Pràctiques de Sistemes Digitals i Microprocessadors

Curs 2016-2017

Pràctica 2 – Fase 1

El Walkie-Textie

	Login	Nom
Alumnes	Ls30759	Samuel Tavares da Silva
	Ls30652	Gabriel Cammany Ruiz

	Placa	Memòria	Nota
Entrega			

Data	11/05/2017

Portada de la memòria

Pràctiques de Sistemes Digitals i Microprocessadors

Curs 2016-2017

Pràctica 2 – Fase 1

El Walkie-Textie

	Login	Nom
Alumnes	Ls30759	Samuel Tavares da Silva
	Ls30652	Gabriel Cammany Ruiz

	Placa	Memòria	Nota
Entrega			

Data	11/05/2017

Portada de l'alumne

Índex

Síntesi de l'enunciat

Aquesta practica es basa en la implementació d'una xarxa de telecomunicació basada en enllaços de radiofreqüència controlats per microcontroladors. En el nostre cas sens proposa l'ús d'un emissor central i receptors a les sucursals, d'aquesta manera podrem enviar missatges sense fils, per tal de no suposar una molèstia a l'hora de testejar i implementar el projecte s'ha decidit enviar el missatges com a text, enlloc de missatges de veu com un sistema de Walkie-Talkie típic.

La implementació d'aquest projecte es pot dividir en dos parts, la part de transmissió i la de recepció. Hem decidit realitzar aquesta practica en dos clares fases, la primera encarregada en implementar l'emissor i la segona en el sistema per tal de rebre els missatges per radio freqüència.

Per el que toca la primera fase es requereix el disseny i implementació d'un sistema que es basa en un microcontrolador PIC18F4321 programat en assembler. Cal destacar que aquest sistema requereix comunicar-se amb un ordinador utilitzant el protocol de comunicacions sèrie RS-232, mitjançant un cable DB9 i una interfície Java.

Per tal de comunicar la placa amb l'ordinador caldrà una interfície gràfica, que serà implementada en java.

Aquesta haurà de permetre per una banda, la configuració de l'enviament de bytes, es a dir, establir el baudrate i port corresponents al canal sèrie de l'ordinador.
I per altre banda, per el que s'ha implementat, enviament de les dades que indiqui l'usuari (Amb un màxim de 300 bytes) per el protocol 232. Per realitzar-ho, l'usuari tindrà a disposició de un TextField per introduir les dades i dos botons.
Boto d'enviament de dades que es comunicarà amb la PIC per enviar les dades i emmagatzemar-les en la ram i el boto d'enviament per Radio Freqüència, que avisarà a la PIC que comenci la transmissió per RF.

Pel que fa la placa que es comunica amb el PC, serà l'encarregada de rebre la informació sèrie i emmagatzemar-la fent ús d'un microcontrolador PIC18F4321, 2 polsadors, que ens permetran tenir un control a part de la interfície, i 10 LEDs encarregats de representar l'estat de transmissió.

Per últim en aquesta fase al rebre l'ordre d'enviament RF, el microcontrolador llegirà els valors guardats a la seva RAM interna, que ha de correspondre amb l'últim missatge carregat i els anirà enviant en sèrie per RF a 100bps i mitjançant una codificació Manchester.

Plantejament

Tal com hem comentat anteriorment, aquesta fase es corresponia de dos apartats. La part de la interfície Java que es controlava des de el PC i per un altre banda la PIC que era l'encarregada d'emmagatzemar la informació i transmetre-la per RF.

