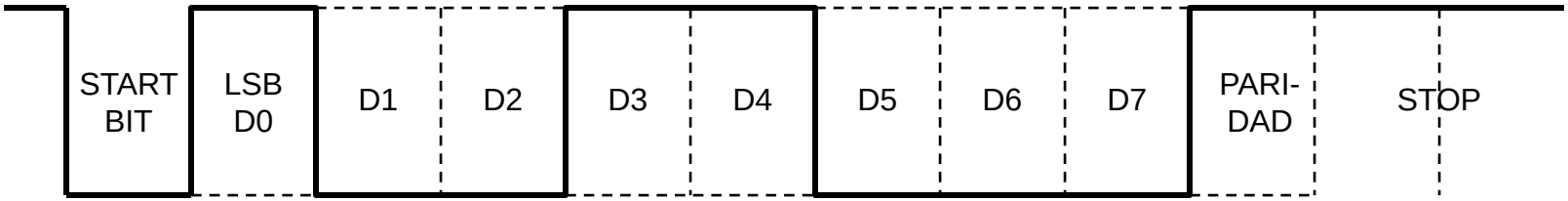
- El mòdul **eUSCI_An** pot treballar com:
  - SPI (3 pin o 4 pin).
  - UART.
  - IrDA.

- El mòdul **eUSCI_Bn** pot treballar com:
  - SPI (3 pin o 4 pin).
  - I^2C.

Nosaltres hem de treballar en mode UART ja que és com es comuniquen els mòduls motors i sensor del robot.

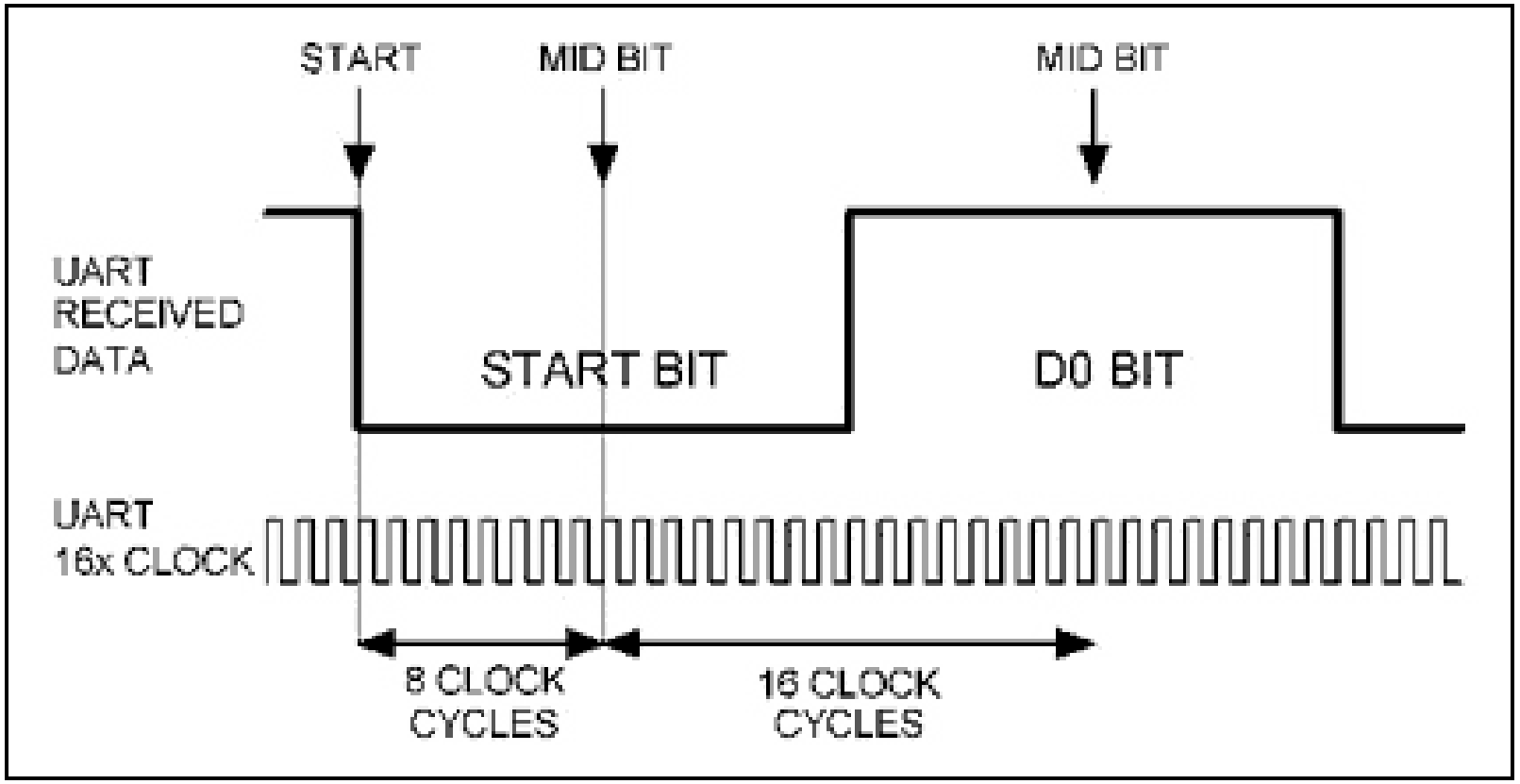
Comunicacions Sèrie UART (RS232)

