# Pràctiques de Sistemes Digitals i Microprocessadors Curs 2023-2024

## Pràctica 2

## LSChat

| | Login | Nom |
|---|---|---|
| Alumnes | oriol.aparicio | Oriol Aparicio Casanovas |
| | hicham.naf | Hicham Naf |

| | Placa | Memòria | Nota |
|---|---|---|---|
| Entrega | | | |

| | |
|---|---|
| Data | 24/05/2024 |

Còpia per als alumnes

# Pràctiques de Sistemes Digitals i Microprocessadors Curs 2023-2024

## Pràctica 2

## LSChat

| | Login | Nom |
|---|---|---|
| Alumnes | oriol.aparicio | Oriol Aparicio Casanovas |
| | hicham.naf | Hicham Naf |

| | Placa | Memòria | Nota |
|---|---|---|---|
| Entrega | | | |

| | |
|---|---|
| Data | 24/05/2024 |

Portada de la memòria

# Index

# 1. Resum de l'enunciat

Per a realitzar la pràctica de LSChat, implementarem un sistema que ens permeti enviar missatges mitjançant un dispositiu WiFi, entre usuaris connectats a la mateixa xarxa.
Per a implementar la pràctica farem ús d'un LCD, mòdem WiFi ESP8266, teclat matriu, Eusart, joystick i speaker.
Per a utilitzar el sistema també farem ús d'una interfície en Java proporcionada per a fer la pràctica. Aquesta es connectarà al sistema mitjançant la Eusart.
El sistema consistirà en la connexió al mòdem WiFi, mitjançant la introducció del port UDP al que la nostra placa enviarà les dades, el port UDP pel que es rebran les dades i el nom de l'usuari. Un cop fet això i establerta la connexió, disposarem d'un menú amb 6 opcions per a efectuar diverses operacions.
També, en paral·lel podrem rebre missatges que s'enviïn a la nostra placa.

# 2. Plantejament del software

Per a realitzar la pràctica hem implementat un TAD per a cada perifèric, i també altres TAD que o bé ajuden amb la gestió dels perifèrics o bé ajunten diverses funcionalitats per a comunicar els diversos perifèrics entre ells. Seguidament, repassarem els blocs principals dissenyats per a la implementació del nostre sistema.

## 2.1 Teclat Matriu

El tad teclat és un motor que el cridem per llegir les tecles, el que fem és estar constantment fent l'escombrat de les columnes en l'estat 0.

```
case 0:
    if (!HiHaTecla() && columna == 0) {
        LATBbits.LATB2=1;
        LATBbits.LATB4=1;
        LATBbits.LATB6=0;
        columna = 1;
        break;
    }
    else if (!HiHaTecla() && columna == 1) {
        LATBbits.LATB2=0;
        LATBbits.LATB4=1;
        LATBbits.LATB6=1;
        columna = 2;
        break;
    }
    else if (!HiHaTecla() && columna == 2) {
        LATBbits.LATB2=1;
        LATBbits.LATB4=0;
        LATBbits.LATB6=1;
        columna = 0;
        break;
    }
    else if (HiHaTecla()) {
        LATAbits.LATA3 = 1;
        TI_ResetTics(timer);
        state++;
    }
```

En el moment que premem una tecla deixem de fer l'escombrat i mirem quina tecla s'ha premut després de filtrar els rebots. Posem a 1 la variable teclapremuda per indicar que s'ha premut una tecla i esperem al fet que és despremi la tecla i repetim el cilcel.

```
case 1:
    if (TI_GetTics(timer)>=T_DEBOUNCES) {
        state++;
    }
    break;
case 2:
    if (!HiHaTecla()) {
        state = 0;
    }
    else if (HiHaTecla()) {
        teclapremuda=1;
        SorollAlt(GetTecla());
        state++;
    }
    break;
case 3:
    if (!HiHaTecla()) {
        TI_ResetTics(timer);
        fiSoroll();
        teclapremuda=0;
        state++;
    }
    break;
```

**ValorsTeclat:**

Aquest tad té un motor que s'encarrega de convertir les tecles premudes a valors rotatius que es van mostrant per l'LCD.

```
static unsigned char lletres_teclat[10][2] = {
    {' ', ' '},
    {'1', 0},
    {'A', 'C'},
    {'D', 'F'},
    {'G', 'I'},
    {'J', 'L'},
    {'M', 'O'},
    {'P', 'S'},
    {'T', 'V'},
    {'W', 'Z'}
};
```

Tenim una taula amb el primer i l'últim valor de cada tecla, quan premem una tecla anem mostrant el primer caracter de la llista que l'hi toca i l'hi anem sumant 1 mentre no sigui igual al segon valor.

Per exemple, quan premem un 2 per primer cop el que mostrarem és A+0, quan ho premem per segon cop sera A+1 que és un B, per 3r com A+2 que és un C, i per 3r cop A+3 que és una D pero com que és més gran que C reiniciem l'index i mostrem el valor de la tecla que és 2.

A més els anem acomulant en una llista i mostrant per pantalla.

## 2.2 LCD

En l'LCD hem hagut de modificar la funció LcPutString perque te un while i no és cooperativa

```
char LcPutString(char *s) {
// Post: Paints the string from the actual cursor position.
// The coordinate criteria is the same as the LcPutChar.
// Post: Can last up to 40us for each char of a routine output.
    static char state = 0;
    static char numChars = 0;
    static char *string_ptr = 0;
    switch (state) {
        case 0:
            string_ptr = s;
            numChars=0;
            state++;
            break;
        case 1:
            if (*string_ptr&&numChars < 16) {
                LcPutChar(*string_ptr++);
                numChars++;
            } else {
                state++;
            }
            break;
        case 2:
            state=0;
            break;
    }
    return state;
}
```

El que fem és passar-li el punter de la llista de caràcters que volem mostrar, i anar mostrant caràcter per caràcter, mentres tinguem un valor a la llista l'anem mostrant, un cop arrivem al final anem a l'estat 2 que acabem de mostrar.

## 2.3 SIO

Per a poder transmetre i rebre dades a la interfície gràfica del Java, hem hagut d'utilitzar la EUART per a poder comunicar el PIC amb l'ordinador. Per a fer-ho utilitzem les seguents funcions:

```
unsigned char SIO_RXAvail() {
    //Post: Retorna CERT si hi ha una caràcter disponible.
    return ((PIR1bits.RCIF==1)?CERT:FALS);
}

unsigned char SIO_GetChar() {
    // Pre: SIO_RXAvail() ha retornat CERT.
    // Post: Fa lectura destructiva del caràcter rebut.
    return RCREG;
}

unsigned char SIO_TXAvail(void){
    //Post: Retorna CERT si hi ha espai per a enviar un caràcter, FALS en cas contrari.
    return ((PIR1bits.TXIF==1)?CERT:FALS);
}

void SIO_PutChar (unsigned char Valor){
    // Pre: SIO_TXAvail() ha retornat CERT.
    // Post: Posa un nou caràcter a enviament.
    TXREG=Valor;
}
```

Per a saber si al port Rx hi ha algun caràcter disponible per llegir, utilitzem la funció SIO_RXAvail, que retorna cert o fals en cas de tenir o no tenir algun caràcter. Tot seguit, si SIO_RXAvail ens retorna cert en què fem es llegir el caràcter del registre RCREG.
Per al Tx farem mes o menys el mateix, utilitzarem la funció SIO_TXAvail, per a saber si podem enviar un caracter per la Euart. Llavors, la funció SIO_PutChar, en cas que SIO_TXAvail retorni cert, el que farà es posar un nou caracter a enviar per el registre TXREG.

