Digital Systems and Microprocessors practical assignment

Course 2020-2021

Practice 2

Phase B

LSSnake

	Login	Name
Students	ivet.mompin	IVET MOMPÍN SÀNCHEZ

	Board	Report	Grade
Delivery			

Date	9/7/2021

Report Cover

TABLE OF CONTENTS

SUMMARY OF THE STATEMENT

The aim of phase B of the practice for the second semester of Digital Systems, LSSnake, is to learn how to program in xc8 using the extension available in MPLABxIDE.

What we will do in this phase is setting and configuring the user interaction with the Snake game, provided to us by the University. So as to do so, we will have to configure and to enable a series of functionalities inside our board so as the user can enter its name, store its avatar in the board, play and send/transfer data to the computer, see the time since the board has started functioning, see the playing time and also modify its personal characteristics and, also, see the top 5 podium of the games played since the board was first opened. On following I am going to be more concrete:

When the board turns on, the user has 6 main options in an interactive and mobile menu displayed in the LCD:

Option 1: New Game

Option 2: Show top 5 scores

Option 3: Show users

Option 4: Modify Users

Option 5: Show Time

Option 6: Modify time

He/she will be able to move around it with the keys 2 and 8 and, after that, if he reaches the desired option, by clicking the asterisk the user will enter inside the option. It is important to take into account that some titles may be too long for the LCD. In that case, the user will have to show the title in marquee mode at a rate of 1 second.

At any time, the user will have the possibility to come back to the main menu by pressing the hashtag either if he is inside the option or outside. So now, lets talk about each option.

In the first case, option 1, the user will have two sub options, he will be able to select an existing user to play and, in that case, if pressed the asterisk again, he will go directly to the game.

On the other hand, he will be able to enter a new user. Thanks to the keypad, which will behave as SMS, the user will be able to choose among all letters of the abecedary and the numbers from 0 to 9. So as to save the keys entered and, therefore, advance one cell in the LCD,, the user will have to wait 500ms if he wants to enter the same key again or press directly another key. When finishing the name, the user will press the asterisk and will be automatically brought to playing mode.

If user goes to option 2, he will be able to see the 5 users with the best score of all games, ordered by score and displayed in marquee mode. In the same way, in option 3, the user will see all usernames already existing in the game, displaying two per screen in the same mode as option 2.

In the fourth case, the user will be able to modify any existing user in the game. He will have the option of deleting that user and the option of changing the name of the user(behaving in the same way as New user mode).

At last, in options 5 and 6 he will be able to see the time passed since we opened the door and adjust the hour(global time) to the hour he wants. For instance, by default the board will start counting in time 00:00 but if the user goes to option 6 and enters 21:00, afterwards, the time will start counting from that time.

During the execution of the game, there has to be background music playing. At least 10 notes have to be played.

While playing, the user has to see both in the LCD and in the computer the username, the time passed and the score. The score will be received from the computer, since it is developed in the game already programmed for us. The time and the username will be sent from the LCD to the computer and both processes will be carried by means of a serial channel.

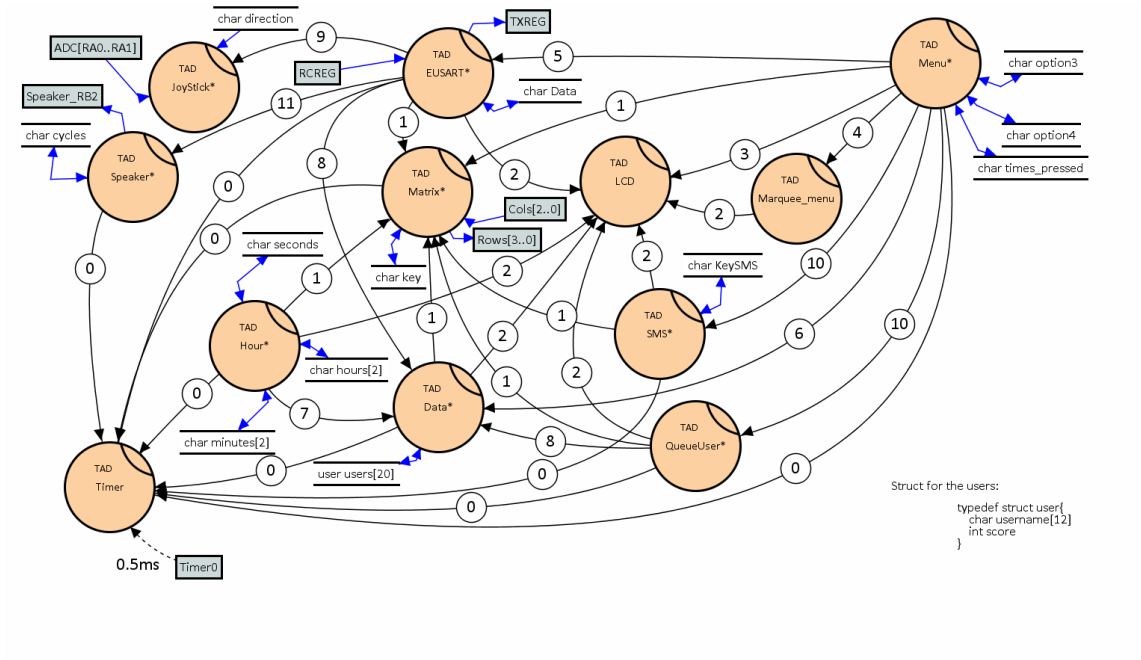
Also, we have to synchronize the keypad and the joystick with the game, since the user will be able to move the snake with the keys of the keypad, the joystick and the keys of the computer.

Finally, when the game finishes, the user will see in the LCD the current score obtained in the game and the highest score achieved for the user among all games(which will be the one to store). If the current score is bigger than the registered one, the registered score will be updated.

Whenever the user wants to exit, by pressing the hashtag he will return to the main menu.

ADT's DIAGRAM OF THE SYSTEM

ADTs diagram



Explanation

The system created for solving the problem proposed has been done in 12 modules, 9 of them being motors.

The three principal ones are the menu, where all option handling is performed, the Data ADT, where we store the most reliable information so as the users, scores, etc. And, finally, the third will be the EUSART, ADT that enables us to communicate with the computer and, therefore, playing the game.