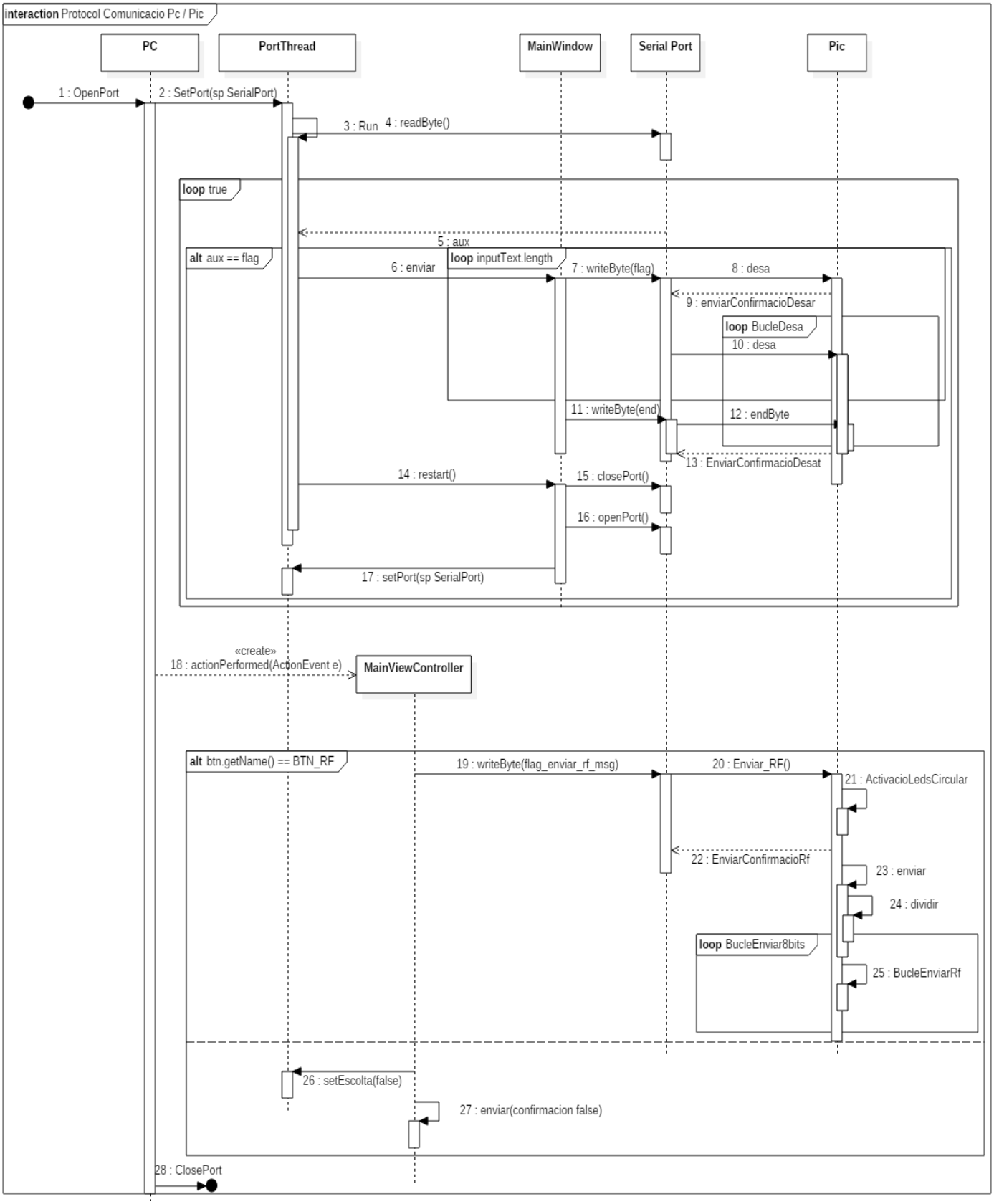
Per el que fa la interfície Java, esta realitzada amb la llibreria de Java Swing/AWT. Aquesta ens venia donada amb l'enunciat amb parts sense realitzar ja que eren diferents per cada cas.
Per el nostre protocol de comunicació amb la PIC, hem tingut que separar el programa en dos apartats.
Una part estava constantment escoltant per el port per si rebia un byte de la PIC indicant que comences a enviar les dades i per altre banda teníem la part del Listener dels botons. Aquesta era l'encarregada de que un cop l'usuari premés els boto de enviament per RF o Carrega dades, avises a la PIC que es comences a realitzar la acció.

Aquest protocol el teníem que controlar que també es tingues en compte en altre banda del DB9, es a dir, en la nostre PIC. Així doncs l'assembler l'hem tingut que dissenyar de tal manera que escolta tant a les peticions del ordinador com als dos botons que hi ha a la placa. Que com ja hem explicat, realitzen la mateixa funció que la interfície gràfica (Sense òbviament, la inserció del missatge i configuracions).

Per desar, el que farem serà anar-nos a la posició de la RAM que volem començar a escriure les dades, en el nostre cas, 80h. Un cop allà, mentre el ordinador no ens digui que ja hem acabat (amb un "End byte"), anirem desant dada a dada tot confirmant en cada moment que ja hem desat la dada correctament.

En el cas d'enviar per RF. El que farem serà per un pin de la PIC, enviarem les dades a 100bps amb la codificació Manchester. Tot aquest procés l'haurem d'anar mostrant per els 10 leds. Es a dir, s'encendra 1 LED per cada 10% que s'hagi enviat. Per dur aterme el recompte del 10%, farem sempre el càlcul del 10% del numero de bytes a enviar abans i no pararem fins que arribem al final. Per no utilitzar les taules, intentarem fer una aproximació de una divisió per 10.

Diagrama de mòduls



Disseny

Configuracions del microcontrolador

Configuracions generals

Les nostres configuracions principals son les següents:

```
CONFIG OSC = HSPLL
CONFIG PBADEN = DIG
CONFIG WDT = OFF
CONFIG LVP = OFF
```

Hem configurat el oscil·lador a 40Mhz, utilitzant el oscil·lador intern de 10Mhz amb el PLL. Es a dir, el multiplicador que fa que de 10Mhz el multipliquem per 4 per poder arribar als 40Mhz. Tot i així, es podria fer perfectament aquesta fase sense l'ús del PLL.

Donat que necessitem el port B en comptes d'analògic a digital, necessitarem definir el PBADEN a digital.

Per un altre part, la configuració del WDT a OFF ens dona l'opció de tenir un bucle infinit sense un reset automàtic del microcontrolador.