## 2.4 Mòdem ESP8266 Wi-Fi

Per poder-nos comunicar amb el mòdem wifi, hem hagut de crear un altre SIO personalitzada perquè només en tenim una per poder comunicar-nos amb el Java.

Com que el Mòdem Wifi pot enviar moltes trames i hem de poder llegir-les bé totes de cop, hem decidit fer que el baudrate d'aquesta SIO Custom, hem fet que vagi a 500 bauds(perque els valors ens quedaven més exactes que amb 300 baudrate).

Per fer-ho, com que el nostre temps de tic és de 100 microsegons, si volem aconseguir un baud rate de 500, el nostre temps de bit és de 0.002 segons, que vol dir que necesitem comptar 20 tics per cada bit.

### 2.4.1 RxCustom

S'encarrega de rebre trames en codi ASCCI dell Mòdem Wifi, aquest tad tè 2 funcions que son cridades, i un motor que s'executa al main.

```
char GetRxMSG(void){
    flag_RX=0;
    return MSG_AUX;
}

char GetFlag(void){
    return flag_RX;
}

void MotorRX(void) {
    static char state;
```

La primera funció GetRxMSG la cridem per obtenir el simbol llegit i posar a 0 el flag per indicar que hem fet la lectura , el segon GetFlag l'utilitzem per mirar l'estat del flag de l'RX, si retorna 0 és que no hi ha nou missatge, si hi val 1 vol dir que tenim un missatge disponible i podem fer la lectura.

En el MotorRX, el que fem és monitoritzar l'entrada B7 que és on tenim l'RX del Modem Wifi, si val 1 en quedem esperant en l'estat 0 i quan val 0 que vol dir que inicia el start bit, comensem a comptar mig bit per posicionar-nos al mig de cada bit.

```
switch(state) {
    case 0:
        if (PORTBbits.RB7==1){
            TI_ResetTics(timer);
        }
        else  {
            if (TI_GetTics(timer)>=10){
            state=2;
            LATAbits.LATA4=0;
        }
        }
        break;
```

Un cop passat mig bit, anem a l'estat 2 que allà ens esperem 1 bit sencer per saltar-nos el start bit que no ens interessa i posar-nos al primer bit.

```
case 2:
    if (TI_GetTics(timer)>=30 + (nMSG*20)) {

        state --;
    }
    break;
```

El reset dels tics del timer només el fem a l'estat 0, per minimitzar l'error de lectura cada cop que fem un reset tics.

Tornem a l'estat 1 on anem acumulant els bits en un Char anomenat MSG_AUX i comptant quants bits anem llegint, mentres sigui menor a 8 la variable nMSG, anem anant a l'estat 2.

```
case 1:
    if ( nMSG < 8 ) {
        MSG_AUX = (MSG_AUX>>1) | (PORTB & 0x80);
        nMSG++;
        state++;
    } else {
        nMSG=0;
        state=3;
    }
    break;
```

Finalment, quan ja hem acumulat els 8 bits, anem a l'estat 3 on esperem un bit sencer per posar-nos sobre del stop bit abans de tornar a l'estat 0 i tindre errors de lectura. Activem el flag de lectura per indicar que tenim un nou missatge i a la vegada l'enviam pel TX de la SIO perquè qualsevol cosa que rebem pel wifi ho passem al Java directament.

```
case 3:
    if (TI_GetTics(timer)>=190) {

        if (inMSG[index] == MSG_AUX){
            index++;
            if (index == 4){
                index=0;
                EQM_MSG();
            }
        }else{
            index=0;
        }
        SIO_PutChar(MSG_AUX);
        flag_RX=1;
        state = 0;
        TI_ResetTics(timer);
    }
break;
```

Aprofitem aquest motor per poder detectar els missatges que ens arriben, en l'estat 3 quan ens arriba una seqüència de missatges del següent format, l'hi indiquem a l'EQM que està arribant un missatge.

```
inMSG[0] = '+';
inMSG[1] = 'I';
inMSG[2] = 'P';
inMSG[3] = 'D';
```

### 2.4.2 TxCustom

En aquest tad el que fem és enviar els missatges cap al Modem Wifi, és similar que la RxCustom, però en lloc d'anar acumulant els bits en un Char, els anem llegir.

Tenim 2 funcions que cridem per consultar i un motor, el SetMSG l'hi passem el caràcter que volem enviar i posem a 0 el flag d'enviament per saber si hi ha un missatge enviant-se en curs o no, i iniciem en motor del Tx Custom posant a 1 el start perquè comenci a enviar.

El GetEND el fem servir per consultar el flag abans d'enviar un missatge, si val 0 vol dir que s'està enviant un missatge i ens esperem, si val 1 vol dir que ha acabat i podem passar-li un altre.

```
void SetMSG(unsigned char new_msg){
    MSG =new_msg;
    end_TX=0;
    start=1;
}

char GetEND(void){
    return end_TX;
}

void MotorTX(void) {
    static char state = 0;
```

En el MotorTX, en l'estat 0 ens esperem al fet que start es posi a 1 per començar a enviar un missatge, quan val 1 anem a l'estat 1 per enviar el start bit posant a 0 la sortida C5 i ens esperem 1 temps de bit i després anem a l'estat 2.

```
case 0:
    if (start == 1) {
        TI_ResetTics(timer);
        LATCbits.LATC5=0;
        state++;
    }
    break;
case 1:
    if (TI_GetTics(timer)>=20 + (20*nMSG)) {
        state++;
    }
    break;
```

En l'estat 2 anem llegint el char MSG on tenim el caràcter que volem enviar, utilitzem una màscara per quedar-nos amb el primer bit i anem desplaçant el character cap a la dreta per tindre sempre el bit que volem enviar a la posició 0.

Comprovem si val 1 o 0 per posar-ho a la sortida i ens n'anem a l'estat 1 per esperar 1 temps de bit, això ho fem 8 cops abans d'anar a l'estat abans d'anar a l'estat 3 per fer el stop bit i activar el flag per indicar que s'ha acabat d'enviar, posar a 0 el start i tornar a l'estat 0 per esperar el pròxim caràcter.

```
case 2:
    if (nMSG < 8) {
        MSG_AUX = 0x01 & (MSG);
        MSG = (MSG>>1);
        nMSG++;
        state--;
        if (MSG_AUX == 0){
            LATCbits.LATC5=0;
        }
        else {
            LATCbits.LATC5=1;
        }
    }
    else {
        LATCbits.LATC5=1;
        nMSG=0;
        state++;
    }
    break;

case 3:
    if (TI_GetTics(timer)>=220) {
        start = 0;
        state=0;
        end_TX=1;
        TI_ResetTics(timer);
    }

break;
```

## 2.5 Comunicació

Hem creat un tad per poder enviar llistes de caràcters pel tx costum més còmodament, la idea principal és passar-li un punter de la llista i ell s'encarregui d'enviar-ho i quan acabi ens avisi.