- BUS SERIE (mínim 2 fils i GND).
- ASÍNCRON (el transmissor i el receptor tenen rellotges diferents).
- El sincronisme es realitza per cada caràcter transmès.
- FORMAT: de 7 a 12 bits per caràcter a transmetre.
  - 1 bit de START.
  - De 5 a 8 bits d'INFORMACIÓ.
  - 1 bit de PARITAT (opcional).
  - 1 o 2 bits de STOP.



Comunicacions Sèrie UART (RS232)

- UART sampling vs oversampling



eUSCI en mode UART

La UART connecta amb l'exterior via 2 pins externs:

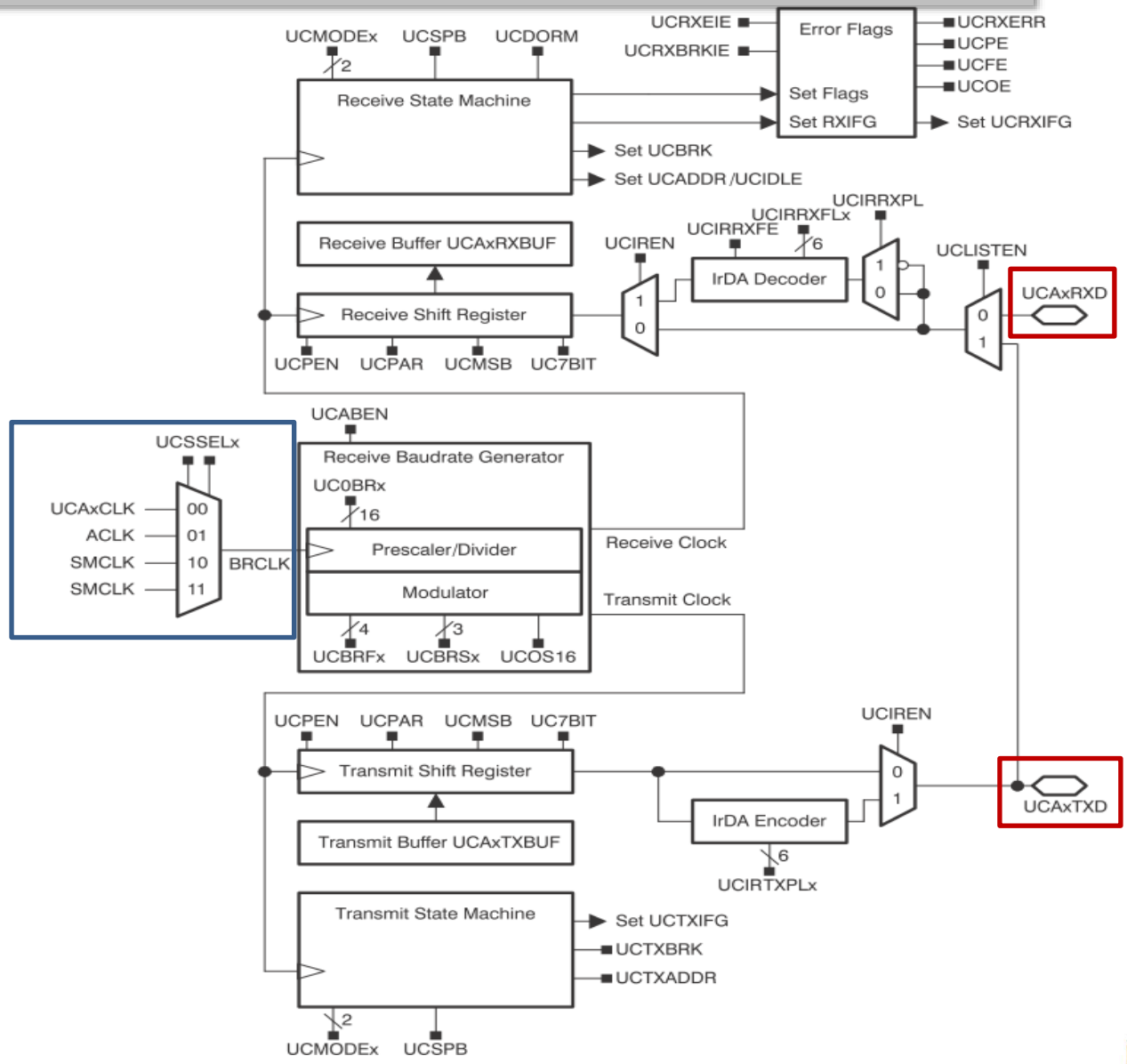
- UCAxRXD (lectura).
- UCAxTXD (escriptura).