The selection of the motors has been done in the following way.

As we know, there are several processes in the system that require of the help of the Timer of the system, so, therefore, some actions that they have to perform have to wait some time. In those cases, we will set that ADT as a motor, where we will have waiting states, transition states or where we will call functions of the interface assigned. The timer we will set, in my case, will be a timer of 0.5ms and we will work with an oscillator in HS mode.

On the other hand, all those ADTs that do not require time or that take advantage of the timers of other ADTs (as the *marquee_users* does) we will consider them as a set of functions that will be used in other ATDs following the ADTs diagram structure. For instance, in interface 4, the menu will take advantage of the marquee performed in *marquee_menu* and will pass its timer when calling the ADT so as a better handling and distribution of memory is offered, the basic purpose of dividing the system into those different modules or ADTs.

MOTORS OF THE SYSTEM

In this section of the report, the different motors of the system created will be explained one by one, so as to understand better the relation between them and the overall functioning of my software.

Matrix's motor

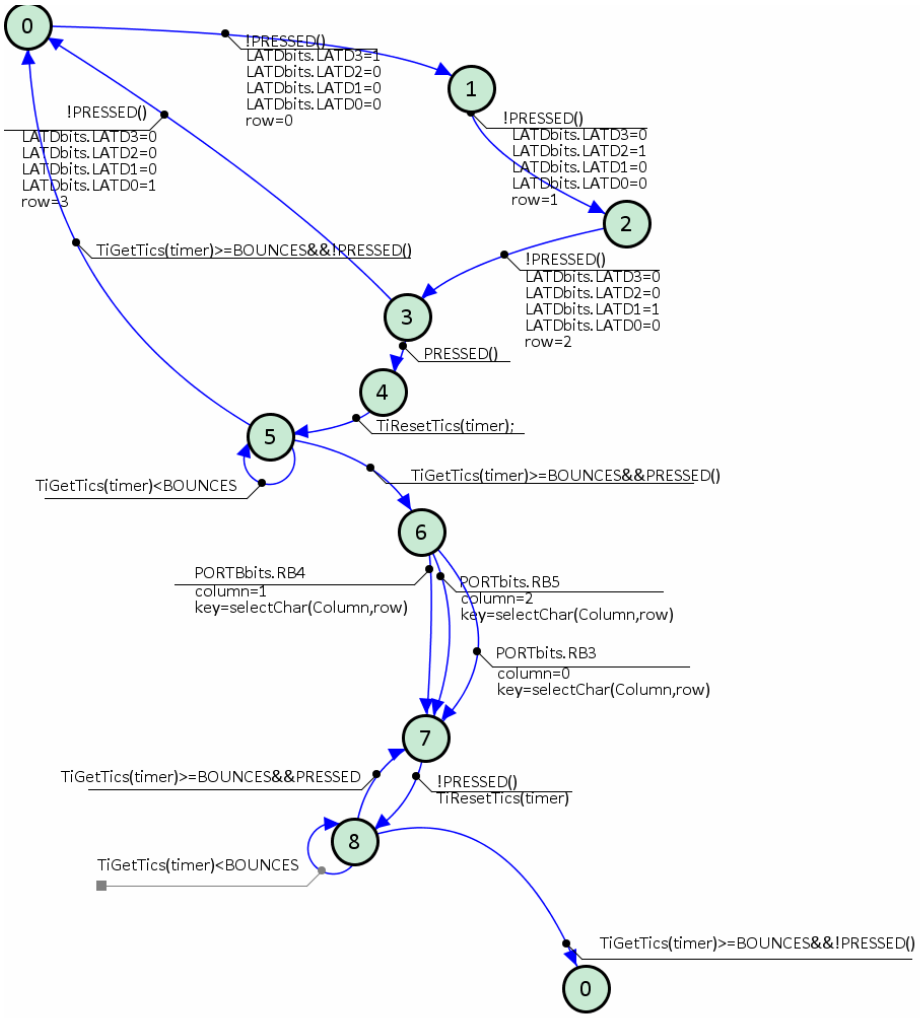
One of the key elements of our board is the matrix keyboard, the peripheral that will enable us to select, deselect and enter data to our system. Nevertheless, as we learnt in the other practices this year, it does not work on its own and, therefore, requires the help of our system.

First of all, the basic requirement to know which key the user has entered is matching the row and the column pressed. So as to enable that, we will have to sort the rows of our keypad. This is, we will have to put a one sequentially in all of them so as, when a key is pressed, the sorting stops and the proper row with the proper column match.

Nevertheless, so as to make communication with the chip and the keypad, we will have to set the rows as outputs(because we give a 1 to them) and the columns as inputs(the keypad returns the column pressed).

After that, waiting for the bounces to end will be needed. We will count until 20ms and then, by means of the *selectChar* function, we will set our variable key to the value of the corresponding key. We will store all possible values *key* can have in a matrix of constants *TABLE[MAXROWS][MAXCOLS]*

When the button is not pressed, we will perform the bounce waiting time and return to state 0. As we can see in our *main.c*, some motors require a precise value of *times_pressed,* the variable that sets each mode on in the correct time. In the case of the matrix motor, it will not happen, since the motor will be always working, will be needed for everything. On following we can see a picture of the motor created:


0
!PRESSED()
LATDbits.LATD3=1
LATDbits.LATD2=0
LATDbits.LATD1=0
LATDbits.LATD0=0
row=0
1
!PRESSED()
LATDbits.LATD3=0
LATDbits.LATD2=1
LATDbits.LATD1=0
LATDbits.LATD0=0
row=1
2
!PRESSED()
LATDbits.LATD3=0
LATDbits.LATD2=0
LATDbits.LATD1=1
LATDbits.LATD0=0
row=2
3
PRESSED()
4
TiResetTics(timer);
5
TiGetTics(timer)<BOUNCES
TiGetTics(timer)>=BOUNCES&&PRESSED()
6
PORTBbits.RB4
column=1
key=selectChar(Column,row)
PORTbits.RB5
column=2
key=selectChar(Column,row)
PORTbits.RB3
column=0
key=selectChar(Column,row)
7
TiGetTics(timer)>=BOUNCES&&PRESSED
!PRESSED()
TiResetTics(timer)
8
TiGetTics(timer)<BOUNCES
TiGetTics(timer)>=BOUNCES&&!PRESSED()
0
!PRESSED()
LATDbits.LATD3=0
LATDbits.LATD2=0
LATDbits.LATD1=0
LATDbits.LATD0=1
row=3
TiGetTics(timer)>=BOUNCES&&!PRESSED()

Joystick's motor

In LSSnake, we will use the joystick in the playing mode, since it will be one of the options the user has to move the snake. Therefore, since the user can move it up, down, right or left, in this case, we are controlling axis X and axis Y, differently from what we did in phase A, where we were controlling only one axis of the joystick.