Això ho fem amb la següent funció que cridarem cada vegada que vulguem enviar un missatge:

```
char MotorCOM(char* new_msgC) {
    static char state = 0;
    static char index = 0;
    switch(state) {
        case 0:
            msgC = new_msgC;
            state++;

            break;
        case 1:
            if (GetEND()==1) {
                SetMSG(*msgC);
                *msgC++;
                state++;
            }
            break;
        case 2:
            if (*msgC) {
                state--;
            }
            else {
                state++;
            }
            break;
        case 3:
            if (GetEND()==1) {
                state++;
            }
```

El que fem és passar-li la llista de caràcters que vulguem enviar:

```
case 1:
    if(MotorCOM(LIST)==4) {
        state_option++;
    }
    return 0;
```

Ens esperem que ens retorni un 4 que és l'estat on ha acabat d'enviar.

Comença en l'estat 0 on copia el punter de la llista a una variable de tipus punter interna.
Tot seguit va cap a l'estat 1 on comprova primer que el TX no està ocupat, l'hi passem el primer caràcter i augmentem en 1 el punter.
Anem a l'estat 2 on comprovem que el contingut actual existeix, ja que per defecte el codi afegeix \0 al final dels textos, si encara existeix tornem a l'estat 2 per enviar-ho, si ha acabat anem a l'estat 3 per esperar al fet que acabi d'enviar, no caldria, però ens dona-va problemes si no ho fèiem. Tot seguit anem a l'estat 4 que retorna a l'estat 0.

## 2.6 Joystick

Per a convertir el valor analògic del joystick que rebem utilitzem el port RA0, que ens permet llegir inputs analògics. Per a començar una conversió utilitzem la funció startConversion que indica que podem llegir un valor del port i per a finalitzar la lectura utilitzarem la funció finishConversion.

```
void startConversion(void) {
    //Pre: ---
    //Post: inicia la conversió d'un canal analogic.
    ADCON0bits.GODONE=1;
}

char finishConversion(void) {
    //Pre: s'ha cridat la funci startConversion().
    //Post: retorna CERT si ha acabat la conversio i viceversa.
    return ((ADCON0bits.GODONE==0)?CERT:FALS);
}
```

Un cop iniciem la conversió, anirem a la funció mouJoyAvail, que depenent del valor que hàgim rebut, i en funció del rang definit, decidirem si moure amunt, avall o no fer cap acció. Això es retornarà a una variable que definirà que fem.

```
char mouJoyAvail(void) {
    //Pre: la funcio finishConversion() ha retornat CERT.
    //Post: retorna CERT si hi ha una accio del joystick i viceversa.
    if (ADRESH < 5 && estatAnterior== JOYSTICK_CENTER) {
        estatAnterior=JOYSTICK_DOWN;
        return CERT;
    }
    else if (ADRESH > 245 && estatAnterior == JOYSTICK_CENTER) {
        estatAnterior=JOYSTICK_UP;
        return CERT;
    }
    else if (ADRESH > 5 && ADRESH < 245) {
        estatAnterior=JOYSTICK_CENTER;
        return FALS;
    }
    return FALS;
}
```

Finalment la funció getAction retornarà l'estat en el que es troba el joystick.

```
char getAction(void) {
    //Pre: la funcio mouJoyAvail() ha retornat CERT.
    //Post: retorna l'accio que cal fer.
    return estatAnterior;
}
```

## 2.7 Altaveu

Per fer sonar els tons, tenim aquest motor que el cridem en el main i consta de 2 estats, constantment va fent el PWM amb la freqüència que l'hi indiquem i mentres sound val 1.

```
void MotorAltaveu(void) {
    static char state = 0;

    switch(state) {
        case 0:
            if (TI_GetTics(timer)*2>=dutycycle) {
                TI_ResetTics(timer);

                LATAbits.LATA4=1;
                state++;
            }
            break;
        case 1:
            if (TI_GetTics(timer)*2>=dutycycle && sound==1) {
                TI_ResetTics(timer);
                LATAbits.LATA4=0;
                state--;

            }
            break;
    }
}
```

L'hi indiquem la freqüència i activem el só amb la següent funció:

```
void SorollAlt(unsigned char tecla) {
    if(tecla=='*')  {dutycycle=300;}
    if(tecla=='#')  {dutycycle=950;}
    if(tecla==0)    {dutycycle=1300;}//
    if(tecla==1)    {dutycycle=900;}
    if(tecla==2)    {dutycycle=1100;}//
    if(tecla==3)    {dutycycle=2600;}//
    if(tecla==4)    {dutycycle=1000;}
    if(tecla==5)    {dutycycle=800;}//
    if(tecla==6)    {dutycycle=750;}
    if(tecla==7)    {dutycycle=2900;}
    if(tecla==8)    {dutycycle=400;}
    if(tecla==9)    {dutycycle=3100;}//
    sound=1;
}
```

I per parar ho fem amb la seguent:

```
void fiSoroll(void) {
    sound=0;
}
```

## 2.8 EQM (El Que Manda)

Tenim un motor principal que fa tot el funcionament de la pràctica, està dividit en vàries funcions per simplificar i que sigui tot més facil d'identificar.

El MotorEQM és el que es crida en el main, està dividit en 2 parts, una és quan start val 1 que es fa el funcionament normal, quan val 0 vol dir que estem rebent un missatge i aném a executar la funció ReciveMSG.

```
void MotorEQM(void) {
    static unsigned char state = 0;

    if (start==1){
    switch(state) {
        case 0:
            if (start == 1) {
                maxMSG = 0;
                lastMSG = 7;
                Opt3 = 0;
                Opt4 = 0;
                INTENTS=0;
                port[4]=',';
                port[9]='\0';
                state++;
            }
            break;
```

El que fem primer és inicialitzar les variables necesaries.

```
case 1:
    if (ConfWIFI()==1){
        if(wifiSTATE==1){
            startMenu();
            state_option=0;
            state++;
        }else{
            state--;
        }
    }
    break;
```

En l'estat 1 executem la funció que s'encarrega de la connexió wifi ConfWIFI, quan retorna 1 vol dir que ha obtingut un resultat, després mirem si la connexió ha sigut exitosa o no amb la variable wifiSTATE.
Si val 1 vol dir que la connexió ha set exitosa i activem el motor del menú i anem a l'estat 3, però si no tornem a l'estat 0 per refer la connexió.

**ConfWIFI:**

S'encarrega de la connexió del wifi, en l'estat 0 netegem l'LCD, en l'estat 1 mostrem el missatge que ens introdueixin un port destí i cridem la funció StartRead del tad ValorsTeclat que l'hi indiquem la mida del text que volem llegir.
En l'estat 2 esperem a que el motor ValorsTeclat acabi de llegir i ens guardem el resultat a la variable port.

```
] char ConfWIFI(void){
    switch (state_option){
        case 0:
            LcClear();
            LcGotoXY(0,0);
            state_option++;
            return 0;
        case 1:
            if (LcPutString("DEST.PORT: ") == 2) {
                StartRead(4);
                state_option++;
            }
            return 0;

        case 2:
            if (ValorsState()==0) {
                port[0] = getMSG(0);
                port[1] = getMSG(1);
                port[2] = getMSG(2);
                port[3] = getMSG(3);
                LcClear();
                LcGotoXY(0,0);
                state_option++;
            }
            return 0;
```

Fem el mateix pel Reciver port i l'user name amb l'única diferència que aquest últim la mida del text a introduir és de 3.

```
case 7:
    if (MotorCOM(CIPCLOSE)==4){
        index = 0;
        state_option++;
    }
    return 0;
case 8:
    if (GetFlag()==1){
        index++;
        msg= GetRxMSG();
        state_option++;
    }
    return 0;
```

Tot seguit enviem la comanda per finalitzar la connexió al wifi, i esperem la resposta d'aquest.

```
    case 11:
        if (MotorCOM("AT+CIPSTART=\"UDP\",\"255.255.255.255\",\0")==4){
            //endMSG();
            state_option++;
        }
        return 0;
    case 12:
        msgToSend =(char*)port;
        if (MotorCOM(msgToSend)==4){
            state_option++;
        }
        return 0;
    case 13:
        if (MotorCOM("\r\n\0")==4){
            comptSEC=0;
            TI_ResetTics(timer);
            index=0;
            state_option++;
        }
        return 0;
```

Al finalitzar enviem la comanda per establir la connexió i els valors dels ports que hem introduït. Tot seguit ens esperem al fet que s'estableixi la connexió. Reenviem com a màxim la trama 3 cops si no s'aconsegueix establir, si no rep resposta en aquests 3 intents reiniciem el codi.