Característiques:

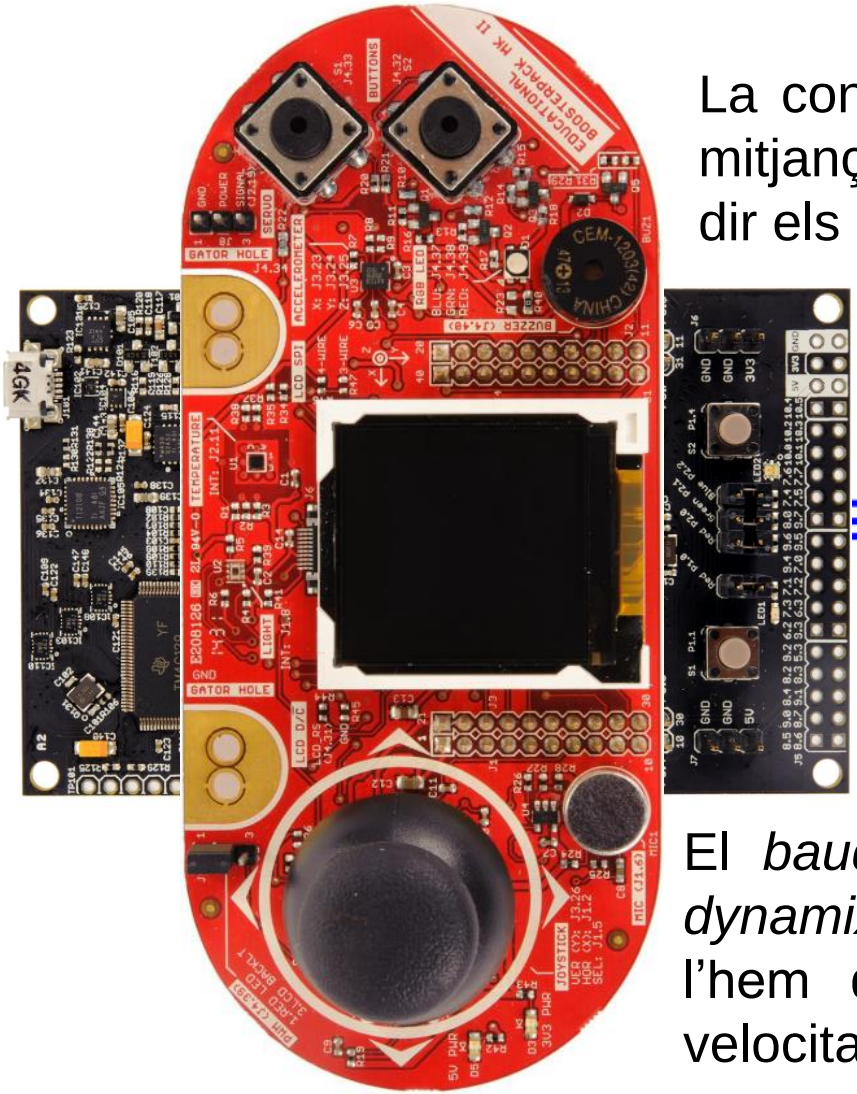
- **7 o 8** bits de dades amb paritat senar o parella o sense paritat.
- *Buffers* independents de transmissió i recepció.
- Transmissió i recepció amb LSB primer o MSB primer programable.
- Detecció de bit de Start a la recepció per treballar en modes de baix consum.
- Programació del *baud rate*.
- *Flags* de status per detecció d'errors.
- Interrupcions independents per transmetre i rebre.

Per triar el mode UART a la USCI, s'ha de posar a 0 el bit UCSYNC.

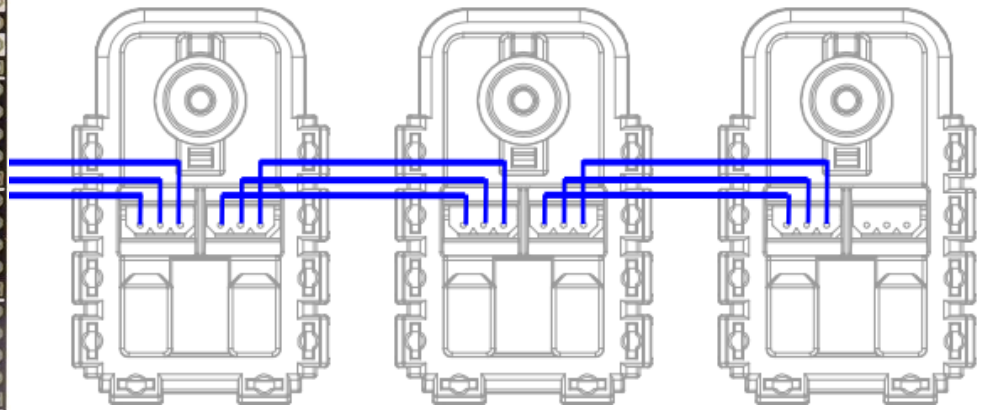
Diagrama de blocs de la eUART del MSP432P401



