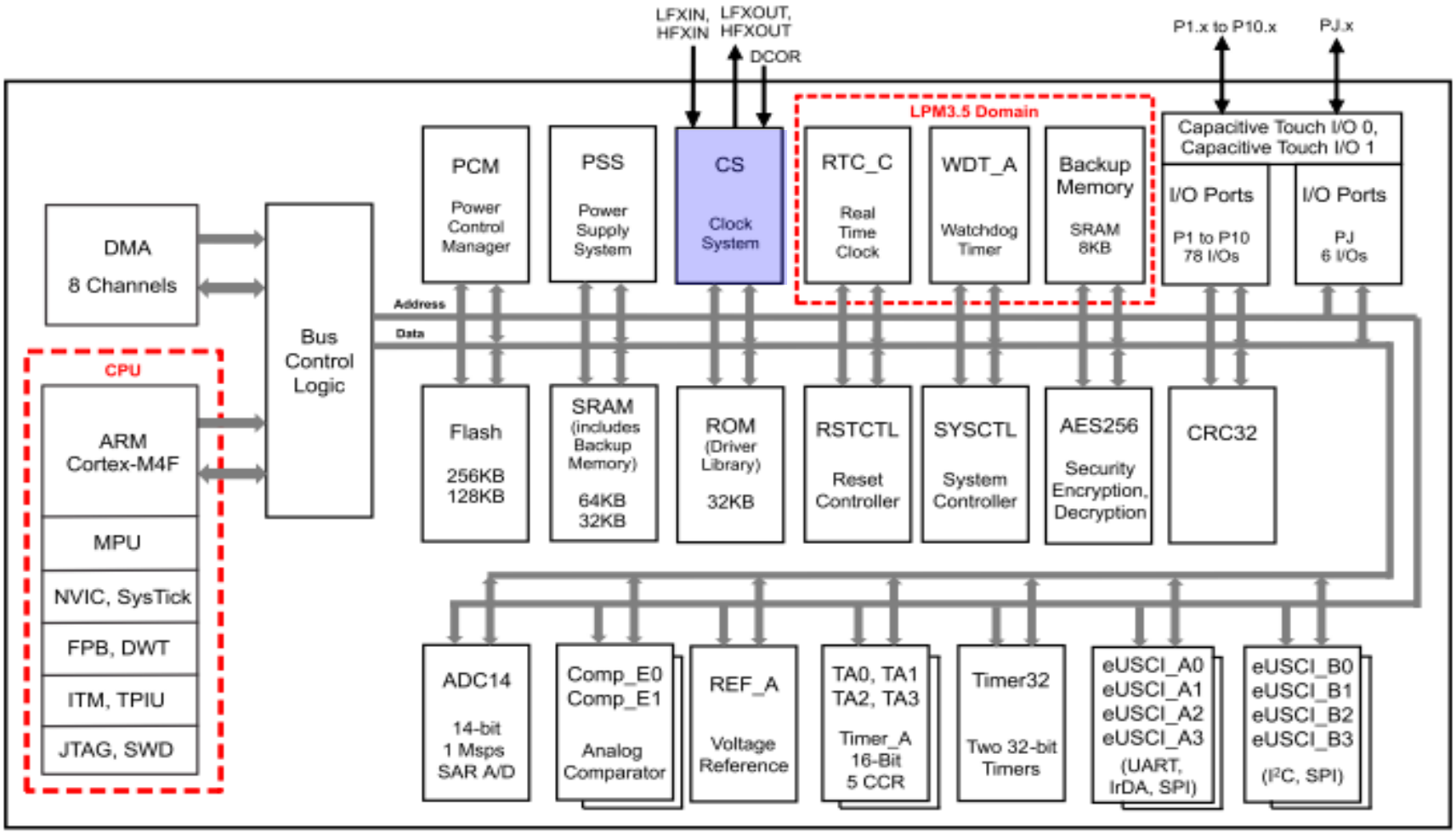
PROGRAMACIÓ D'ARQUITECTURES ENCASTADES

UART

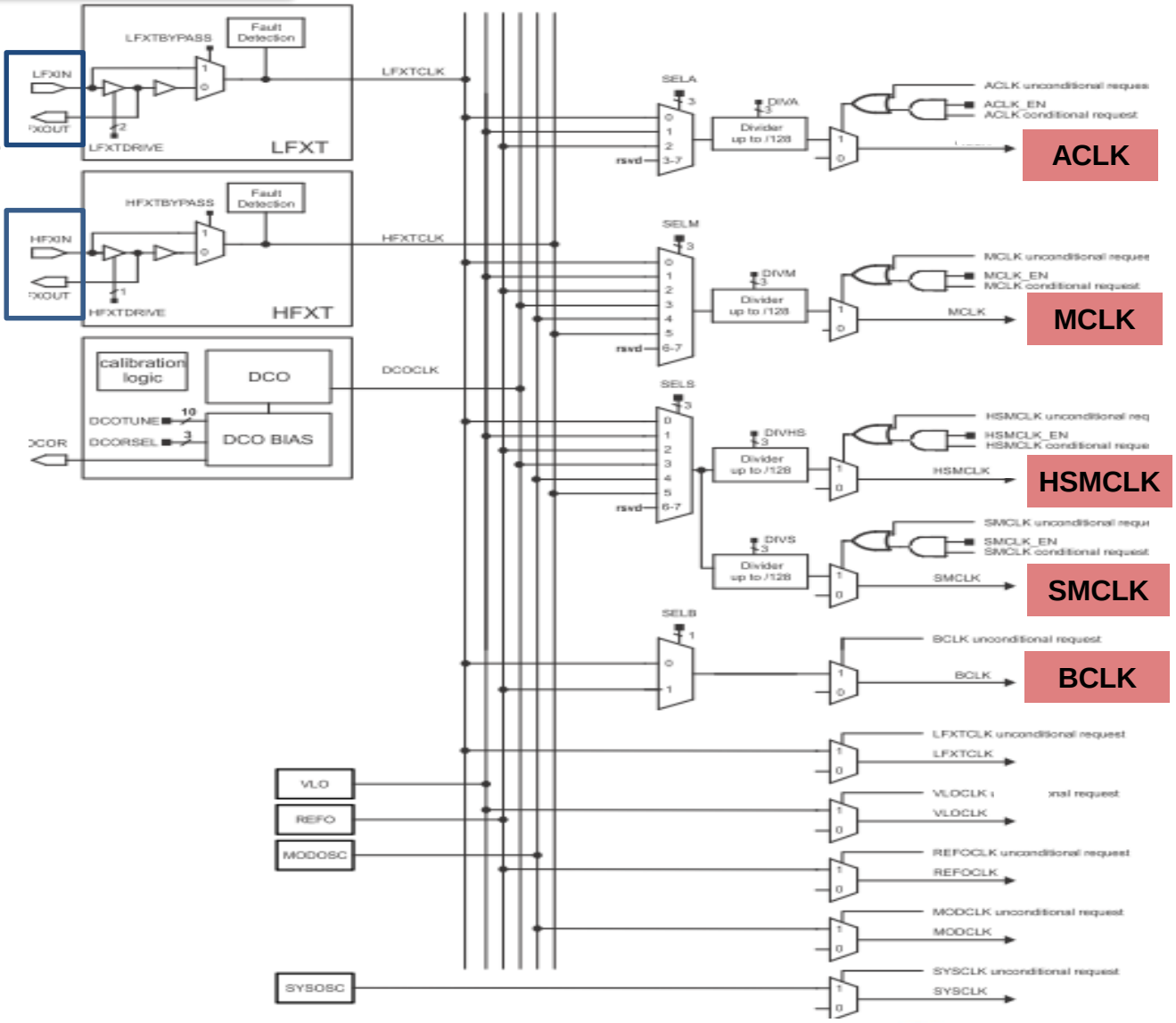
Classe 4

Bloc de Rellotge del MSP432P401