So as to do that, I will base my guessing of the direction the user chooses to go on comparations between the values the X and Y axis adopt regarding the Threshold stated for the ADC, which will be 100.

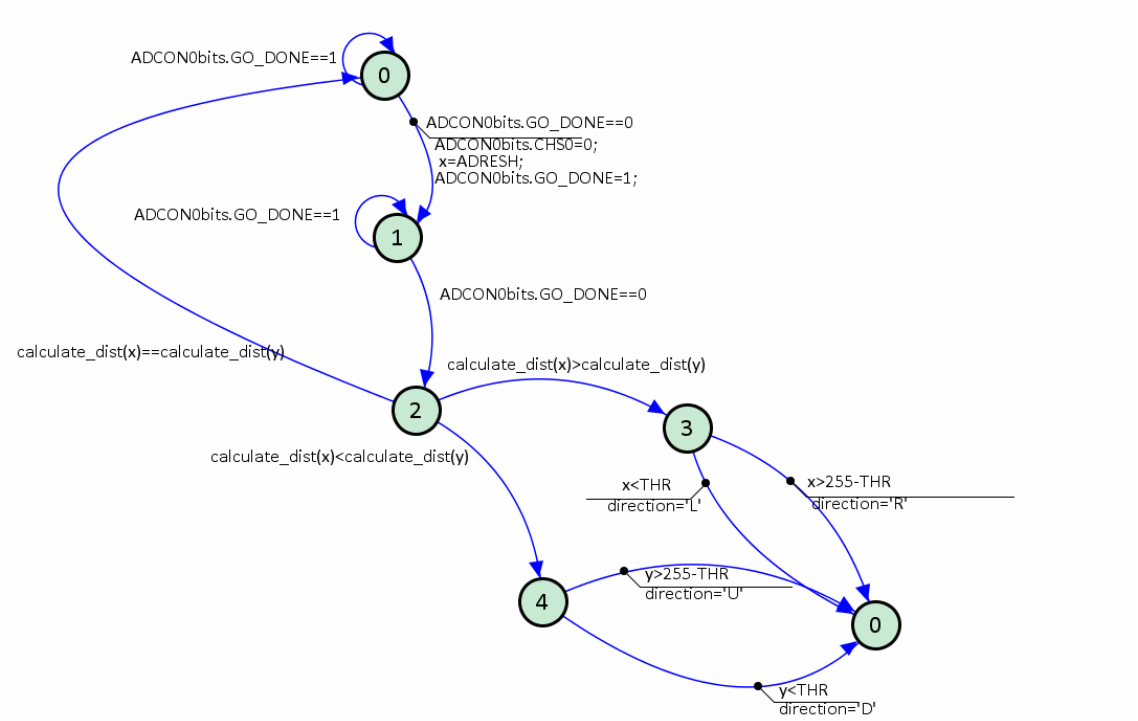
In my case, the function created *calculate_dist()* will be used for calculating the value x has, since we can have up to 256 values for each axis, we will subtract the value of the axis obtained from the ADC from 128, so as to know the position of the joystick in that axis.

The same process will be done for the other axis and, after that, we will compare both of them.

After that, we will take the biggest axis and compare it with the threshold. For instance, if x is bigger than y and also bigger than 255- threshold, we will know the user wants to go to the righr. If x is smaller than that, the user's choice will be left. And the same will happen in the case y for up and down.

Since we only want to use the joystick when we are playing(*times_pressed==15*), we will activate that motor only in that case.

On following we can see how the motor has been set:



Speaker's motor

As was stated in the requirements of the practice, when we play the game we have to play some music. Therefore, we will need to set a motor that enables us to play that music, which will be the speaker's one.

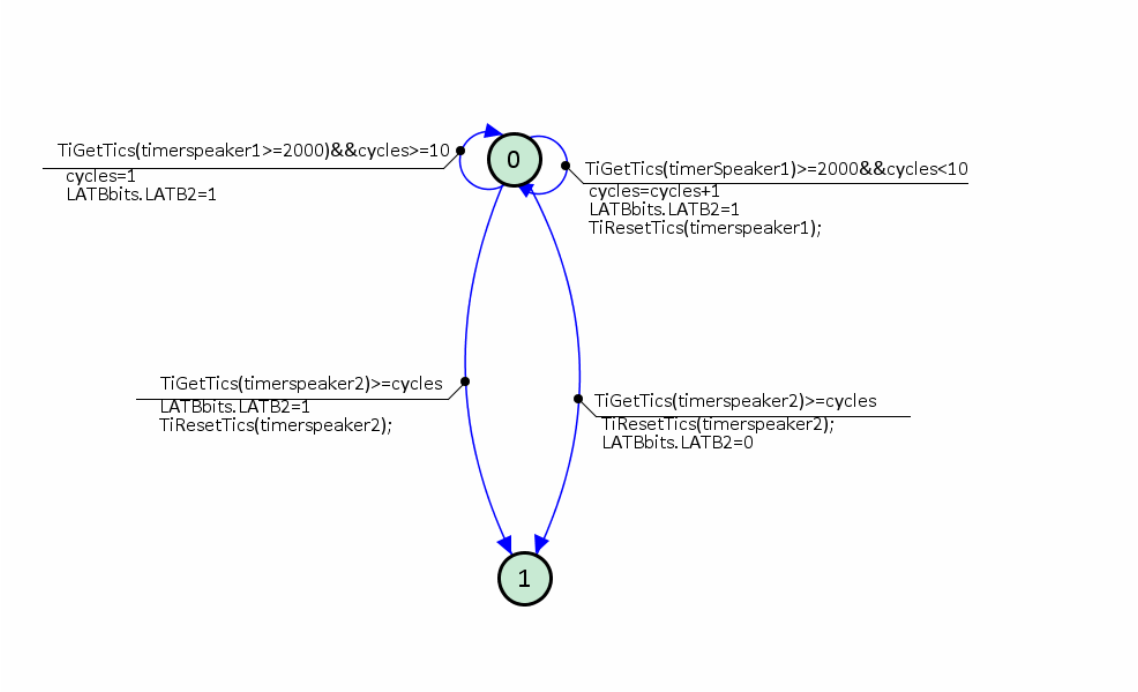
With microcontrollers, playing music is relatively easy because our peripheral(the amplifier) needs a PWM of variable period and 50% duty cycle that, depending on the amount of time it receives a 1, it will perform a acutest or a gravest tone.