Connexió UART amb els Mòduls Dynamixel



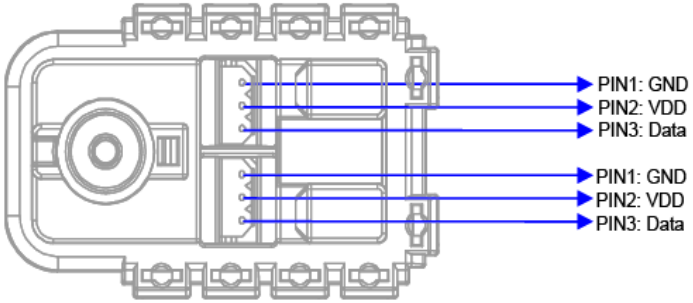
La connexió entre la placa i els mòduls la fem mitjançant una UART A0 del port 1. I com vàrem dir els mòduls es connecten en *daisy chain*.



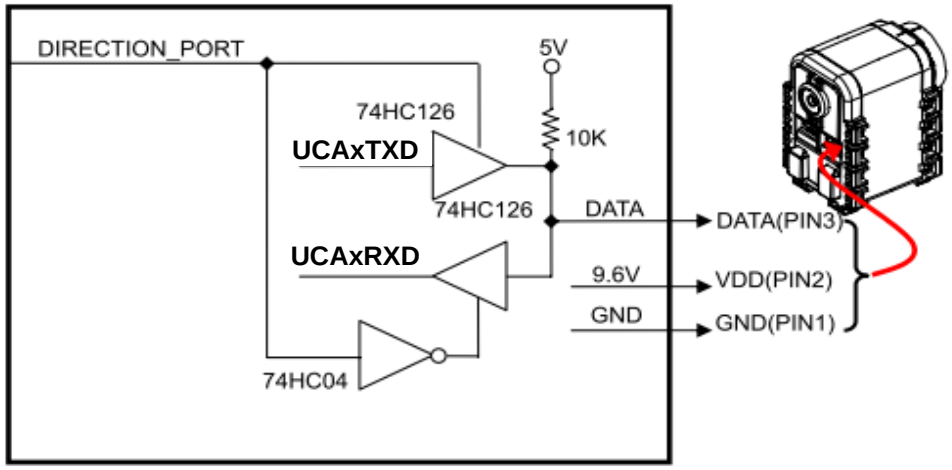
El *baud rate* que hem programat als mòduls *dynamixel* és de 115200 bps, i per tant la UART l'hem de programar per treballar a aquesta velocitat.

Connexió UART amb els Mòduls Dynamixel

El segon punt a tenir en compte és que els mòduls *dynamixel* fan servir comunicació asíncrona però *Half-duplex* (només té una línia "Data" per transmetre i rebre) mentre que la UART en principi és *Full-duplex* (té una línia per transmetre UCAxTXD i una per rebre UCAxRXD).



Per solucionar-lo fem un circuit que passa de dues línies a una i viceversa:



Hem de tenir en compte que des del microcontrolador, per programa, hem de controlar el senyal "*DIRECTION_PORT*". A la nostra placa l'hem connectat al *port* 3, al *pin* 3.0. Hem d'inicialitzar-lo com GPIO de sortida, i gestionar el senyal en funció de si volem enviar o rebre missatges.

Configuració de la UART

```
void Init_UART(void)
{
  UCA0CTLW0 |= UCSWRST;                  //Fem un reset de la USCI, desactiva la USCI
  UCA0CTLW0 |= UCSSEL__SMCLK;            //UCSYNC=0 mode  asíncron
                                         //UCMODEx=0 seleccionem mode UART
                                         //UCSPB=0 nomes 1 stop bit
                                         //UC7BIT=0 8 bits de dades
                                         //UCMSB=0 bit de menys pes primer
                                         //UCPAR=x ja que no es fa servir bit de paritat
                                         //UCPEN=0 sense bit de paritat
                                         //Triem SMCLK (24MHz) com a font del clock BRCLK
  UCA0MCTLW = UCOS16;                    // Necessitem sobre-mostreig => bit 0 = UCOS16 = 1
  UCA0BRW = 13;                          //Prescaler de BRCLK fixat a 13. Com SMCLK va a24MHz,
                                         //volem un baud rate de 115200kb/s i fem sobre-mostreig de 16
                                         //el rellotge de la UART ha de ser de ~1.85MHz (24MHz/13).
  UCA0MCTLW |= (0x25 << 8);              //UCBRSx, part fractional del baud rate
  //Configurem els pins de la UART
  P1SEL0 |= BIT2 | BIT3;                 //I/O funció: P1.3 = UART0TX, P1.2 = UART0RX
  P1SEL1 &= ~ (BIT2 | BIT3);
  UCA0CTLW0 &= ~UCSWRST;                 //Reactivem la línia de comunicacions sèrie
  EUSCI_A0->IFG &= ~EUSCI_A_IFG_RXIFG;       // Clear eUSCI RX interrupt flag
  //EUSCI_A0->IE |= EUSCI_A_IE_RXIE;     // Enable USCI_A0 RX interrupt, nomes quan tinguem la recepcio
  }
```