Finalment, la configuració del LVP l'hem usat perquè fem us del bit 5 del port b en sortida. I sense configurar el LVP a OFF, aquest bit queda inhabilitat per el ICSP.

Timer

Com ja hem comentat amb anterioritat, farem ús d'un timer per poder controlar variïs factors del nostre projecte, com venen a ser, l'enviament RF, efectes dels LEDs, etc.

Per fer ús del timer, configurarem el registre T0CON, l'encarregat de controlar el TMR0.
Aquest registre quedarà configurat de la següent manera:

- TMR0ON = 1. Es el bit d'activació del timer zero.
- T08bit = 0. Aquest bit es l'encarregat de definir el número de bits que el comptador utilitzarà. Necessitarem un comptador de 16 bits.
- T0CS = 0. Usarem el oscil·lador extern, per tant aquest bit ha d'anar a 0.
- PSA = 0. No utilitzarem un prescaler per el oscil·lador.
- T0PS2:TPS0 = 0. Donat que no utilitzarem el prescaler, aquets bits els posarem a 0.

```
INIT_TIMER
    ;10001000
    ;88
    movlw 0x88
    movwf T0CON,0
    bcf RCON, IPEN, 0

    ;10100000
    ;E0
    movlw 0xE0
    movwf INTCON, 0

    call RESET_TIMER
    return
```

Un cop hem configurat el T0CON, hem de continuar per el INTCON, el registre encarregat de les interrupcions. Per aquest projecte, necessitarem les interrupcions però no volem una prioritat amb les interrupcions. Per aquest motiu configurarem aquest registre a 1010 0000. Tanmateix, necessitarem el bit de overflow del comptador, per aquest motiu el bit 5 del INTCON, TMR0IE, ha de estar a 1.

Per un altre banda, les interrupcions d'aquest timer han de ser cada 5ms. Per aquest motiu, necessitarem fer un reset cada cop que el comptador fagi overflow. Per dur a terme això, utilitzarem una funció de reset que carregarà en el comptador el valor que 100ns * (valor) = 5ms. Aquest valor haurà de ser 50000.

SIO

La configuració de la EUSART(SIO) es separa en dos parts; Recepció i transmissió.
Utilitzarem 5 registres per tal de configurar la SIO com nosaltres volem.

Recepció

Configurarem la recepció amb el registre RCSTA. Afegirem la següent configuració:

- SPEN = 1. Aquest bit es el enable de la recepció.
- RX9 = 0. No volem tenir una recepció de 9 bits, per tant desactivem aquest bit.
- SREN = 0. Donat que utilitzem la SIO asincronament, no ens importa aquest bit.
- CREN = 1. Volem rebre contínuament, per tant activarem aquest bit.
- ADDEN = 0. Com anteriorment hem vist, no rebrem 9 bits sinó 8, per tant aquest bit no ens afecta.
- FERR = 0. No volem activar el error Farming.
- OERR = 0. Com en el anterior, no volem tenir el error de overrun activat.
- RX9D = 0. Es tracta del bit 9, donat que no l'utilitzarem no ens afecta el que li posem.

Transmissió

Per transmetre, utilitzarem el registre TXSTA amb la següent configuració:

- CSRC = 0. Aquest bit no ens importa ja que ho fem asincronament.
- TX9 = 0. Com en la recepció, enviarem 8 bits no 9, per tant desactivarem l'enviament per 9 bits.
- TXEN = 1. Activarem la transmissió amb aquest bit.
- SYNC = 0. Com que tota la transmissió com la recepció la farem asíncrona, desactivarem aquest bit.
- SENDB = 0. Activarem el sync break quan la transmissió s'hagi completa.
- BRGH = 1. Volem que el nostre baud rate sigui a alta velocitat, per aquest motiu activarem aquest bit.
- TRMT = 1. Per enviar quan estigu buit.
- TX9D = 0. Com ja hem comentat anteriorment, farem una transmissió per 8 bits per tant aquest bit no ens afecta.

Baud Rate

Després de configurar la SIO, necessitem definir el baud rate, es a dir, la velocitat de l'enviament de les dades.
Nosaltres hem definit el nostre baud rate a 19200 bps. Per poder anar a aquesta velocitat ho hem configurat de la següent manera.
EL registre SPBRG li hem de posar en la part baixa, 0x81 i en la part alta tot zeros.

BAUD RATE (K)	SYNC = 0, BRGH = 1, BRG16 = 0											
	Fosc = 40.000 MHz			Fosc = 20.000 MHz			Fosc = 10.000 MHz			Fosc = 8.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	—	—	—	—	—	—	—	—	—	—	—	—
1.2	—	—	—	—	—	—	—	—	—	—	—	—
2.4	—	—	—	—	—	—	2.441	1.73	255	2.403	-0.16	207
9.6	9.766	1.73	255	9.615	0.16	129	9.615	0.16	64	9.615	-0.16	51
19.2	19.231	0.16	129	19.231	0.16	64	19.531	1.73	31	19.230	-0.16	25
57.6	58.140	0.94	42	56.818	-1.36	21	56.818	-1.36	10	55.555	3.55	8
115.2	113.636	-1.36	21	113.636	-1.36	10	125.000	8.51	4	—	—	—

Ja que com podem veure en la següent imatge, la nostre freqüència de oscil·lació va a 40Mhz. Per tant, per anar a 19200bps, necessitarem afegir 129 decimal (81 en hexadecimal) al registre de SPBRG. Hem escollit aquesta configuració ja que podem anar a una velocitat considerable tot tenint un error de 0.16%.