**ListAccesPoints:**

El que fem és mostrar Scanning i enviar la trama per a la llista i esperar que acabi.

```
char ListAccesPoints(void) {

    switch(state_option) {
        case 0:
            msgToLCD =(char*)SCAN;
            if(LcPutString(msgToLCD) == 2){
                state_option++;
            }
            return 0;

        case 1:
            if(MotorCOM(LIST)==4) {
                state_option++;
            }
            return 0;
        case 2:
            if(SIO_RXAvail()==1){
                msg = SIO_GetChar();
                state_option=0;
                return 1;
            }
            return 0;
    }
}
```

**ShowConnStatus:**

El que fem és mostrar per l'LCD Scanning, tot seguit enviem la trama i esperem la resposta i la guardem a la variable statisSTATE.

```
char ShowConnStatus(void) {
    switch(state_option) {
        case 0:
            msgToLCD =(char*)SCAN;
            if(LcPutString(msgToLCD)==2) {

                state_option++;
            }
            return 0;
        case 1:
            if(MotorCOM(STATUS)==4){
                state_option++;
            }
            return 0;
        case 2:
            if(SIO_RXAvail()==1){
                msg = SIO_GetChar();
                if(msg == 0x06){
                    statusSTATE=1; //Indiquem que l'estat de la connexio és correcte
                } else {
                    statusSTATE=0; //Indiquem que l'estat de la connexio és incorrecta
                }
                state_option = 0;
                return 1;
            }
            return 0;
    }
```

**CloseConn:**

Primer netejem la pantalla LCD, tot seguit mostrem el missatge amb el nom de l'usuari i ens esperem 3 segons.

```
char CloseConn(void) {
    switch(state_option) {
        case 0:
            LcClear();
            LcGotoXY(0,0);
            state_option++;
            return 0;
        case 1:
            if(LcPutString("bye bye ")==2){
                state_option++;
            }
            return 0;
        case 2:
            if(LcPutString(user)==2){
                state_option++;
            }
            return 0;
        case 3:
            LcPutChar('!');
            state_option++;
            TI_ResetTics(timer);
            return 0;
        case 4:
            if(TI_GetTics(timer)>20000){      //2seg
                LcClear();
                LcGotoXY(0,0);
                return 1;
            }
```

EnterChatMode:

S'encarrega d'enviar els missatges, primer indiquem la mida del missatge que volem llegir que es de 10 màxim  a mesura que el motor ValorsTeclat els va mostrant per pantalla.

```
case 0:
    LcClear();
    LcGotoXY(0,0);
    StartRead(10);
    state4++;
    return 0;
case 1:
    if (ValorsState()==0) {
        if(GetTecla()=='*'){
            state4 = 0;
            return 1;
        }
        state_option=1;
        state4++;
    }
    return 0;
```

Tot seguit comprovem l'estat de la connexió, si no es pot establir mostrem un missatge d'error i tornem a l'inici.

```
        return 0;
    case 2:
        if (ShowConnStatus()==1){
            if (statusSTATE==1){
                state4=6;
            } else {
                state4++;
                state_option=7;
                Opt4=1;
            }
        }
        return 0;
    case 3:
        if(ConfWIFI()==1) {
            if (wifiSTATE==1){
                state4=6;
            } else{
                state4++;
            }
        }
        return 0;
    case 4:
        if(LcPutString("ERROR...F")==2){
            TI_ResetTics(timer);
            state4++;
        }
```

Si tot és correcte procedim a enviar la trama i tornar a l'estat inicial per llegir el següent indefinidament fins que introduïm un '*' per teclat.

**ListLastMessages:**

S'encarrega de visualitzar els missatges, aquests missatges els guardem a una array de 2 dimensions 8x14, on guardem 8 missatges.

Tenim 2 variables per saber l'estat d'aquesta llista una és maxMSG que ens indica el valor màxim de missatges que tenim, quan val 0 aquesta funció mostrara NO MSGS, però si val més mostrarem els missatges acord a això.
L'altre és lastMSG, que ens indica on està l'últim missatge, ja que com a molt podem guardar 8 missatges i si ens arriben de més sobreescrivim la llista i hem de saber en quina posició està l'últim missatge en tot moment.

En aquesta funció cridem funcions de lectura del Joystic i del teclat per poder desplaçar-nos per la llista de missatges i del teclat esperar prémer '*' per tornar a l'inici.

**ReciveMSG:**
Quan rebem un missatge parem tot el que estàvem fent i comencem a guardar el missatge que rebem, el missatge comença a partir dels ':' així que tot el que ve abans no ens interessa, per aquest motiu en l'estat 1 ens esperem a aquest missatge abans de començar a guardar.

```
void ReciveMSG(void){
    static unsigned char stateMSG = 0;
        switch(stateMSG) {
            case 0:
                if(StateMenu()==1){
                    stopMenu();
                    resetMENU=1;
                }else{
                    resetMENU=0;
                }
                stateMSG++;
                break;
            case 1:
                if (GetFlag()==1) {
                    msg=GetRxMSG();
                    if (msg==':'){
                        stateMSG++;
                        index=0;
                    }
                }
                 break;
```

Anem acumulant el missatge en la llista, en l'índex que l'hi toca, sabem que hem d'acabar de guardar quan rebem un '\r'. Si el missatge te una llargada menor a 10 caracters, afegim un '*' per saber on acaba.

```
    case 2:
        if (GetFlag()==1 ) {
            msg=GetRxMSG();

            if(msg=='\r'){
                if(index!=13){
                    msgs[lastMSG][index] = '*';

                }
                stateMSG++;
            }else{
                msgs[lastMSG][index] = msg;
                index++;
            }
            }
        break;
```

Tot seguit mostrem el missatge guardat i fem 5 segons de tons.

```
case 3:
    LcClear();
    LcGotoXY(0,1);
    index=0;
    stateMSG++;
    //SorollAlt(5);
    //SIO_PutChar('*');
    break;
case 4:
    if(index < 14 ){
        if(msgs[lastMSG][index] != '*'){

            LcPutChar(msgs[lastMSG][index]);
            index++;
        }else {
            index=20;

        }
    }else{
        SorollAlt(4);
        comptSEC=0;
        stateMSG=5;
    }
    TI_ResetTics(timer);
    break;
```