Configuració de la UART

NOTE: Baud-rate settings quick set up

To calculate the correct settings for the baud-rate generation, perform these steps:

1. Calculate N = f_{BRCLK} / baud rate [if N > 16 continue with step 3, otherwise with step 2]
2. OS16 = 0, UCBRx = INT(N) [continue with step 4]
3. OS16 = 1, UCBRx = INT(N/16), UCBRFx = INT([(N/16) – INT(N/16)] × 16)
4. UCBRSx can be found by looking up the fractional part of N (= N - INT(N)) in table Table 24-4
5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed

Table 24-4. UCBRSx Settings for Fractional Portion of N = f_{BRCLK}/Baud Rate

Fractional Portion of N	UCBRSx[1]		Fractional Portion of N	UCBRSx[1]
0.0000	0x00		0.5002	0xAA
0.0529	0x01		0.5715	0x6B
0.4003	0x92		0.9004	0xFB
0.4286	0x53		0.9170	0xFD
0.4378	0x55		0.9288	0xFE

(1) The UCBRSx setting in one row is valid from the fractional portion given in that row until the one in the next row

Technical reference Manual 24.3.10 Setting a Baud Rate

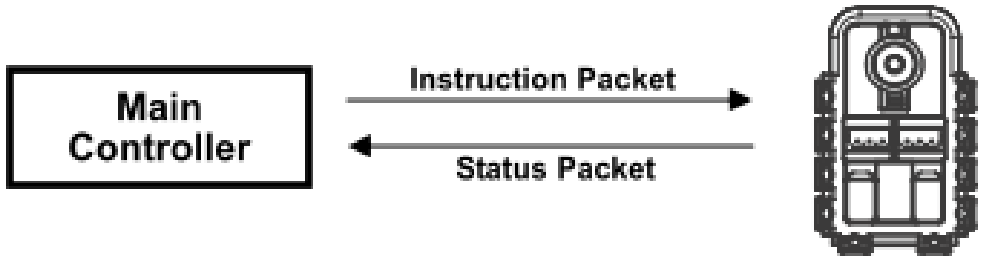
Funcions Bàsiques de Comunicació amb la UART

```
//Defines
typedef uint8_t byte;
#define TXD0_READY (UCA0IFG & UCTXIFG)

/* funcions per canviar el sentit de les comunicacions */
void Sentit_Dades_Rx(void)
{                          //Configuració del Half Duplex dels motors: Recepció
  P3OUT &= ~BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 0 (Rx)
}

void Sentit_Dades_Tx(void)
{                          //Configuració del Half Duplex dels motors: Transmissió
  P3OUT |= BIT0;     //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Tx)
}
```

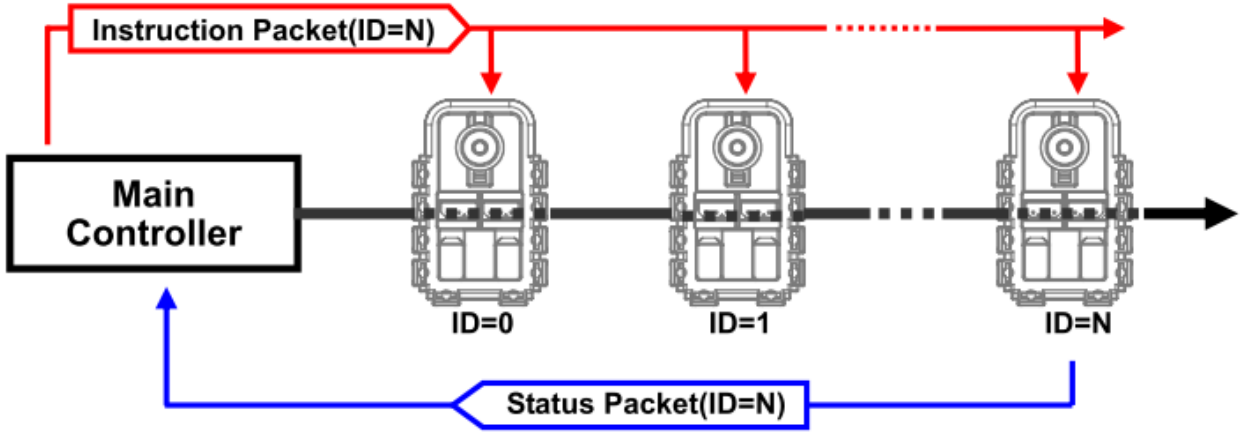
```
/* funció TxUACx(byte): envia un byte de dades per la UART 0 */
void TxUACx(uint8_t bTxdData)
{
  while(!TXD0_READY);         // Espera a que estigui preparat el buffer de transmissió
  UCA0TXBUF = bTxdData;
}
```

Protocol Comunicació dels Mòduls del Robot

Paquets: El microcontrolador envia un "*Instruction Packet*" (format per diversos bytes) al mòdul robot i aquest li contesta amb un "*Status Packet*".