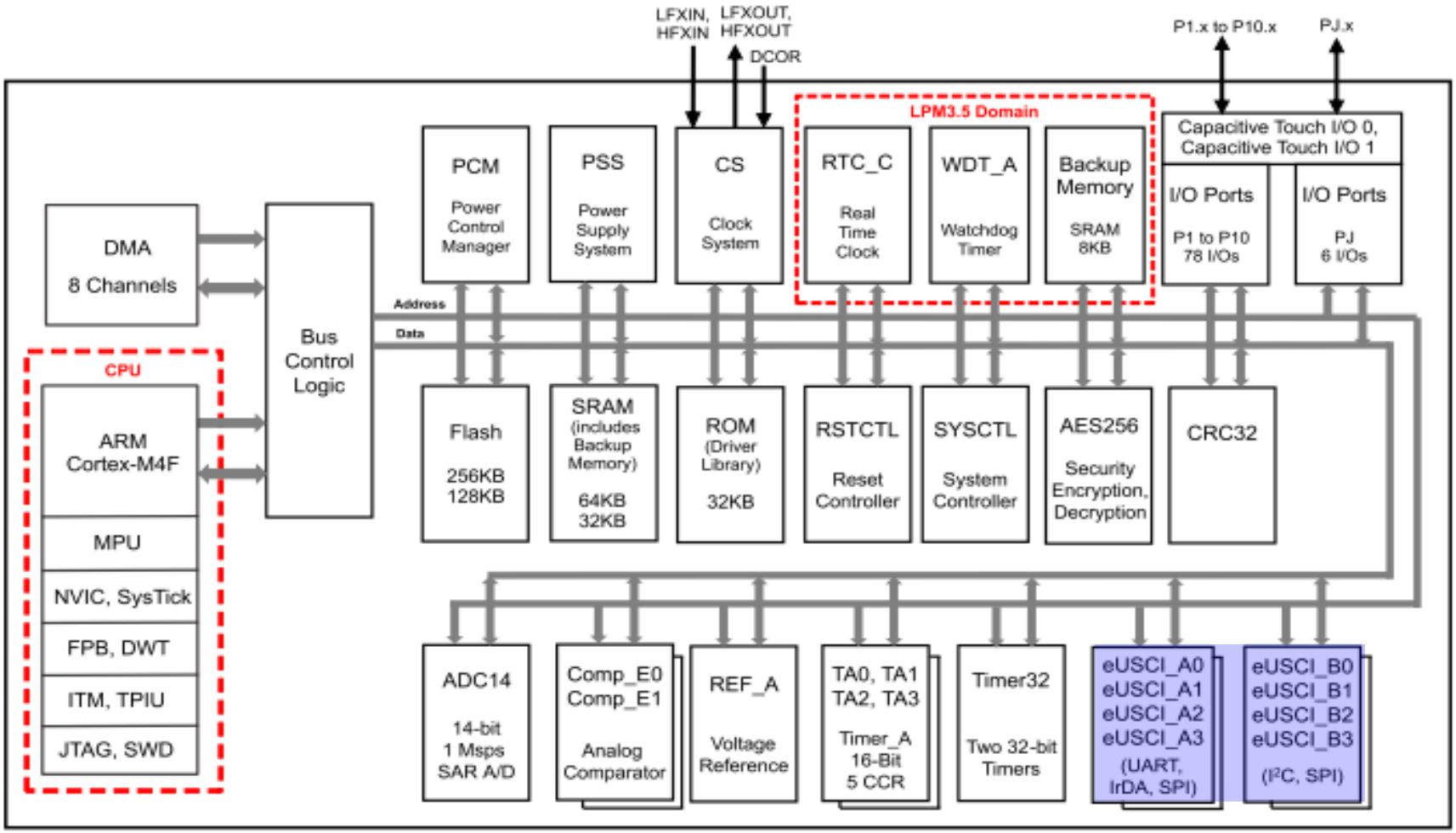
Sistema de Rellotge CS*

- **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.

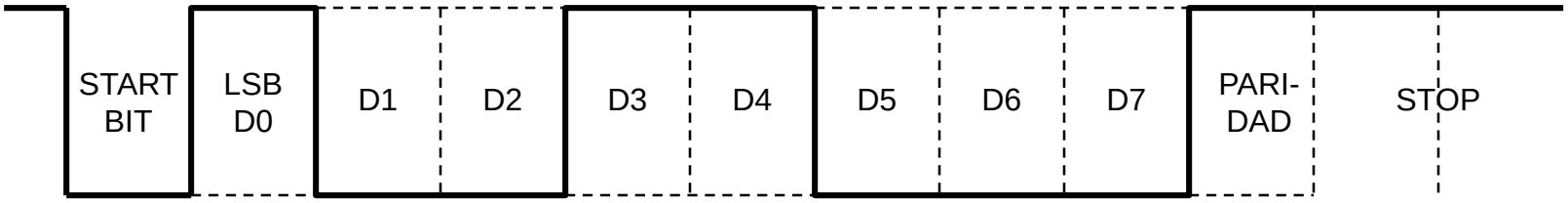
- El mòdul **eUSCI_An** pot treballar com:
  - SPI (3 pin o 4 pin).
  - UART.
  - IrDA.

- 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.