## 3. Configuracions del microcontrolador

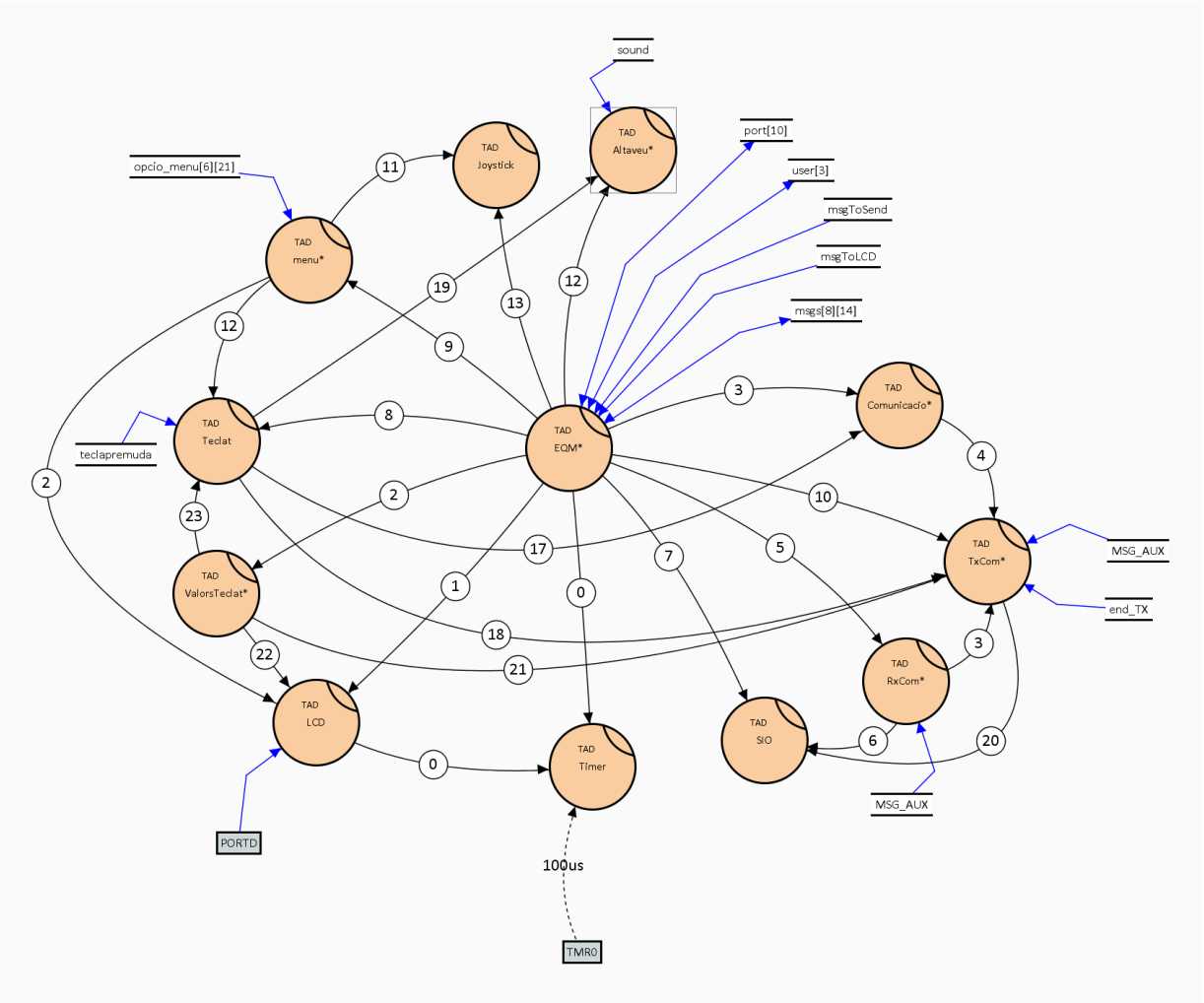
Per a configurar el Timer, l'hem configurat perquè funcioni a 100us, amb una FOSC de 40MHz. El T0CON l'hem configurat sense prescaler i l'hi carreguem un 64536 cada vegada que salta la interrupció perquè compti 1000 que equival a 100us.
És a dir que el nostre temps de tic és de 100us

```
#include <xc.h>
#include "picl8f4321.h"
#include "timer.h"

// Definicions, per interrupci? cada 100 us.
#define T0CON_CONFIG 0x88
//#define RECARREGA_TMR0 64536        // 100 us, suposant FOsc a 40MHz.
#define RECARREGA_TMR0 64536        // 100 us, suposant FOsc a 40MHz.
#define TI_NUMTIMERS 12             // Nombre de timers virtuals gestionats per aquest TAD. Si ca
```

Per a la selecció de ports, hem dedicat el PortB al teclat matriu, ja que era el port on teníem tots els espais lliures i per a connectar tots els pins del teclat era el que millor ens anava.
Per a connectar el LCD hem posat del port RD4 al RD6 el Rs, Rw i e, en aquest ordre. Del RD0 al RD3 hem connectat el d0 al d3 respectivament.
Per a la SIO, hem utilitzat els pins que ja estan directament soldats a la placa que són, per al RC6 el TX i pel RC7 el RX.
Per al wifi custom, hem utilitzat el RC5 per al RX i el RD7 per al TX.
Finalment per al Joystick, hem utilitzat el RA0, ja que és el primer port analògic que tenim disponible, sense haver de configurar els altres. El speaker, l'hem connectat el RA4.

## 4. Diagrama de TADs

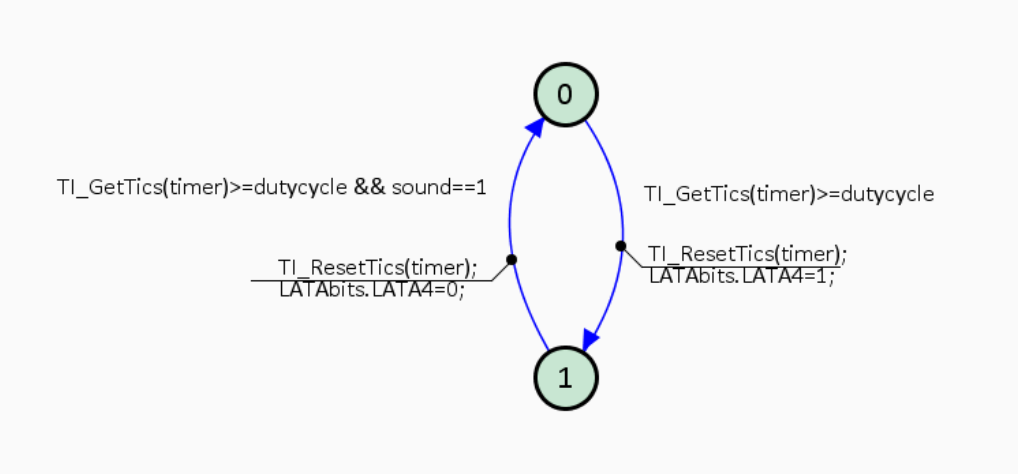



Per al diagrama de TADs, tenim un total de 12 TADs diferents, dels quals 8 d'ells tenen motor. El TAD principal serà el de EQM (El Que Manda), que serà el que gestionarà tots els diferents TADs i serà l'encarregat de gestionar les opcions i guiar els altres TAD, per a saber qui ha de fer que.