Al "*Instruction Packet*" hi ha un paràmetre que és l'Identificador (ID) del mòdul al que va dirigit, només aquest mòdul rebrà les dades que s'envien. L'ID ha de ser únic, no pot repetir-se a cap altre mòdul.



Mapa de memòria DYNAMIXEL

No-volàtil (EEPROM)

Address	Item	Access	Initial Value
0(0X00)	Model Number(L)	RD	12(0x0C)
1(0X01)	Model Number(H)	RD	0(0x00)
2(0X02)	Version of Firmware	RD	?
3(0X03)	ID	RD,WR	1(0x01)
4(0X04)	Baud Rate	RD,WR	1(0x01)
5(0X05)	Return Delay Time	RD,WR	250(0xFA)
6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
10(0x0A)	(Reserved)	-	0(0x00)
11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
14(0X0E)	Max Torque(L)	RD,WR	255(0XFF)
15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
16(0X10)	Status Return Level	RD,WR	2(0x02)
17(0X11)	Alarm LED	RD,WR	4(0x04)
18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
19(0X13)	(Reserved)	RD,WR	0(0x00)
20(0X14)	Down Calibration(L)	RD	?
21(0X15)	Down Calibration(H)	RD	?
22(0X16)	Up Calibration(L)	RD	?
23(0X17)	Up Calibration(H)	RD	?

Volàtil (RAM)

24(0X18)	Torque Enable	RD,WR	0(0x00)
25(0X19)	LED	RD,WR	0(0x00)
26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
32(0X20)	Moving Speed(L)	RD,WR	0
33(0X21)	Moving Speed(H)	RD,WR	0
34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
36(0X24)	Present Position(L)	RD	?
37(0X25)	Present Position(H)	RD	?
38(0X26)	Present Speed(L)	RD	?
39(0X27)	Present Speed(H)	RD	?
40(0X28)	Present Load(L)	RD	?
41(0X29)	Present Load(H)	RD	?
42(0X2A)	Present Voltage	RD	?
43(0X2B)	Present Temperature	RD	?
44(0X2C)	Registered Instruction	RD,WR	0(0x00)
45(0X2D)	(Reserved)	-	0(0x00)
46[0x2E)	Moving	RD	0(0x00)
47[0x2F)	Lock	RD,WR	0(0x00)
48[0x30)	Punch(L)	RD,WR	32(0x20)
49[0x31)	Punch(H)	RD,WR	0(0x00)

Instruccions

Instruction	Function	Value	Number of Parameter
PING	No action. Used for obtaining a Status Packet	0x01	0
READ DATA	Reading values in the Control Table	0x02	2
WRITE DATA	Writing values to the Control Table	0x03	2 ~
REG WRITE	Similar to WRITE_DATA, but stays in standby mode until the ACION instruction is given	0x04	2 ~
ACTION	Triggers the action registered by the REG_WRITE instruction	0x05	0
RESET	Changes the control table values of the Dynamixel actuator to the Factory Default Value settings	0x06	0
SYNC WRITE	Used for controlling many Dynamixel actuators at the same time	0x83	4~

Protocol Comunicació dels Mòduls del Robot

INSTRUCTION PACKET

Format dels Paquets: seqüència de bytes enviat pel microcontrolador

OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	…	PARAMETER N	CHECK SUM

0xFF, 0xFF: Indiquen el començament d'una trama.

ID: Identificador únic de cada mòdul *Dynamixel* (entre 0x00 i 0xFD).
El identificador 0xFE és un "*Broadcasting ID*" que van a tots els mòduls (aquests no retornaran *Status Packet*)

LENGTH: El número de bytes del paquet (trama) = Nombre de paràmetres + 2

INSTRUCTION: La instrucció que se li envia al mòdul.

PARAMETER 1...N: No sempre hi ha paràmetres, però hi ha instruccions que si necessiten.

CHECK SUM: Paràmetre per detectar possibles errors del comunicació, es calcula així:

Check Sum = ~(ID + LENGTH + INSTRUCTION + PARAMETER 1 + ... + PARAMETER N)