Knowing the facts previously mentioned, so as to create the speaker motor we will need two timers and a variable that each time we perform a note, increment our period for the next note. This variable will be the *cycles* one and the port where we will give the signal to the amplifier will be RB2.

In the first place, we will wait for the first timer to have the value of 2000(1second) and, after that, we will use the other timer to count the amount of cycles we have to perform. When we reach the cycles wanted, we will set the signal to 0 until the first timer that was set reaches one second. If that happens, the motor will restart again, but with the desired period.

If that period reaches the value of 10, then we will start from 1 again.

On following the speaker is shown:



Menu's motor

As it has been mentioned, so the user knows his election every time, it will be needed to do a menu that displays all options he has available.

Therefore, in this motor I will manage all the configurations to make this menu work, as well as the different submenus my game will have.

Since our LCD is formed by 2 rows, I will have two variables that will indicate what has to be displayed in each of the rows: *option3* for the first and *option4* for the 2nd row.

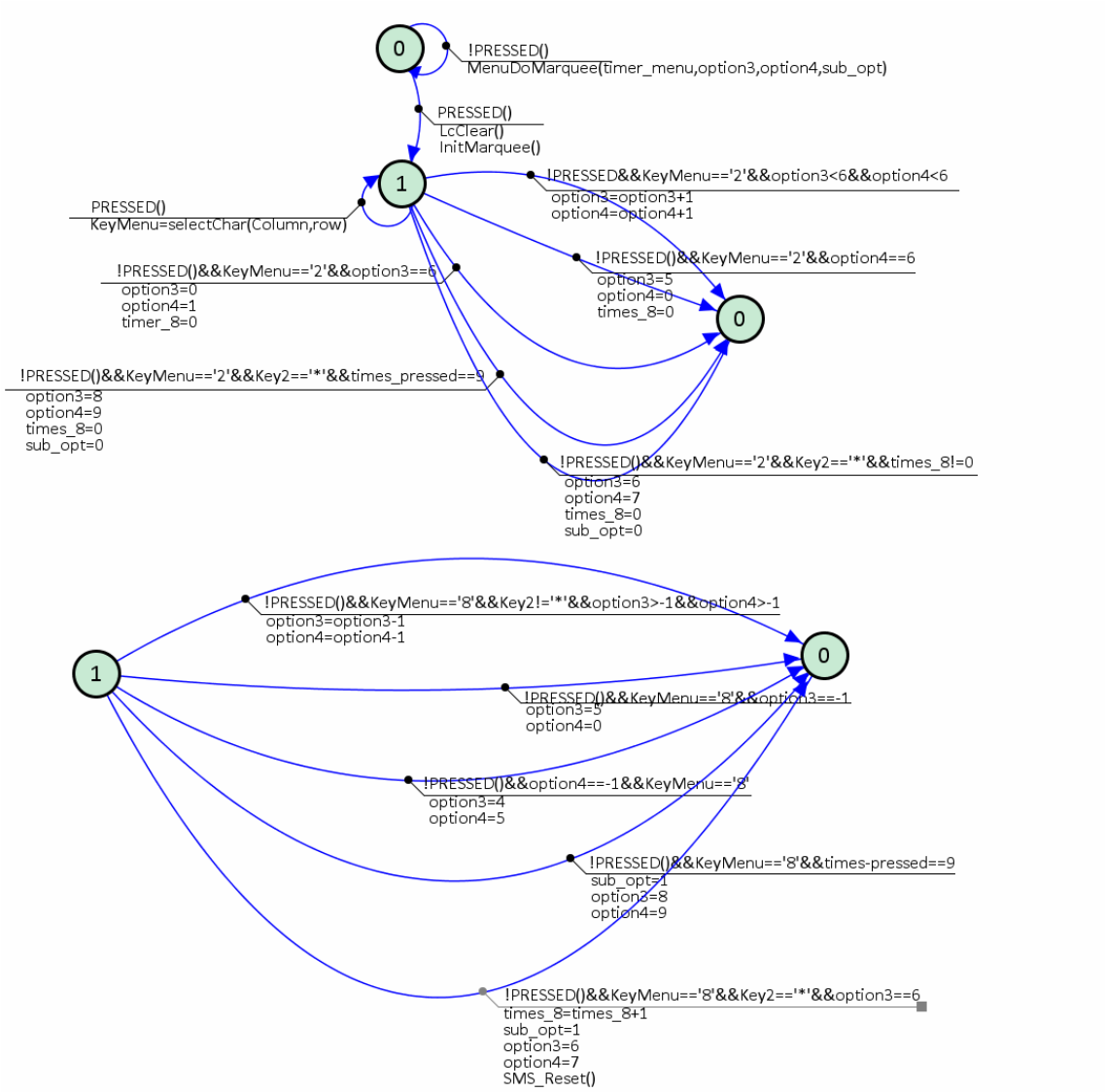
It is important to mention that he menu will be cyclical. This is, whenever the user reached more than option 6 by pressing key 2, the menu will start again with option 1. In the same way, if the user reaches an option smaller than option 1 by pressing key 8, option 6 will be shown again. If the board is turned on and the user does not do any interaction with it, by default we will show options 1 and 2 and, therefore, we will call *marquee_menu()* ADT to perform it for option 2.