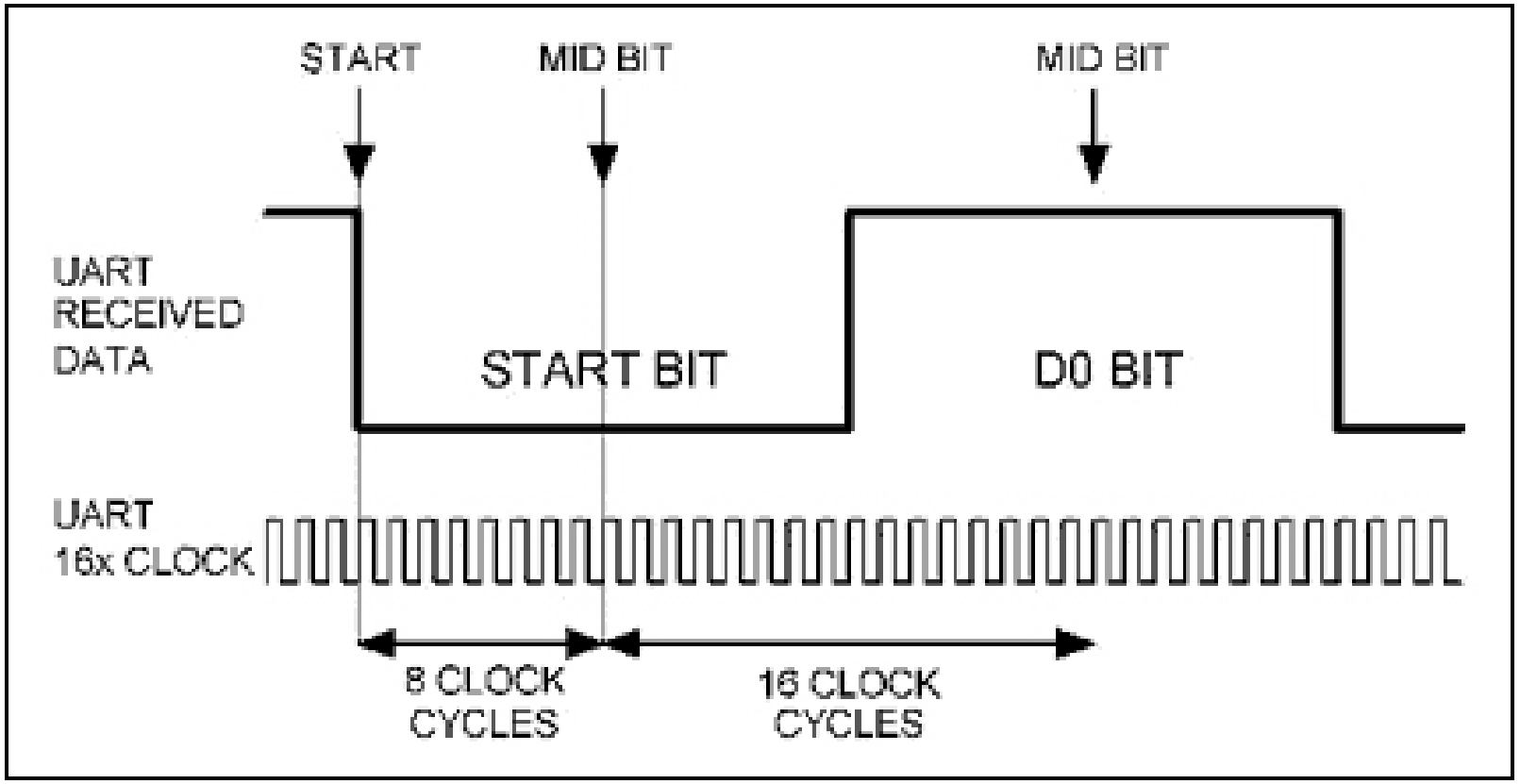
Comunicacions Sèrie UART (RS232)

- 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:

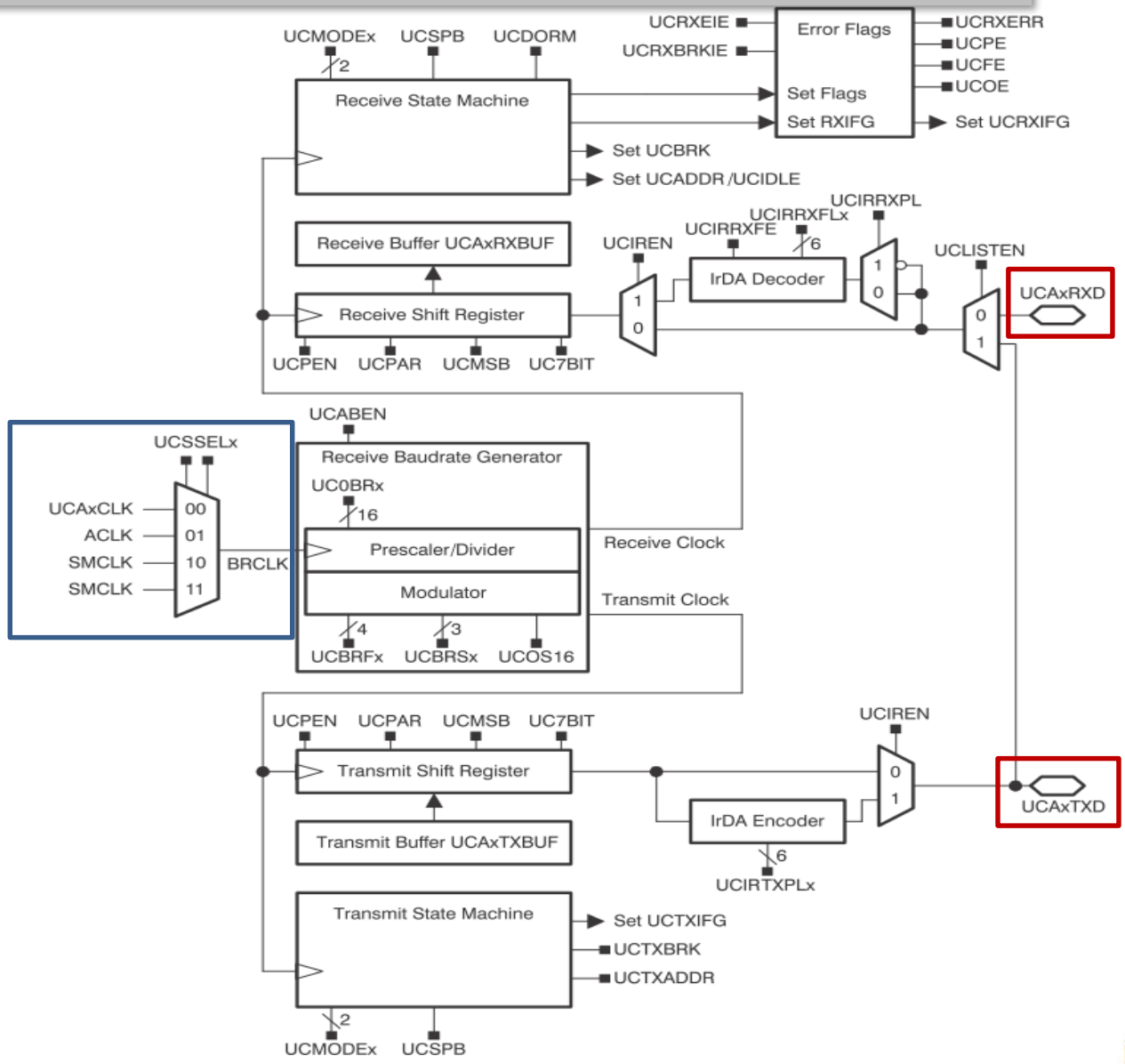
- UCAxRXD (lectura).
- UCAxTXD (escriptura).

Característiques:

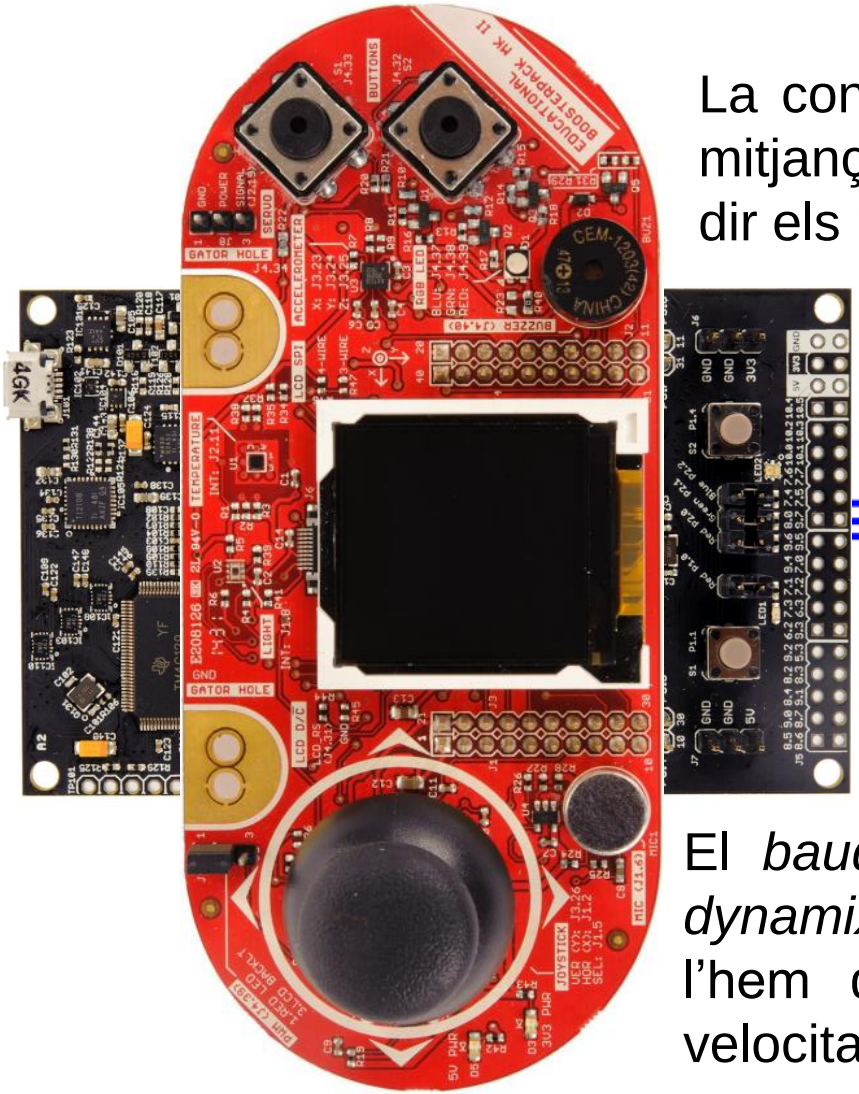
- **7 o 8** bits de dades amb paritat senar o parella o sense paritat.
- *Buffers* independents de transmissió i recepció.
- Transmissió i recepció amb LSB primer o MSB primer programable.
- Detecció de bit de Start a la recepció per treballar en modes de baix consum.
- Programació del *baud rate*.
-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