En la part de configuració, es a dir, el registre BAUDCON, l'hem posat tot a zeros ja que:

- No necessitem les dades invertides (RXDTP,TXCKP).
- Com que 129 es pot definir en 1 byte, no necessitarem la part alta del registre SPBRG. I per tant el bit BRG16 ha d'anar a 0.

Els altres bits no son d'importància per aquesta configuració.

Estructuració de memòria

Hem utilitzat la memòria ram per desar el missatge del usuari. Donat que el missatge te una màxima longitud de 300 bytes, hem omplert la memòria ram des de la posició 80h del primer banc fins la posició ACh del segon banc.

Protocols de comunicació

En aquesta fase s'han implementat una sèrie de protocols de comunicació per tal de poder complir amb les especificacions. Ho podem separar generalment en dos apartats.

Microcontrolador – PC

Per aquest protocol hi passaran dos tipus de dades:

- El missatge del usuari que ha introduït en la interfície Java. Aquestes dades començaran a ser enviades tant bon punt el microcontrolador enviï la petició de desar dades o be el propi usuari premi el boto. Ambdós casos, el START_BYTE serà el mateix. Aquestes dades s'aniran desant en la memòria RAM fins que el microcontrolador rebi des de l'ordinador un ENDBYTE.
- Peticions o respostes del ordinador. Tal com hem comentat anteriorment, s'utilitzarà STARTBYTES i ENDBYTES per tal de tenir un control del procés actual. Així doncs hem definit una sèrie de prefixos per establir les diferents peticions o respostes que hi haurà en tot el moment.
    - FLAG_DESAR_MSG -> 0x81. Aquest byte es per la petició del ordinador per desar les dades.
    - FLAG_DESAT_MSG -> 0x85. Cada cop que rebem un byte, confirmarem que l'hem rebut a partir d'aquest byte.
    - FLAG_ENVIAR_RF_MSG -> 0x82. Petició del ordinador per enviar les dades per RF.
    - FLAG_DESAR_SENSE_CONFIRMACIO_MSG -> 0x84. Per acabar, aquest byte envia la petició al ordinador que li envii el missatge.

RF

Aquest protocol es més senzill que el protocol anterior. Esta basat en la codificació Manchester. Es a dir, per un port determinat, enviarem totes les dades des de l'inici de la posició de la ram (80h) fins el número de bytes que tenim.
Aquest protocol a diferencia del anterior que anava a 19200 bps, enviarem les dades a 100bps tenint en compte que el temps de cada bit es de 10ms. Ja que el protocol Manchester defineix que en el cas del 0, amb un duty cycle del 50%, la primera part ha d'anar a 0 i la segona a 1, l'invers en el cas del 1.
Per tant, el temps de 1 i de 0 de cada bit que enviem ha de ser 5ms per complir el duty cycle definit.

Implementació

Després de definir les configuracions que necessitarem i els protocols necessaris, hem continuat a implementar-ho.
Donat que hem implementat aquesta practica en dos entorns diferents, Java i Assembler, hem separat aquest apartat en la implementació per a cada entorn.

Entorn Java

Per dur a terme el disseny que hem proposat que hauria de fer la interfície Java, hem separat en dos grans apartats:

Escolta activa

Aquesta part esta definida en un thread que durant tot el programa, hi ha haurà una escolta constant del port per tal de esperar a rebre una petició del microcontrolador. Quan rebem quelcom, comprovarem que aquest byte es tracta d'una flag coneguda i en cas de ser així, enviarem el missatge que conte el text field de la nostra interfície.

```
while (escolta) {
    try {
        aux = sp.readByte();
        Thread.sleep(2);
        if (aux != 0) System.out.print(aux);
        if (aux == flag) {
            mainWindowController.enviar(true);
            mainWindowController.restart();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Controlador interfície

No ens referim al controlador del MVC del codi Java sinó al fet de l'estructura de comunicació entre la interfície i la pic que es accionada des de la interfície.
Esta separada en dues parts:

Listener boto

Aquesta part, definirà quina acció realitzar depenent del boto que es premi des de la interfície. Es una funció força senzilla que controla quina acció realitzar depenent del el event.

```
public void actionPerformed(ActionEvent e) {
    if(e.getSource() instanceof JButton){
        JButton btn = (JButton) e.getSource();
        try {
            switch(btn.getName()){
                case MainWindowView.BTN_RF:
                    try {
                        sp.writeByte(flag_enviar_rf_msg);
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                    break;
                case MainWindowView.BTN_UART:
                    portThread.setEscolta(false);
                    enviar(false);
                    break;

            }
        } catch (Exception e1) {
            e1.printStackTrace();
        }
        portThread.setEscolta(true);

    }
}
```

Enviar missatge

El propi nom de la funció es força explicatiu.
S'encarregarà d'enviar totes les dades que hi han en el text field de la interfície. Sempre comprovant que el microcontrolador ha rebut el missatge be, ja que com podem veure, cada byte que enviiem rebrem una resposta del microcontrolador i fins que no la rebem no continuarem a enviar el següent. D'aquesta manera ens assegurem que no estem saturant el microcontrolador i per tant no desant les dades correctament.