Funció TX d'una trama "*Instruction*" als Mòduls del Robot

```
//TxPacket()  3 paràmetres: ID del Dynamixel, Mida dels paràmetres, Instruction byte. torna la mida del "Return packet"
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16])
{
  byte bCount,bCheckSum,bPacketLength;
  byte TxBuffer[32];
  Sentit_Dades_Tx();                    //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Transmetre)
  TxBuffer[0] = 0xff;                   //Primers 2 bytes que indiquen inici de trama FF, FF.
  TxBuffer[1] = 0xff;
  TxBuffer[2] = bID;                     //ID del mòdul al que volem enviar el missatge
  TxBuffer[3] = bParameterLength+2; //Length(Parameter,Instruction,Checksum)
  TxBuffer[4] = bInstruction;            //Instrucció que enviem al Mòdul
  for(bCount = 0; bCount < bParameterLength; bCount++) //Comencem a generar la trama que hem d'enviar
    {
                TxBuffer[bCount+5] = Parametros[bCount];
    }
  bCheckSum = 0;
  bPacketLength = bParameterLength+4+2;
  for(bCount = 2; bCount < bPacketLength-1; bCount++) //Càlcul del checksum
    {
                bCheckSum += TxBuffer[bCount];
    }
  TxBuffer[bCount] = ~bCheckSum;                    //Escriu el Checksum (complement a 1)
  for(bCount = 0; bCount < bPacketLength; bCount++) //Aquest bucle és el que envia la trama al Mòdul Robot
    {
                TxUACx(TxBuffer[bCount]);
    }
  while( (UCA0STATW & UCBUSY));     //Espera fins que s'ha transmès el últim byte
Sentit_Dades_Rx();                  //Posem la línia de dades en Rx perquè el mòdul Dynamixel envia resposta
return(bPacketLength);
}
```

Protocol Comunicació dels Mòduls del Robot

STATUS PACKET

Format dels Paquets: seqüència de bytes amb que respon el mòdul

OXFF	0XFF	ID	LENGTH	ERROR	PARAMETER1	PARAMETER2	…	PARAMETER N	CHECK SUM

0xFF, 0xFF: Indiquen el començament d'una trama.

ID: Identificador del mòdul.

LENGTH: El número de bytes del paquet.

ERROR:

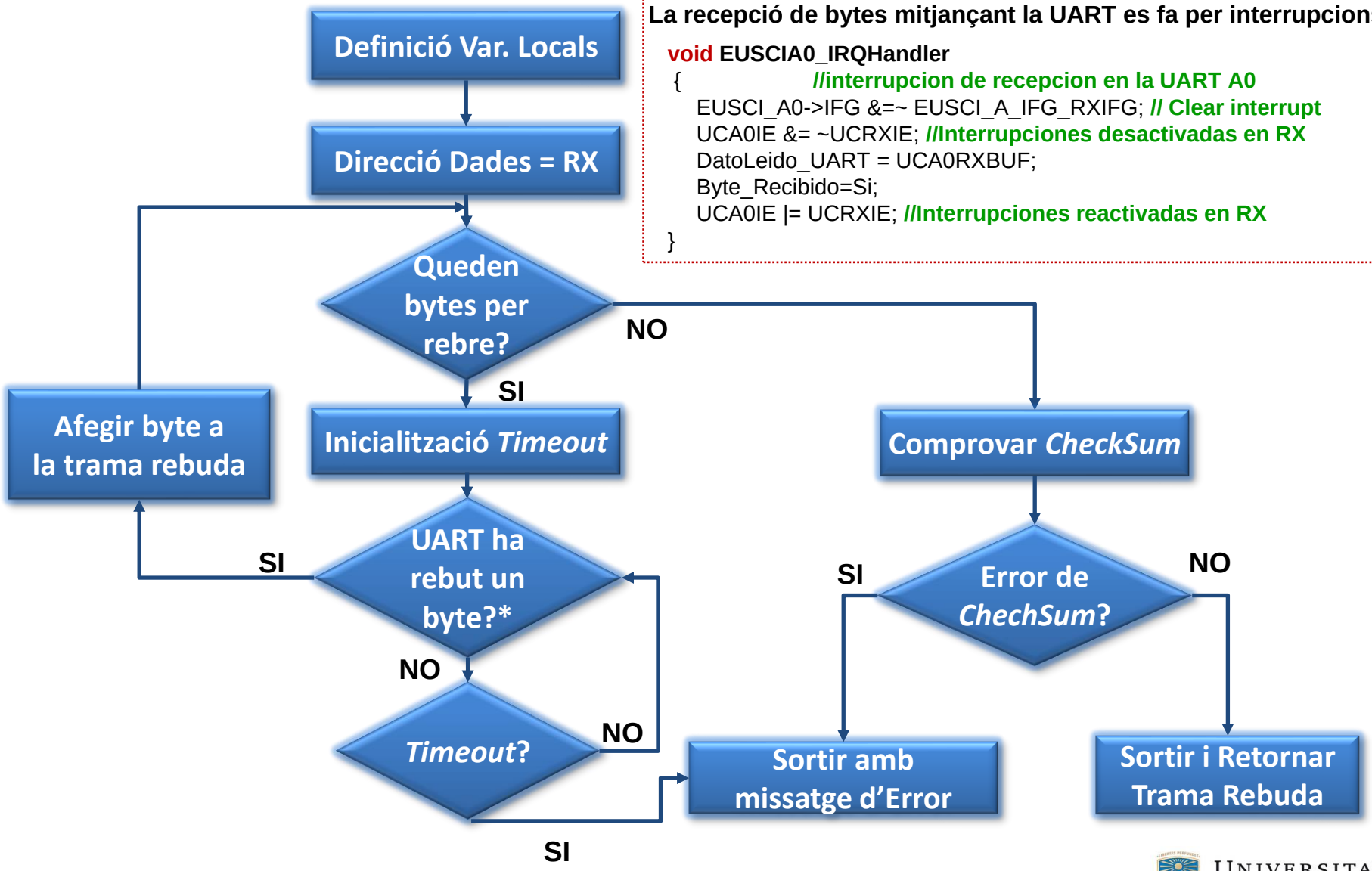
Bit	Name	Details
Bit 7	0	-
Bit 6	Instruction Error	Set to 1 if an undefined instruction is sent or an action instruction is sent without a Reg_Write instruction.
Bit 5	Overload Error	Set to 1 if the specified maximum torque can't control the applied load.
Bit 4	Checksum Error	Set to 1 if the checksum of the instruction packet is incorrect.
Bit 3	Range Error	Set to 1 if the instruction sent is out of the defined range.
Bit 2	Overheating Error	Set to 1 if the internal temperature of the Dynamixel unit is above the operating temperature range as defined in the control table.
Bit 1	Angle Limit Error	Set as 1 if the Goal Position is set outside of the range between CW Angle Limit and CCW Angle Limit.
Bit 0	Input Voltage Error	Set to 1 if the voltage is out of the operating voltage range as defined in the control table.