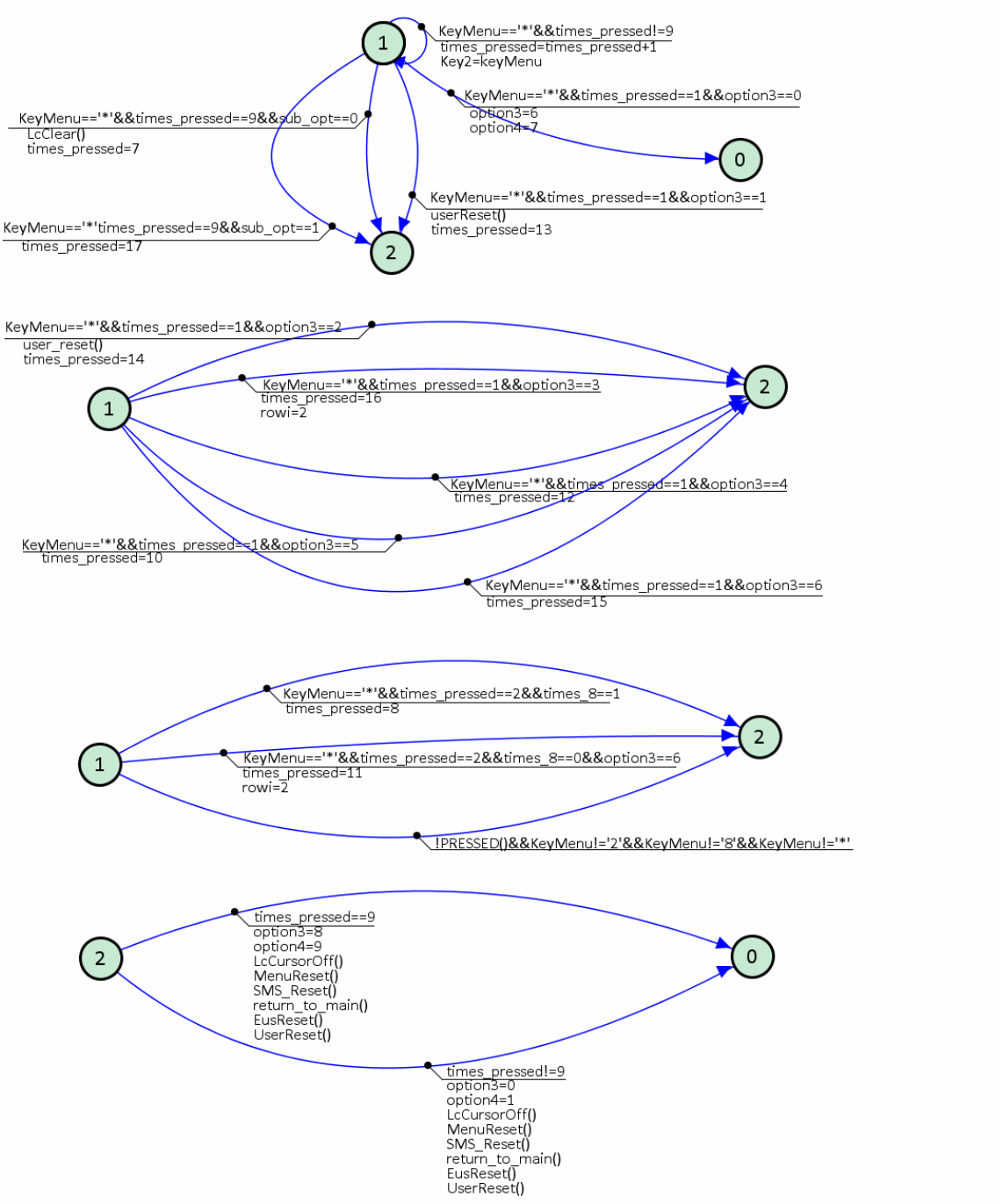
For the marquee we will have an initial pointer *i[]*, a pointer to point in the sentence *k[],* a pointer to move along the columns of the LCD *f[]* and, finally, a pointer to move in the rows *r[].*

Whenever we detect a pressing in the asterisk, the user will enter in the option he selected because our motor will have put a value in *times_pressed* that will activate the option he desired in the *main.c.* Since we want this option to perform as it previously performed, if so, we will have to reset the variables involved in the process and the related ones by calling the reset functions of our system *EusReset(), SMSReset(), UserReset(),menuReset()* and *return_to_main().* Also, will be necessary(in the case *times_pressed* is not 9(because we will be in the modify mode), to set *option 3 and 4* to 0 and 1 again, so as the user when presses the hashtag and comes back to the main menu returns to the default configuration. Here comes the relation of the ADT menu with the other ones.

In addition, we will have to control the times the user pressed * because it can happen different things. We can have that we have to perform an option directly or maybe we have to show a submenu(as in New Game mode). Therefore, depending on if the user presses * on time or 2, we will do different things.

On following the menu motor is shown:


0
!PRESSED()
MenuDoMarquee(timer_menu,option3,option4,sub_opt)
PRESSED()
LcClear()
InitMarquee()
1
!PRESSED&&KeyMenu=='2'&&option3<6&&option4<6
option3=option3+1
option4=option4+1
PRESSED()
KeyMenu=selectChar(Column,row)
!PRESSED()&&KeyMenu=='2'&&option4==6
option3=5
option4=0
times_8=0
!PRESSED()&&KeyMenu=='2'&&option3==6
option3=0
option4=1
timer_8=0
0
!PRESSED()&&KeyMenu=='2'&&Key2=='*'&&times_pressed==9
option3=8
option4=9
times_8=0
sub_opt=0
!PRESSED()&&KeyMenu=='2'&&Key2=='*'&&times_8!=0
option3=6
option4=7
times_8=0
sub_opt=0
!PRESSED()&&KeyMenu=='8'&&Key2!='*'&&option3>-1&&option4>-1
option3=option3-1
option4=option4-1
1
0
!PRESSED()&&KeyMenu=='8'&&option3==-1
option3=5
option4=0
!PRESSED()&&option4==-1&&KeyMenu=='8'
option3=4
option4=5
!PRESSED()&&KeyMenu=='8'&&times-pressed==9
sub_opt=1
option3=8
option4=9
!PRESSED()&&KeyMenu=='8'&&Key2=='*'&&option3==6
times_8=times_8+1
sub_opt=1
option3=6
option4=7
SMS_Reset()