```
public void enviar(boolean confirmacio){
    try {
        if(model.checkInputText(view.getText())){
            byte resposta;
            byte[] utf8Bytes = view.getText().getBytes("UTF-8");
            if(confirmacio){
                sp.writeByte(flag);
            }else{
                sp.writeByte(flag_desar_msg);
                resposta = sp.readByte();
                while(resposta == 0){
                    resposta = sp.readByte();
                }
            }
            for (byte value:utf8Bytes) {
                sp.writeByte(value);
                resposta = sp.readByte();
                while(resposta == 0){
                    resposta = sp.readByte();
                }
            }
            sp.writeByte(end_byte);
            System.out.print(" Done!\n");

        }else{
            JOptionPane.showMessageDialog(null, "No has escrit cap missatge! ",
            "Error",JOptionPane.ERROR_MESSAGE);
        }

    } catch (Exception e1) {
        JOptionPane.showMessageDialog(null, "Error en enviar les dades!\n "+
        e1.getMessage(), "Error",JOptionPane.ERROR_MESSAGE);

    }
}
```

Entorn Assembler

```
MAIN
    call INIT_VARS
    call INIT_PORTS
    call INIT_SIO
    call INIT_TIMER

LOOP
    btfsc PIR1,RCIF,0
    call REBUT

    btfsc PORTC,3,0
    call POLS_CARREGA_MISSATGE

    btfsc PORTC,4,0
    call POLS_ENVIAR_RF

    btfsc ESTAT, 1,0
    call LEDS_CIRCULAR

    btfsc ESTAT, 0,0
    call ESPERA_POLSADOR

    btfsc ESTAT, 2,0
    call BLINKING_10HZ

    btfsc ESTAT, 3,0
    call BLINKING_5HZ

    goto LOOP
```

Tal com ho hem explicat en el diagrama de mòduls, hem separat el nostre codi en diferents blocs per tant de resoldre petits problemes en blocs i no tot de cop.

I per aquest motiu necessitem un apartat genèric que es dediqui a cridar cada mòdul en el moment que es necessiti. Per aquest motiu tenim aquesta funció.

Es un bucle infinit que es dedicarà a cridar a cada mòdul depenent del estat que es trobi. En els casos de comunicació amb el ordinador, cada cop que rebi informació de la SIO, anirà a REBUT i cridarà a les funcions que siguin necessàries depenent de la informació rebuda.

Per un altre banda, els dos polsadors, s'estaran constantment controlant en cas de rebre una petició de l'usuari. Tant el POLS_CARREGA_MISSATGE com el POLS_ENVIA_RF, realitzaran el que el seu nom indica.

Després de comprovar que no hem rebut una pulsació ni un byte des del ordinador, hem definit uns estats que realitzen una acció diferent en cada cas. Hem utilitzat una variable que depenent del bit que estigui actiu, estarem en un estat u altre.

- Bit 0 Estat. Aquest estat es dedicarà a realitzar la espera que s'ha de fer quan ens premen el botó de carrega missatge i estem esperant al ordinador a que ens enviï una resposta.
- Bit 1 Estat. Si aquest bit esta actiu, vol dir que estarem realitzant la animació dels LEDs circular.
- Bit 2 Estat. En el cas de que aquest bit valgui 1, estarem activant els LEDs a un ritme de 10Hz.
- Bit 3 Estat. Finalment, en aquest estat, estarem fent la mateixa funció que el bit 2 però a una freqüència diferent, 5Hz.

Un cop que hem implementat el bucle principal podem començar a realitzar la part troncal de la practica.

Existeixen dos blocs principals i els altres es dediquen a ajudar al funcionament d'aquests.

Bloc desar

Per accedir a aquest bloc, podrem arribar des de una petició de la interfície o una petició del polsador.
Depenent des de quina part arribem, enviarem una petició de confirmació diferent.

Es important des de un inici posicionar-nos en la posició de la ram que volem començar a posar les dades. A partir d'allà, de manera iterativa, anirem desant cada byte que rebem fins que ens enviïn el END_BYTE.
Aquesta funció a diferencia d'altres, bloqueja tota la PIC fins que no acabi. Ja que no volem realitzar cap altre acció mentre les desem.

Aquest bloc, utilitza funcions del bloc comunicació per tal de enviar les confirmacions al PC com també, modifica variables que es tenen en compte en el bucle principal per tal d'entrar en un estat determinat, com pot ser el de blinking a 5hz.