PARAMETER 1...N: Si es necessiten.

CHECK SUM: Paràmetre per detectar possibles errors del comunicació, es calcula així:

Check Sum = ~(ID + LENGTH + ERROR + PARAMETER 1 + ... + PARAMETER N)

Funció RX d'una trama "*Status*" dels Mòduls del Robot



La recepció de bytes mitjançant la UART es fa per interrupcions.

```
void EUSCIA0_IRQHandler
 {                  //interrupcion de recepcion en la UART A0
    EUSCI_A0->IFG &=~ EUSCI_A_IFG_RXIFG; // Clear interrupt
    UCA0IE &= ~UCRXIE; //Interrupciones desactivadas en RX
    DatoLeido_UART = UCA0RXBUF;
    Byte_Recibido=Si;
    UCA0IE |= UCRXIE; //Interrupciones reactivadas en RX
 }
```

Exemple

Example 16	**Turn on the LED and Enable Torque for a Dynamixel actuator with an ID of 0**
Instruction Packet	Instruction = WRITE_DATA, Address = 0x18, DATA = 0x01, 0x01
Communication	->[Dynamixel]:FF FF 00 05 03 18 01 01 DD (LEN:009) <-[Dynamixel]:FF FF 00 02 00 FD (LEN:006)
Status Packet Result	NO ERROR

Funció RX d'una trama "*Status*" als Mòduls del Robot

// Aquest exemple no és complert, en principi RxPacket() torna una estructura*"Status packet*" que bàsicament consisteix en un array amb "Status Packet"+ un byte indicant si hi ha un TimeOut. Això s'ha fet així perquè en C no es pot posar un array com paràmetre de tornada.
Per altre banda, la part mostrada només llegeix els primers 4 bytes del *status packet*. Això és perquè el quart byte indica precisament quants bytes queden per llegir, el que vol dir que s'ha de fer un altre bucle "for" semblant per llegir els bytes que falten....
Evidentment, es podria fer d'altres maneres, per exemple enviant com paràmetre el número de bytes a llegir...

```
struct RxReturn RxPacket(void)
{
      struct RxReturn respuesta;
      byte bCount, bLenght, bChecksum;
      byte Rx_time_out=0;
      DataDirection_Rx();      //Ponemos la linea half duplex en Rx
      Activa_TimerA1_TimeOut();
      for(bCount = 0; bCount < 4; bCount++) //bRxPacketLength; bCount++)
      {
        Reset_Timeout();
        Byte_Recibido=No;    //No_se_ha_recibido_Byte();
        while (!Byte_Recibido) //Se_ha_recibido_Byte())
        {
          Rx_time_out=TimeOut(1000);       // tiempo en decenas de microsegundos
          if (Rx_time_out)break;//sale del while
         }
         if (Rx_time_out)break;  //sale del for si ha habido Timeout
         //Si no, es que todo ha ido bien, y leemos un dato:
         respuesta.StatusPacket[bCount] = DatoLeido_UART; //Get_Byte_Leido_UART();
      }//fin del for
      if (!Rx_time_out)
     // Continua llegint la resta de bytes del Status Packet
```

Afegiu un codi per garantir no escriure en les primeres posicions!

Bibliografia i Documentació

- MSP432P4xx *Technical Reference Manual*.
- MSP432P401 *Datasheet*.
- www.ti.com/msp432
- MSP-EXP432P401R *LaunchPad User's Guide*.
- *Educational BoosterPack* EDUMKII *User's Guide*.
- http://www.bioloid.info/tiki/tiki-index.php
- Manuals mòduls Bioloid AX-12 i AX-S1.