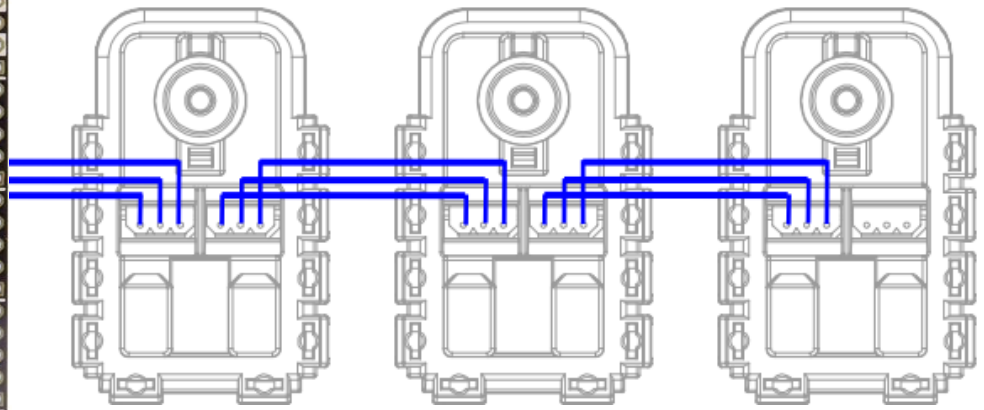
Diagrama de blocs de la eUART del MSP432P401



Connexió UART amb els Mòduls Dynamixel



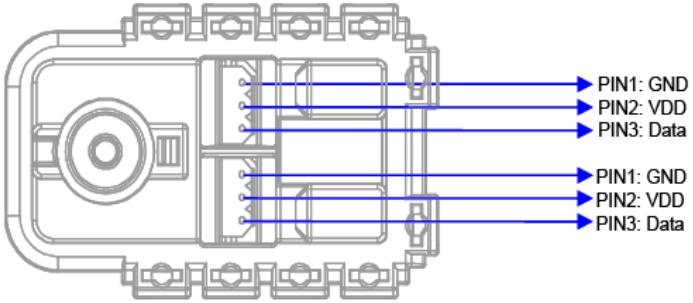
La connexió entre la placa i els mòduls la fem mitjançant una UART A2 del port 3. I com vàrem dir els mòduls es connecten en *daisy chain*.



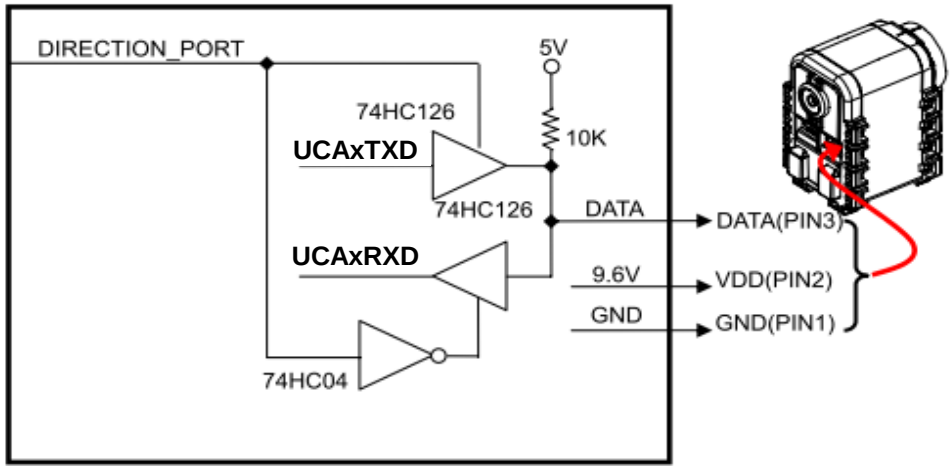
El *baud rate* que hem programat als mòduls *dynamixel* és de 500kb/s, i per tant la UART l'hem de programar per treballar a aquesta velocitat.