```
DESA
    call ENVIAR_CONFIRMACIO_DESAR

DESA_SENSE_CONFIRMACIO
    clrf COMPTA_BYTES_L,0
    clrf COMPTA_BYTES_H,0
    clrf ESTAT,0
    clrf FSR0H,0
    movlw POSICIO_A_DESAR_RAM
    movwf FSR0L, 0

BUCLE_DESAR
    btfss PIR1,RCIF,0
        goto BUCLE_DESAR
    movlw END_BYTE
    cpfslt RCREG, 0
        goto RETORNA_DESAR
    movff RCREG, POSTINC0
    incf COMPTA_BYTES_L,1,0
    btfsc STATUS,C,0
    call RESTART_COMPTA
    call ENVIAR_CONFIRMACIO_DESAT
    goto BUCLE_DESAR

RETORNA_DESAR
    movlw ESTAT_BLINKING_5HZ
    movwf ESTAT,0
    clrf TEMPS_UN,0
    return

RESTART_COMPTA
    incf COMPTA_BYTES_H,1,0
    clrf COMPTA_BYTES_L,0
    return
```

Bloc RF

El principal funcionament d'aquest bloc tractarà sobre dos bucles iteratius.
Cada un tindrà unes restriccions de sortida diferents. Cal tenir en compte que aquesta funció estarà bloquejant tota la PIC fins que acabi d'enviar tots els bytes.
El primer bucle, el principal anirà descomptant d'una variable el numero total de bytes enviats fins que aquesta arribi a 0.
En aquest cas, canviarà el estat global perquè estigui en estat de blinking i tornarà al Loop.

En cada iteració d'aquest, hi haurà un bucle iteratiu secundari que s'encarregarà de enviar cada byte. Que com ja hem comentat anteriorment, enviarà cada bit cada 10 ms.

Durant tot el procés, s'aniran activant els diferents leds cada 10 % que s'hagi enviat. Aquest càlcul el rebre a partir del bloc divisor.

```
BUCLE_RESTANT_ENVIAR_RF
    movlw 0x00
    cpfsgt RESTANT,0
    goto PRE_FINAL_RF
    movff POSTINC0, AUXILIAR
    clrf TEMPS_UN,0

BUCLE_ENVIAR_8_BITS
    movlw 0x08
    cpfslt ENVIAT,0
    goto FINAL_BUCLE_ENVIAR
    btfsc TEMPS_UN, 0,0
    goto ENVIAR_BIT_PRIMERA_MEITAT
    btfsc TEMPS_UN, 1,0
    goto ENVIAR_BIT_SEGONA_MEITAT
    goto BUCLE_ENVIAR_8_BITS

FINAL_BUCLE_ENVIAR
    incf INDEX, 1,0
    clrf ENVIAT,0
    decf RESTANT,1,0
    movf RESULTAT_DIVISIO,0,0
    cpfslt INDEX,0
    call ACTIVA_LEDS_RF
    goto BUCLE_RESTANT_ENVIAR_RF

ENVIAR_BIT_PRIMERA_MEITAT
    btfsc AUXILIAR,0,0
    bcf LATC, 5, 0
    btfss AUXILIAR,0,0
    bsf LATC, 5, 0
    goto BUCLE_ENVIAR_8_BITS

ENVIAR_BIT_SEGONA_MEITAT
    btfsc AUXILIAR,0,0
    bsf LATC, 5, 0
    btfss AUXILIAR,0,0
    bcf LATC, 5, 0
    incf ENVIAT,1,0
    rrncf AUXILIAR,1,0
    clrf TEMPS_UN,0
    goto BUCLE_ENVIAR_8_BITS

ACTIVA_LEDS_RF
    incf LEDS,1,0
    call ACTIVAR_LEDS_PROCES
    clrf INDEX, 0
    return
```

Per tal d'efectuar la divisió de 10 sobre el número original, hem arribat a la següent solució.

El nostre objectiu es poder dividir entre 10 de 0 fins a 300, que seran el numero màxim de bytes. Tenint en compte aquest fet hem fet la següent deducció

(Numero de bytes) * (1/10) = resultat

D'aquesta deducció hem de treure la manera de que puguem trobar una manera de dividir entre 10 sense utilitzar taules. Per tant, a partir d'aquesta premissa, podem veure que si busquem un numero que puguem dividir-lo per un numero de base 2 i que aquest sigui aproximadament el mateix valor del numerador, però dividit entre 10, podrem aconseguir un valor aproximat a 0.1.