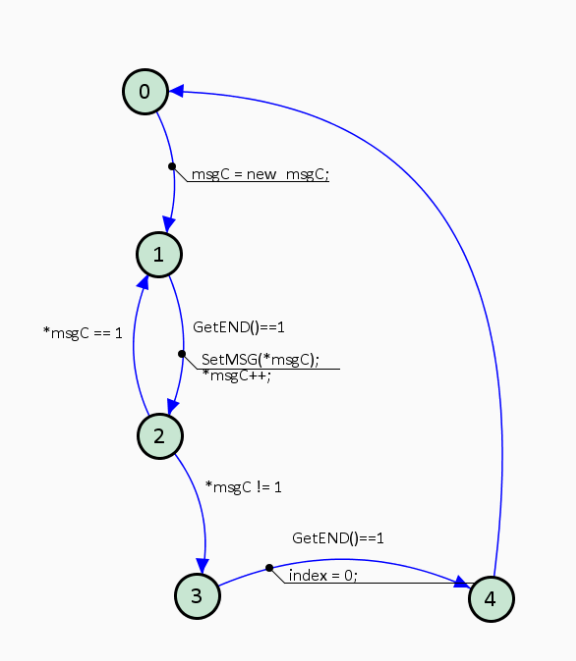
Cal tenir en compte que tots els TAD, a excepció del TAD sio i Joystick, tenen una fletxa cap al timer, que no s'ha posat per fer més net el diagrama

## 5. Motors dels TADs

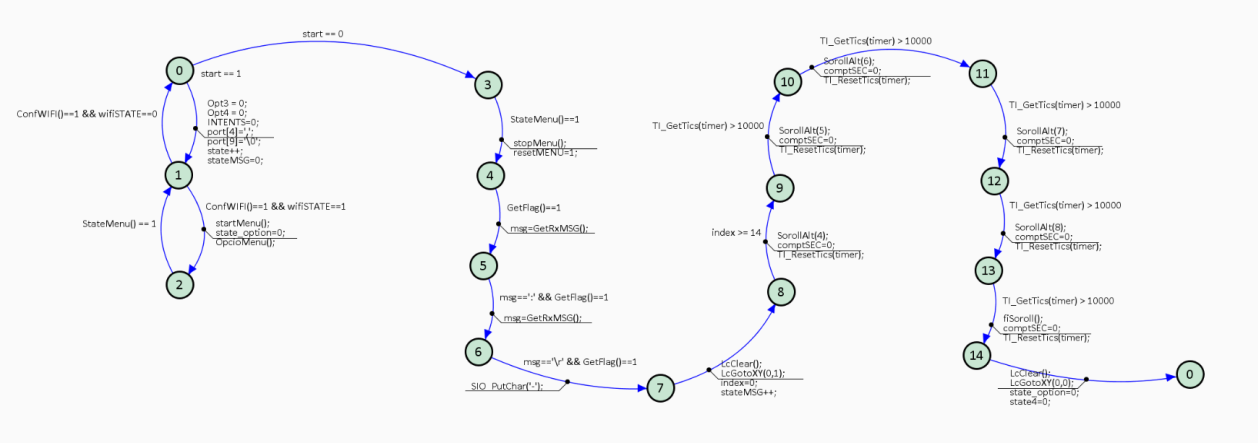
Motor Altaveu:



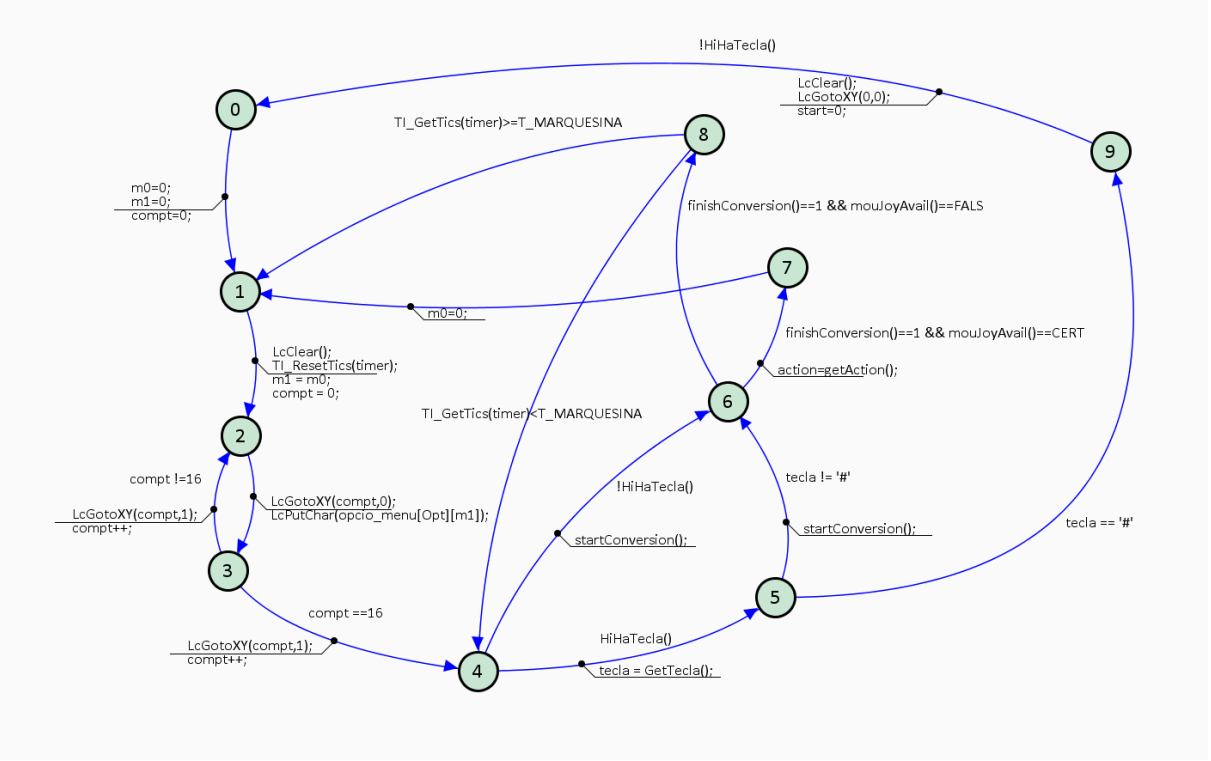

Motor Comunicació:



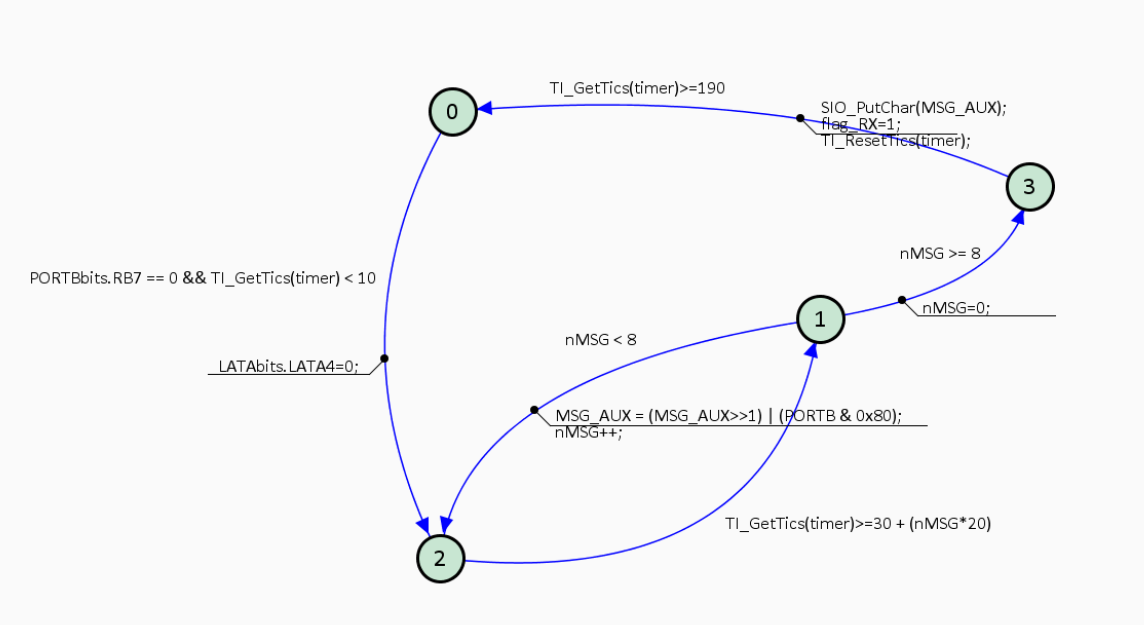

Motor EQM:



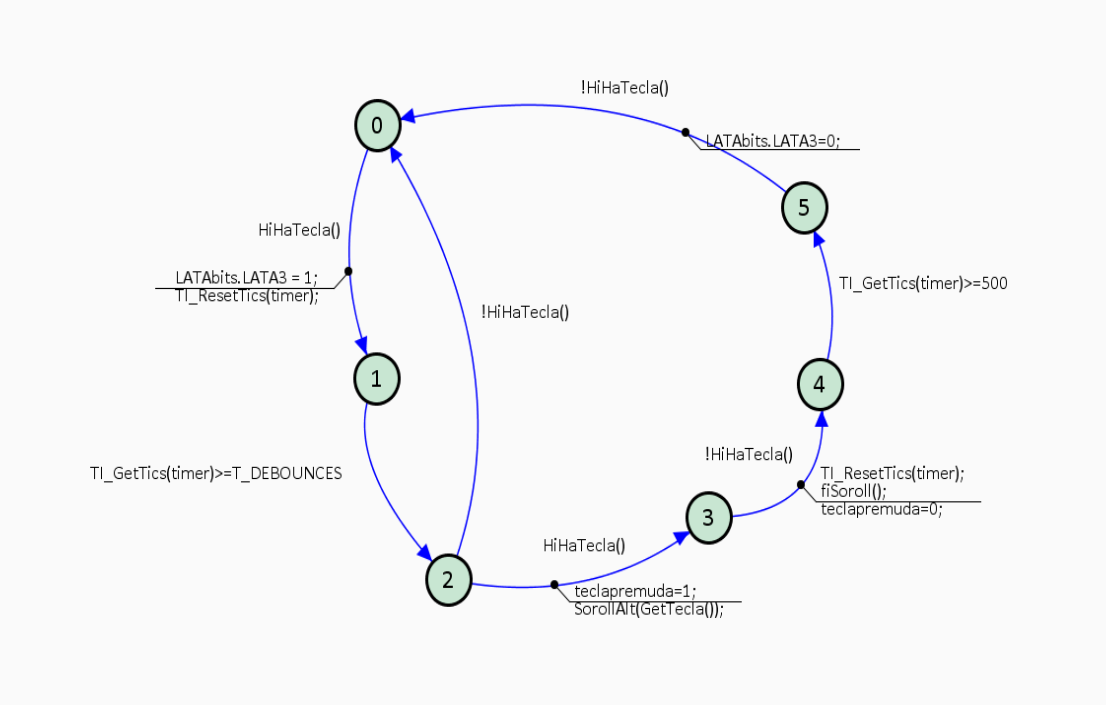

Motor Menu:



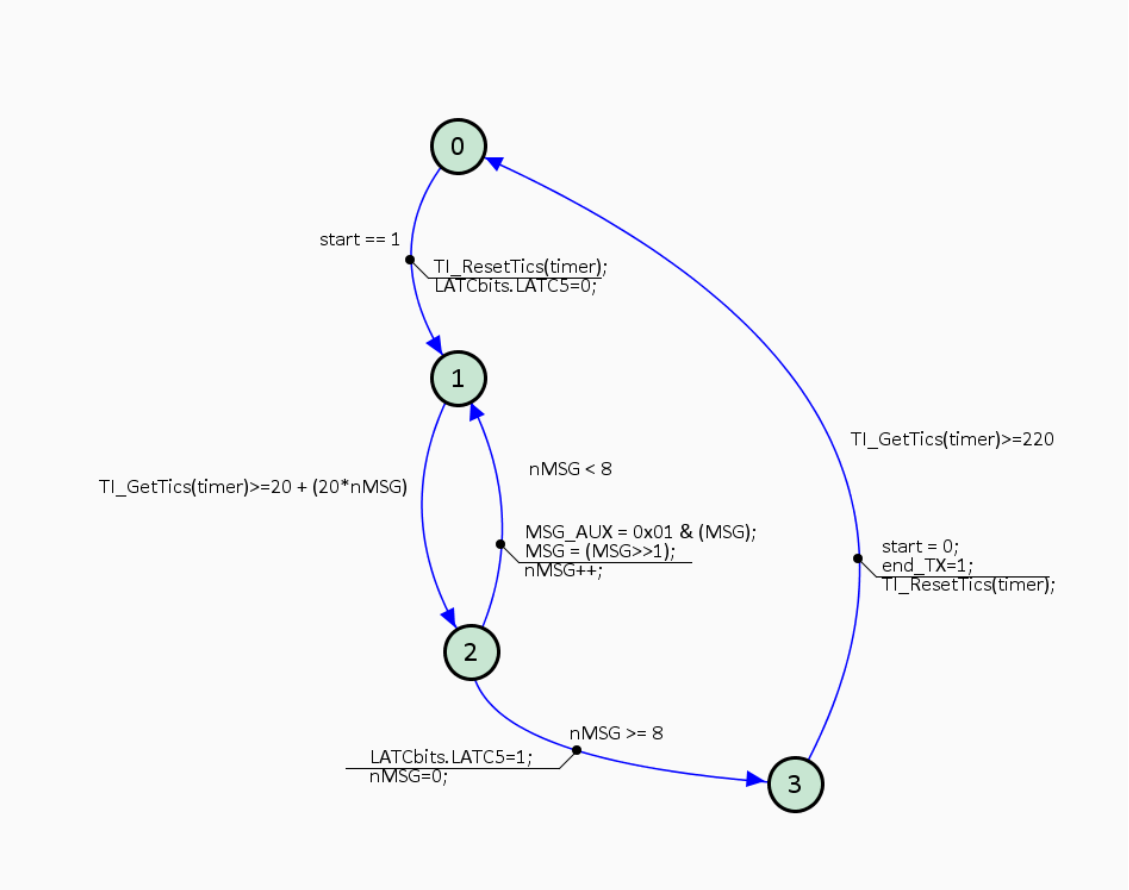

Motor RX:



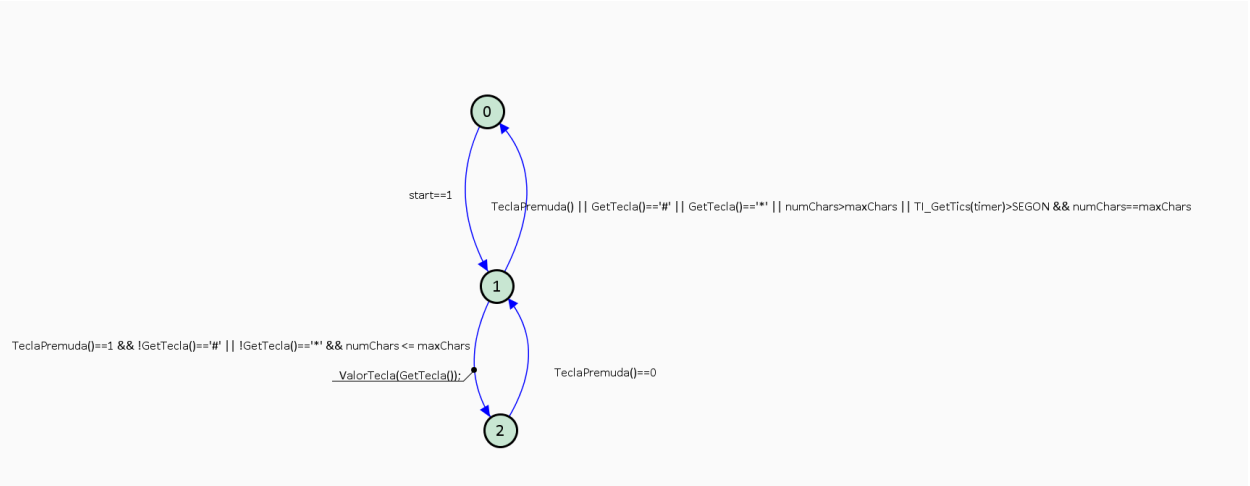

Motor Teclat:



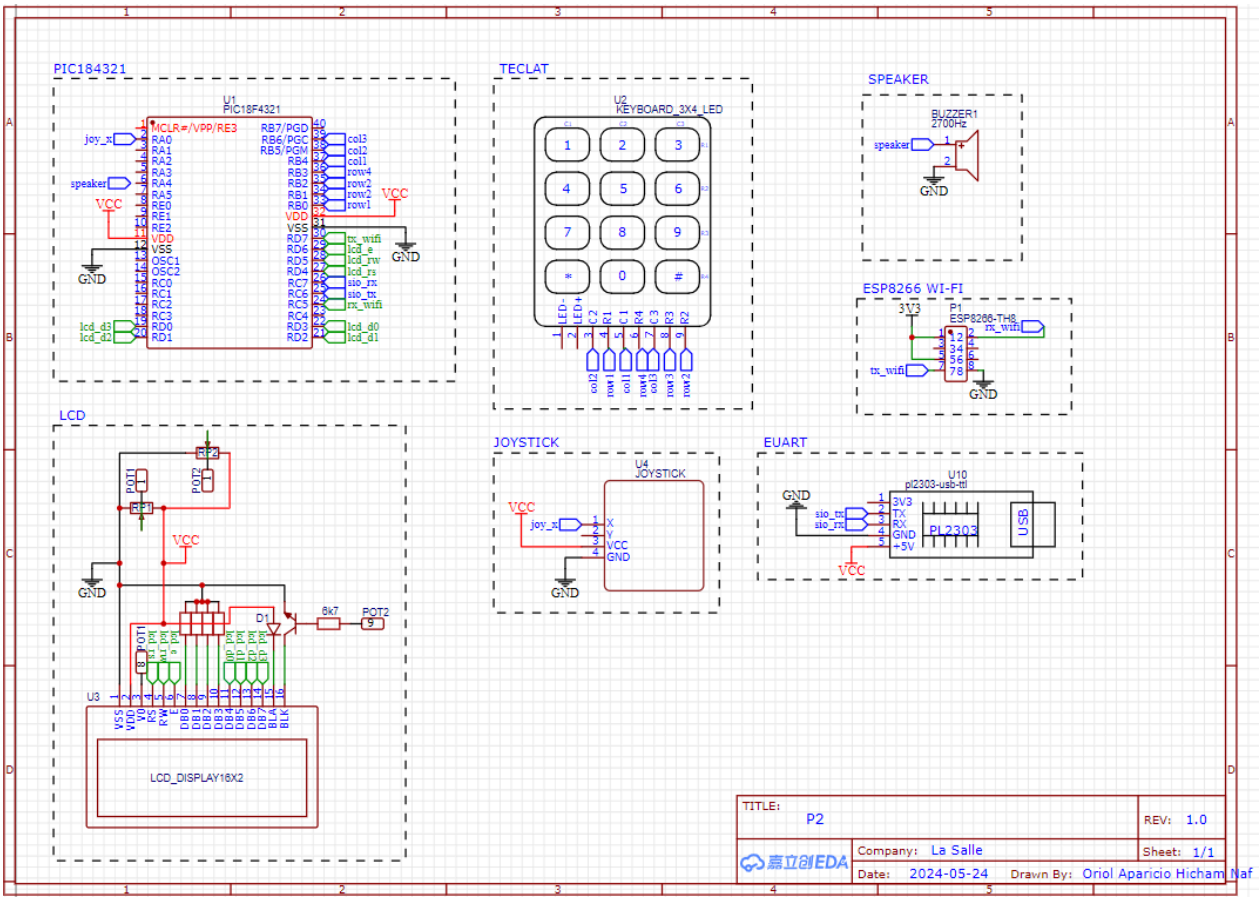

Motor TX:




Motor Valorsteclat:



## 6. Esquema elèctric




## 7. Conclusions i problemes observats

Els principals problemes que hem tingut han estat relacionats amb l'apartat de la memòria del programa, ja que al ser una pràctica amb tants TADs hem hagut d'optimitzar al màxim totes les funcions i Motors perquè ocupessin en mínim de memòria possible. Tot i això, hem d'utilitzar l'optimitzador S del MPLAB per a poder fer posar tot el programa.

Un altre problema ha estat relacionat amb el mòdul ESP8266 Wi-Fi, ja que hem hagut d'utilitzar un baudrate de 500 per a assegurar que rebíem tota la informació correctament, i ens donaven problemes amb la connexió, i per això hem hagut d'actualitzar el mòdul de tal manera que funciones.

Per la resta hem estat capaços de resoldre tota la practica amb tots els requisits demanats.

# 8. Planificació

**Hores estimades i planificació:**

| | Hores estimades | 01/04/24 | 08/04/24 | 15/4/24 | 22/4/24 | 29/4/24 | 6/5/24 | 13/5/24 |
|---|---|---|---|---|---|---|---|---|
| Comprensió de l'enunciat | 1 | | | | | | | |
| Disseny sobre paper | 5 | | | | | | | |
| Programació MPLAB | 15 | | | | | | | |
| Soldar | 1 | | | | | | | |
| Debugar | 5 | | | | | | | |
| Realització de la memòria | 3 | | | | | | | |

**Hores totals dedicades:**

| | Hores estimades | 01/04/24 | 08/04/24 | 15/4/24 | 22/4/24 | 29/4/24 | 6/5/24 | 13/5/24 |
|---|---|---|---|---|---|---|---|---|
| Comprensió de l'enunciat | 1 | | | | | | | |
| Disseny sobre paper | 7 | | | | | | | |
| Programació MPLAB | 20 | | | | | | | |
| Soldar | 2 | | | | | | | |
| Debugar | 6 | | | | | | | |
| Realització de la memòria | 4 | | | | | | | |

En total varem estimar unes 30 hores per a realitzar la pràctica completament i al final, a causa dels imprevistos i el temps dedicat a entendre com funcionava el WIFI hem estat més temps del previst, en total unes 40 hores.