1
KeyMenu=='*'&&times_pressed!=9
times_pressed=times_pressed+1
Key2=keyMenu
KeyMenu=='*'&&times_pressed==1&&option3==0
option3=6
option4=7
0
KeyMenu=='*'&&times_pressed==9&&sub_opt==0
LcClear()
times_pressed=7
KeyMenu=='*'&&times_pressed==1&&option3==1
userReset()
times_pressed=13
KeyMenu=='*'times_pressed==9&&sub_opt==1
times_pressed=17
2
KeyMenu=='*'&&times_pressed==1&&option3==2
user_reset()
times_pressed=14
1
KeyMenu=='*'&&times_pressed==1&&option3==3
times_pressed=16
rowi=2
2
KeyMenu=='*'&&times_pressed==1&&option3==4
times_pressed=12
KeyMenu=='*'&&times_pressed==1&&option3==5
times_pressed=10
KeyMenu=='*'&&times_pressed==1&&option3==6
times_pressed=15
KeyMenu=='*'&&times_pressed==2&&times_8==1
times_pressed=8
1
2
KeyMenu=='*'&&times_pressed==2&&times_8==0&&option3==6
times_pressed=11
rowi=2
!PRESSED()&&KeyMenu!='2'&&KeyMenu!='8'&&KeyMenu!='*'
2
times_pressed==9
option3=8
option4=9
LcCursorOff()
MenuReset()
SMS_Reset()
return_to_main()
EusReset()
UserReset()
0
times_pressed!=9
option3=0
option4=1
LcCursorOff()
MenuReset()
SMS_Reset()
return_to_main()
EusReset()
UserReset()

Data's motor

As we know, this game is not only for one user but it can store a maximum of 20 users. In this case, we need some sort of structure that enables us to store the data for each one of them and this is what we are going to do in this motor. Therefore, the main function of this motor is storing all the reliable data of the game: name and score of the user and also show it in the sub options it is required(select user, modify user, delete user).

So as to do so, we will use the help of two variables: num_users and total_users. On the one hand, the num_users will be the pointer of our array of structs, this is, each time we want to show the stored data, we will keep incrementing it until we reach the value of *total_users*: the value that indicates us the total amount of users our system has stored.

Therefore, since we have to show it in the LCD, we will require of the help of the LCD ADT so as to be able to go to the row/ column we need to do the printing, to put a char and also to clear the screen when we exit or when we move around the menu using the functions *LcGotoXY()*, *LcPutChar()* and *LcClear().*

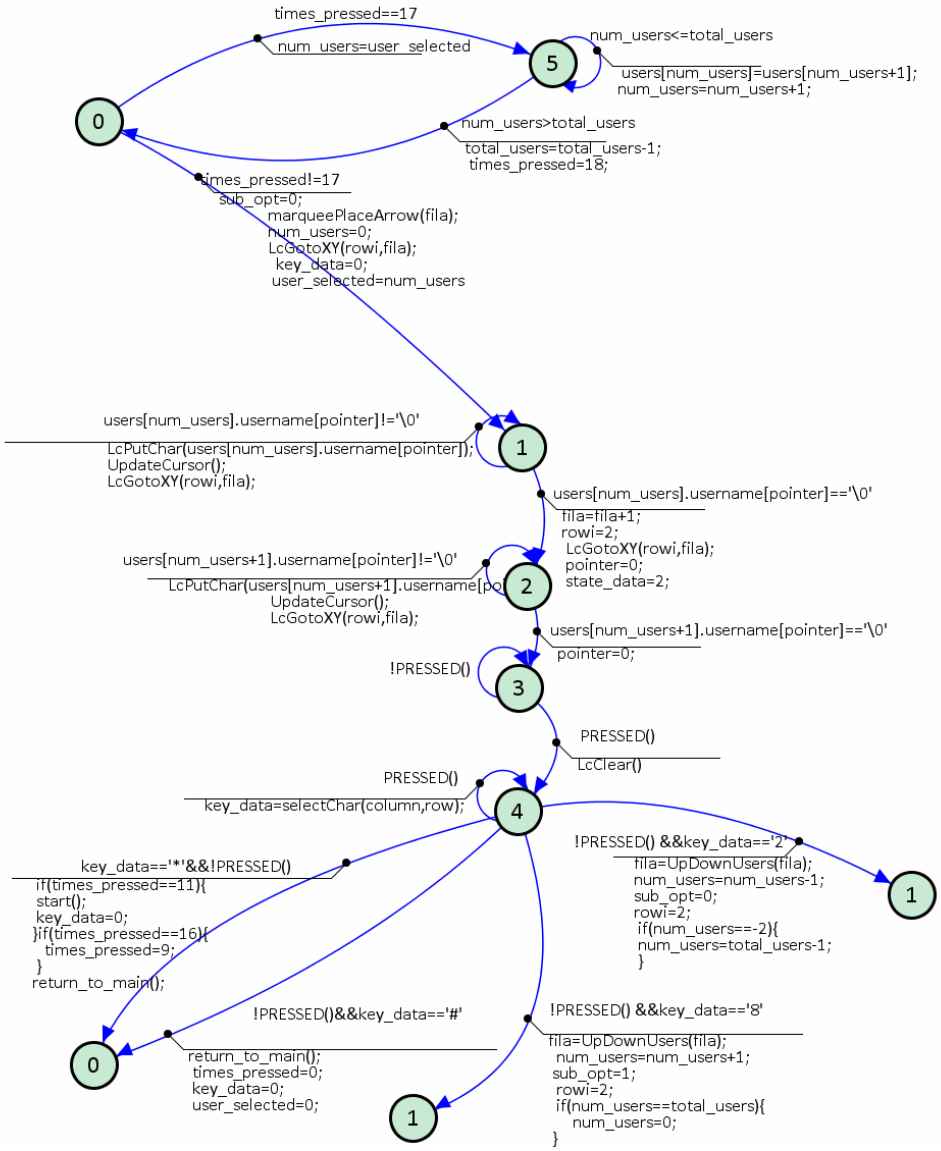
If we want to write the username, we will use a variable called *pointer* that will point into the character of the string stored that belongs to the user that has to be displayed. If the string reaches '\0', we will finishing printing the string.