```
DIVIDIR_10
    clrf RESULTAT_DIVISIO,0
    movlw VALOR_A_MULTIPLICAR
    mulwf COMPTA_BYTES_L,0
    btfsc COMPTA_BYTES_H,0,0
    call SUMA_PART_ALTA
    movff PRODH, RESULTAT_DIVISIO
    rrncf RESULTAT_DIVISIO,1,0
    rrncf RESULTAT_DIVISIO,1,0
    rrncf RESULTAT_DIVISIO,1,0
    movlw 0x1F
    andwf RESULTAT_DIVISIO,1,0
    movlw 0x00
    cpfsgt RESULTAT_DIVISIO,0
    incf RESULTAT_DIVISIO,1,0
    return

SUMA_PART_ALTA
    movlw 0xCC
    addwf PRODH,1,0
    movlw 0x33
    addwf PRODL,1,0
    return
```

La restricció que tenim però, es que no podem superar els 16 bits de resultat i a la vegada intentar tenir el menor error possible.
Fent proves, hem pogut arribar a la solució de que si nosaltres multipliquem per 205 el valor de bytes i després al resultat el dividim per 2048 (Hem de tenir en compte que com que es un numero de base 2, per dividir per aquest valor nomes hem de treure els 11 bits de menor pes).

(Numero de bytes) * (205) = resultat
resultatFinal = ((resultat.High)>>3)

Per aquest motiu, agafem el byte superior i d'aquest mateix rotem 3 cops el valor.
Després de deduir aquest càlcul, hem de comprovar que el error que ens dona aquesta "divisió" es assumible.

Valor real divisió 1/10 = 0.1
Valor aproximat divisió 205/2048 = 0,10009765625
Un error total del 0.009%

Donat que el error es força baix, hem considerat que era acceptable aplicar-ho en aquesta practica.

Bloc LEDS

Els LEDs, a diferencia dels altres blocs, treballen de manera asíncrona. Es a dir, no bloquegen la PIC mentre realitzen una animació en concret.
D'aquesta manera, podem continuar escoltant els polsadors i si ens enviem bytes des del ordinador.

En aquest apartat, el dividirem en dos seccions.

Circular

Per dur a terme l'animació circular, tindrem dos comprovacions principals, si volem anar cap a l'esquerra o dreta.
Donat que son 10 leds i no hi ha un port amb 10 pins, hem tingut que utilitzar dos ports el B i el D.
Depenent del sentit que estiguem anant, anirem comprovant amb el bit de carry del registre status.
I a la vegada rotar el registre LATD i LATB cap al sentit que vulguem.
Com que hi ha poca diferencia entre un sentit i altre hem afegit com a il·lustració del codi la funció de rotar cap a l'esquerra.

```
LEDS_CIRCULAR_ESQUERRA
   movlw 0x00
   cpfsgt LATD,0
   goto LEDS_CIRCULAR_ESQUERRA_LATB
   rlcf LATD,1,0
   bcf LATD, 3, 0
   btfsc STATUS, C,0
   goto LEDS_CIRCULAR_ESQ_SEGONA_PART
   return

LEDS_CIRCULAR_ESQUERRA_DRETA
    bsf INDEX,0,0
    rrcf LATB,1,0
    clrf LATD,0
    return

LEDS_CIRCULAR_ESQUERRA_LATB
   rlcf LATB,1,0
   bcf LATB,LED4,0
   btfsc LATB,6,0
   goto LEDS_CIRCULAR_ESQUERRA_DRETA
   return

LEDS_CIRCULAR_ESQ_SEGONA_PART
   bsf LATB, LED4,0
   clrf LATD,0
   return
```

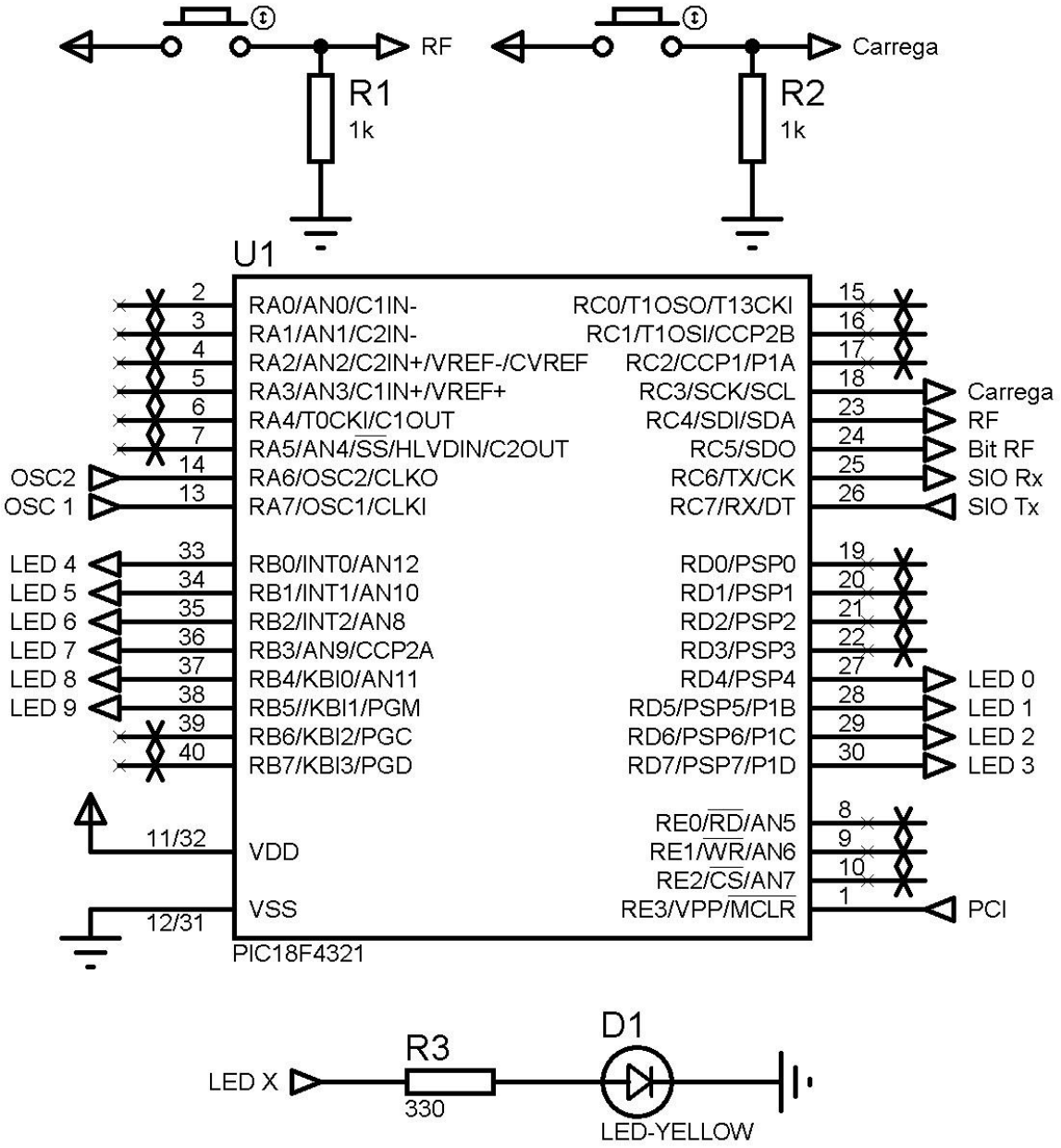
Blinking

Per fer el blinking a diferencia freqüència, hem utilitzat la comprovació amb el timer del temps que estem desde el reset de una variable que ens compta quants ms han passat. Com es mostra en la següent il·lustració. A partir del coneixement que el timer incrementa la variable cada 5mseg.