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)
{
  UCA2CTLW0 |= UCSWRST;                 //Fem un reset de la USCI, desactiva la USCI
  UCA2CTLW0 |= 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
  UCA2MCTLW = UCOS16;                   // Necessitem sobre-mostreig => bit 0 = UCOS16 = 1
  UCA2BRW = 3;                          //Prescaler de BRCLK fixat a 3. Com SMCLK va a24MHz,
                                        //volem un baud rate de 500kb/s i fem sobre-mostreig de 16
                                        //el rellotge de la UART ha de ser de 8MHz (24MHz/3).
  //Configurem els pins de la UART
  P3SEL0 |= BIT2 | BIT3;                //I/O funció: P3.3 = UART2TX, P3.2 = UART2RX
  P3SEL1 &= ~ (BIT2 | BIT3);
  //Configurem pin de selecció del sentit de les dades Transmissió/Recepeció
  P3SEL0 &= ~BIT0;                       //Port P3.0 com GPIO
  P3SEL1 &= ~BIT0;
  P3DIR |= BIT0;                        //Port P3.0 com sortida (Data Direction selector Tx/Rx)
  P3OUT &= ~BIT0;                       //Inicialitzem Sentit Dades a 0 (Rx)
  UCA2CTLW0 &= ~UCSWRST;                //Reactivem la línia de comunicacions sèrie
 // UCA2IE |= UCRXIE;                   //Això només s'ha d'activar quan tinguem la rutina de recepció
  }
```

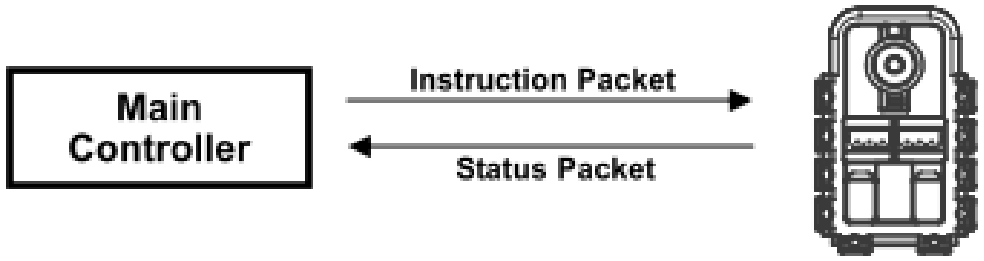
Funcions Bàsiques de Comunicació amb la UART