If we want to move around the menu of users, as explained, we will use the same logic as in the menu. We will use 2 to move up and 8 to move down. In the first case, therefore, we will decrement *num_users* to move to the previous user until we reach user 0.

On the other hand, if we press 8, we will increment in 1 *num_users*, so as to point to the next user until our variable reaches *total_users*.

If the user presses *, the user will be brought immediately to the game because we will send a 'Y' through the serial channel(the initializer of our java program) and we will set *times_pressed* to 15, that will make us go directly to EUSART motor.

From that moment, we will have exited the data motor.


times_pressed==17
num_users=user_selected
num_users<=total_users
users[num_users]=users[num_users+1];
num_users=num_users+1;
num_users>total_users
total_users=total_users-1;
times_pressed=18;
times_pressed!=17
sub_opt=0;
marqueePlaceArrow(fila);
num_users=0;
LcGotoXY(rowi,fila);
key_data=0;
user_selected=num_users
users[num_users].username[pointer]!='\0'
LcPutChar(users[num_users].username[pointer]);
UpdateCursor();
LcGotoXY(rowi,fila);
users[num_users].username[pointer]=='\0'
fila=fila+1;
rowi=2;
LcGotoXY(rowi,fila);
pointer=0;
state_data=2;
users[num_users+1].username[pointer]!='\0'
UpdateCursor();
LcGotoXY(rowi,fila);
users[num_users+1].username[pointer]=='\0'
pointer=0;
!PRESSED()
PRESSED()
LcClear()
PRESSED()
key_data=selectChar(column,row);
!PRESSED() &&key_data=='2'
fila=UpDownUsers(fila);
num_users=num_users-1;
sub_opt=0;
rowi=2;
if(num_users==-2){
num_users=total_users-1;
}
key_data=='*'&&!PRESSED()
if(times_pressed==11){
start();
key_data=0;
}if(times_pressed==16){
times_pressed=9;
}
return_to_main();
!PRESSED()&&key_data=='#'
return_to_main();
times_pressed=0;
key_data=0;
user_selected=0;
!PRESSED() &&key_data=='8'
fila=UpDownUsers(fila);
num_users=num_users+1;
sub_opt=1;
rowi=2;
if(num_users==total_users){
num_users=0;
}
0
1
2
3
4
5

EUSART's motor

When we receive the 'Y' character mentioned in the previous section, the transmission and reception of data wth the computer will start. Therefore, we will have entered and activated the EUSART motor, the motor that will help us playing the game.

During the game, the motor will perform multiple functions. On the one hand, as soon as we start, the motor will send the username, char by char, to the computer and will count the time passed since the activation of the motor. This is, by means of the timer, we will count every second that passes and send a '+' to the computer. In that way, we will be able to see the time passed in both the computer and the board.

In the same time we are playing and sending info through the serial channel we will be constantly listening to the RCREG so as to check if there is new score data to register. If that happens, we will first make a cast of data to integer so as to be able to compare it with the stored score of the user and, if the score obtained is greater than the stored one, we will store in the array of structures of the user selected on *Select User* or in *New User.*

While playing, in the screen we will see the username, the time played and also the current score, which will be updated everytime the user earns points. By means of the function ScoreDigits() and DivideScore() we will be able to transform the integer value of the score into digits in the LCD.

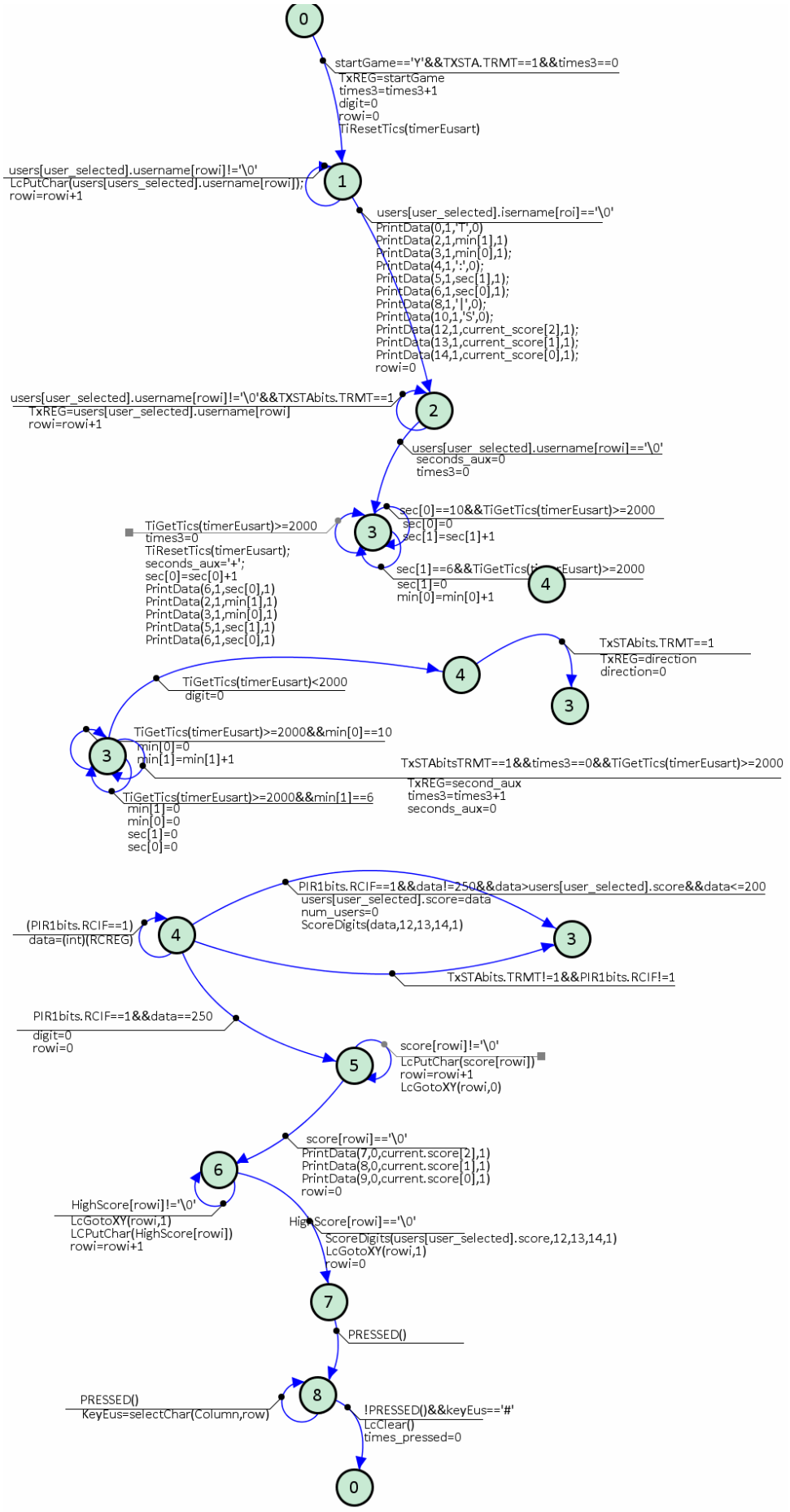
Next, if the snake hits the wall, the game is finished. At that time, the computer will send a 250 so as to indicate the end of the game and we will exit from the current screen mode. In that time, we will access to the new screen, where we will show the current score and the high score of the user.