```
BLINKING_10HZ
    movlw TEMPS_100_MSEG
    cpfseq TEMPS_UN,0 ;Si estem als 100 no els encendrem
    goto COMPROVA_APAGAR_10HZ ;En cas de estar per sota o sobre entrarem aqui
    goto ENCENDRE_LEDS
    return

COMPROVA_APAGAR_10HZ
    movlw TEMPS_200_MSEG
    cpfslt TEMPS_UN,0 ;Si estem en menys de 200 no farem res, sino els apagarem
    goto APAGAR_LEDS
    return
```

Esquemes elèctrics



Problemes observats

Els problemes que hem tingut aquí segurament han tingut a veure amb el fet de que es la practica on més implementació d'assembler hem tingut que fer. Donat que la fase 3 de l'anterior practica, tenia una estructura molt mes senzilla, aquesta hem tingut que conèixer i practicar mes amb el que realment et pot aportar assembler.
Així doncs, errors de comparacions entre variables, l'ús de les constants, configuració de la SIO com també configuracions de ports que han suposat quelcom mes complicat que l'anterior practica.

Per exemple, l'ús del Port B per realitzar els leds ens va portar problemes, donat que utilitzàvem el bit 5 que es corresponia a un bit de programació, que es tenia que des habilitar per poder utilitzar-lo com un I/O digital. Fins que no vàrem descobrir aquesta configuració, ens vam tenir que llegir amb mes atenció el datasheet per poder trobar quina podria ser la solució.

També la comunicació en sèrie del RS-232, com que era tecnologia que no havíem utilitzat fins ara, vam tenir que dedicar un temps per conèixer de quina manera es comunicava i com utilitzar-ho juntament amb la interfície Java.

Conclusions

Un cop realitzat el plantejament de com pensàvem resoldre la practica vam procedir a la corresponent implementació, moment en el qual ens vam adonar que era totalment diferent de la practica anterior. En aquesta practica l'ús del assembler ha estat el element principal. Gracies a aquest fet, els coneixements d'assembler que hem adquirit en aquesta fase han estat molt mes superiors que en l'anterior practica.

Seguint les recomanacions vistes a classe i que ens han donat alumnes d'altres anys, Abans d'implementar el assembler, hem realitzar tot el codi en pseudocodi. Un cop que ens hem assegurat de que aquest te totes les característiques necessàries que necessitem implementar, hem passat a la programació en assembler. Fer el procés d'aquesta manera, aporta una organització superior a l'hora començar la programació a mes baix nivell.

Com a opinió personal ens ha suposat una experiència més que positiva conèixer no sols més perifèrics sinó que hem pogut veure protocol Manchester típicament utilitzat i lo pràctic que pot ser l'ús de cable DB9 per transmissions en sèrie, ja que ens permet transmissions asíncrones de dades.

Els nous coneixements adquirits amb aquesta fase ens permetran embarcar-nos en nous projectes i tenir un ampli coneixement en transmissions en sèrie mitjançant protocols que per nosaltres eren desconeguts però molt utilitzats en el mercat actual, donant-nos així més valor en un futur laboral.