```
//Defines
typedef uint8_t byte;
#define TXD2_READY (UCA2IFG & 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)
}
```

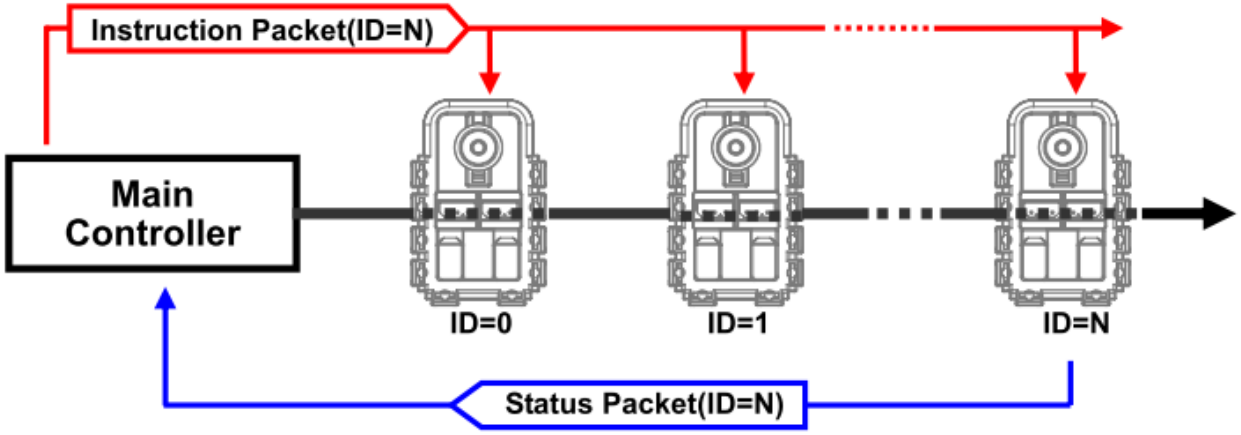
```
/* funció TxUAC0(byte): envia un byte de dades per la UART 0 */
void TxUAC2(byte bTxdData)
{
  while(!TXD2_READY);          // Espera a que estigui preparat el buffer de transmissió
  UCA2TXBUF = bTxdData;
}
```

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.



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
    {
                TxUAC2(TxBuffer[bCount]);
    }
  while( (UCA2STATW&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.

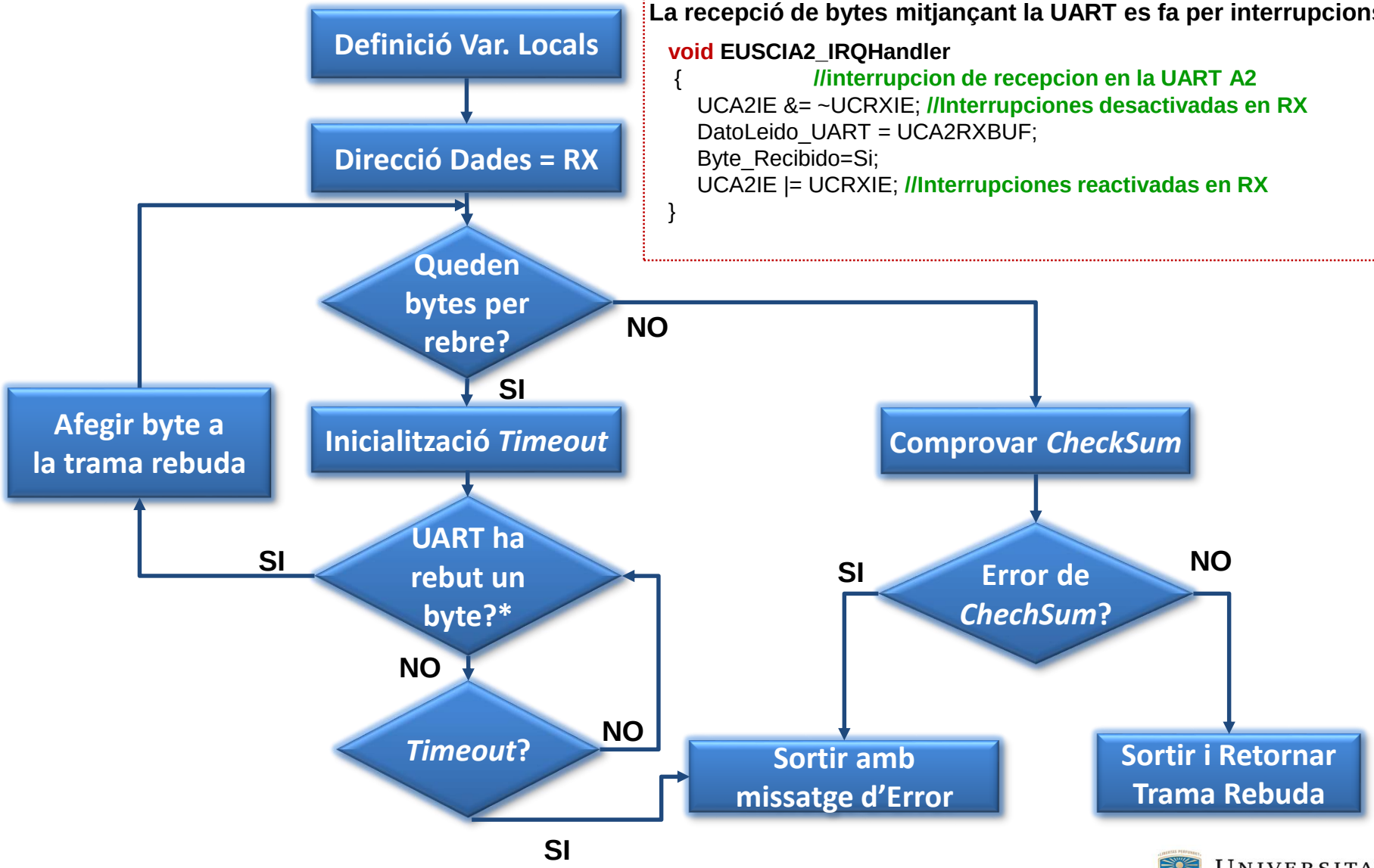
CHECK SUM: Paràmetre per detectar possibles errors del comunicació, es calcula així:

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 EUSCIA2_IRQHandler
 {                    //interrupcion de recepcion en la UART A2
    UCA2IE &= ~UCRXIE; //Interrupciones desactivadas en RX
    DatoLeido_UART = UCA2RXBUF;
    Byte_Recibido=Si;
    UCA2IE |= UCRXIE; //Interrupciones reactivadas en RX
 }
```



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 (ara 10ms)
            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
```

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.