So as to display the message, we will use two constant arrays: *score[]* for the first row and *HighScore[]* for the 2nd. We will print both of them in the same way as in *data.c,* until we reach the '\0'. After that, we will go to the first row to print the current score by entering to the function *ScoreDigits()* the variable *data* as parameter and we will perform the division of the digits. Basically we will perform the modulus of the integer value to get the units and after that, update the score value by dividing by 10. We will perform the modulus and save the tens and divide again. The remaining value will be the hundreds. We will store the three digits in *current_score[]* array.

So as to print the score, we will use the function *PrintData().* Since we are printing values that are not constants, we will introduce to the function the value 1 for *yes*, which means adding '0' to transform it to char and print it.

The same process will be followed for printing the high score but, in this case, instead of passing as parameter in *ScoreDigits()* the variable *data*, we will pass *users[num_users].score.*

Now the only missing thing here is waiting for the user to press the hashtag. If so, the program will go back to the main menu.


0
startGame=='Y'&&TXSTA.TRMT==1&&times3==0
TxREG=startGame
times3=times3+1
digit=0
rowi=0
TiResetTics(timerEusart)
users[user_selected].username[rowi]!='\0'
LcPutChar(users[users_selected].username[rowi]);
rowi=rowi+1
1
users[user_selected].isername[roi]=='\0'
PrintData(0,1,'T',0)
PrintData(2,1,min[1],1)
PrintData(3,1,min[0],1);
PrintData(4,1,':',0);
PrintData(5,1,sec[1],1);
PrintData(6,1,sec[0],1);
PrintData(8,1,'|',0);
PrintData(10,1,'S',0);
PrintData(12,1,current_score[2],1);
PrintData(13,1,current_score[1],1);
PrintData(14,1,current_score[0],1);
rowi=0
users[user_selected].username[rowi]!='\0'&&TXSTAbits.TRMT==1
TxREG=users[user_selected].username[rowi]
rowi=rowi+1
2
users[user_selected].username[rowi]=='\0'
seconds_aux=0
times3=0
sec[0]==10&&TiGetTics(timerEusart)>=2000
sec[0]=0
sec[1]=sec[1]+1
TiGetTics(timerEusart)>=2000
times3=0
TiResetTics(timerEusart);
seconds_aux='+';
sec[0]=sec[0]+1
PrintData(6,1,sec[0],1)
PrintData(2,1,min[1],1)
PrintData(3,1,min[0],1)
PrintData(5,1,sec[1],1)
PrintData(6,1,sec[0],1)
3
sec[1]==6&&TiGetTics(timerEusart)>=2000
sec[1]=0
min[0]=min[0]+1
4
TxSTAbits.TRMT==1
TxREG=direction
direction=0
3
TiGetTics(timerEusart)<2000
digit=0
4
TiGetTics(timerEusart)>=2000&&min[0]==10
min[0]=0
min[1]=min[1]+1
3
TxSTAbitsTRMT==1&&times3==0&&TiGetTics(timerEusart)>=2000
TxREG=second_aux
times3=times3+1
seconds_aux=0
TiGetTics(timerEusart)>=2000&&min[1]==6
min[1]=0
min[0]=0
sec[1]=0
sec[0]=0
PIR1bits.RCIF==1&&data!=250&&data>users[user_selected].score&&data<=200
users[user_selected].score=data
num_users=0
ScoreDigits(data,12,13,14,1)
(PIR1bits.RCIF==1)
data=(int)(RCREG)
4
3
TxSTAbits.TRMT!=1&&PIR1bits.RCIF!=1
PIR1bits.RCIF==1&&data==250
digit=0
rowi=0
score[rowi]!='\0'
LcPutChar(score[rowi])
rowi=rowi+1
LcGotoXY(rowi,0)
5
score[rowi]=='\0'
PrintData(7,0,current.score[2],1)
PrintData(8,0,current.score[1],1)
PrintData(9,0,current.score[0],1)
rowi=0
6
HighScore[rowi]!='\0'
LcGotoXY(rowi,1)
LCPutChar(HighScore[rowi])
rowi=rowi+1
HighScore[rowi]=='\0'
ScoreDigits(users[user_selected].score,12,13,14,1)
LcGotoXY(rowi,1)
rowi=0
7
PRESSED()
PRESSED()
KeyEus=selectChar(Column,row)
8
!PRESSED()&&keyEus=='#'
LcClear()
times_pressed=0
0

Queue user and top 5 motor

As indicated in the statement, in options 2 and 3 we have to show the users of the system in marquee mode. In case of option 2, we will have only to show the 5 users with the best score and their score as well. Regarding option 3, we will directly show the users.

So as to do so, we will use the same motor but we will pass the variable *times_pressed* as a parameter of the motor function. If *times_pressed == 13* (option 3), we will directly go to the marquee and display the users themselves. If *times_pressed==14* (option2), we will first sort the users.

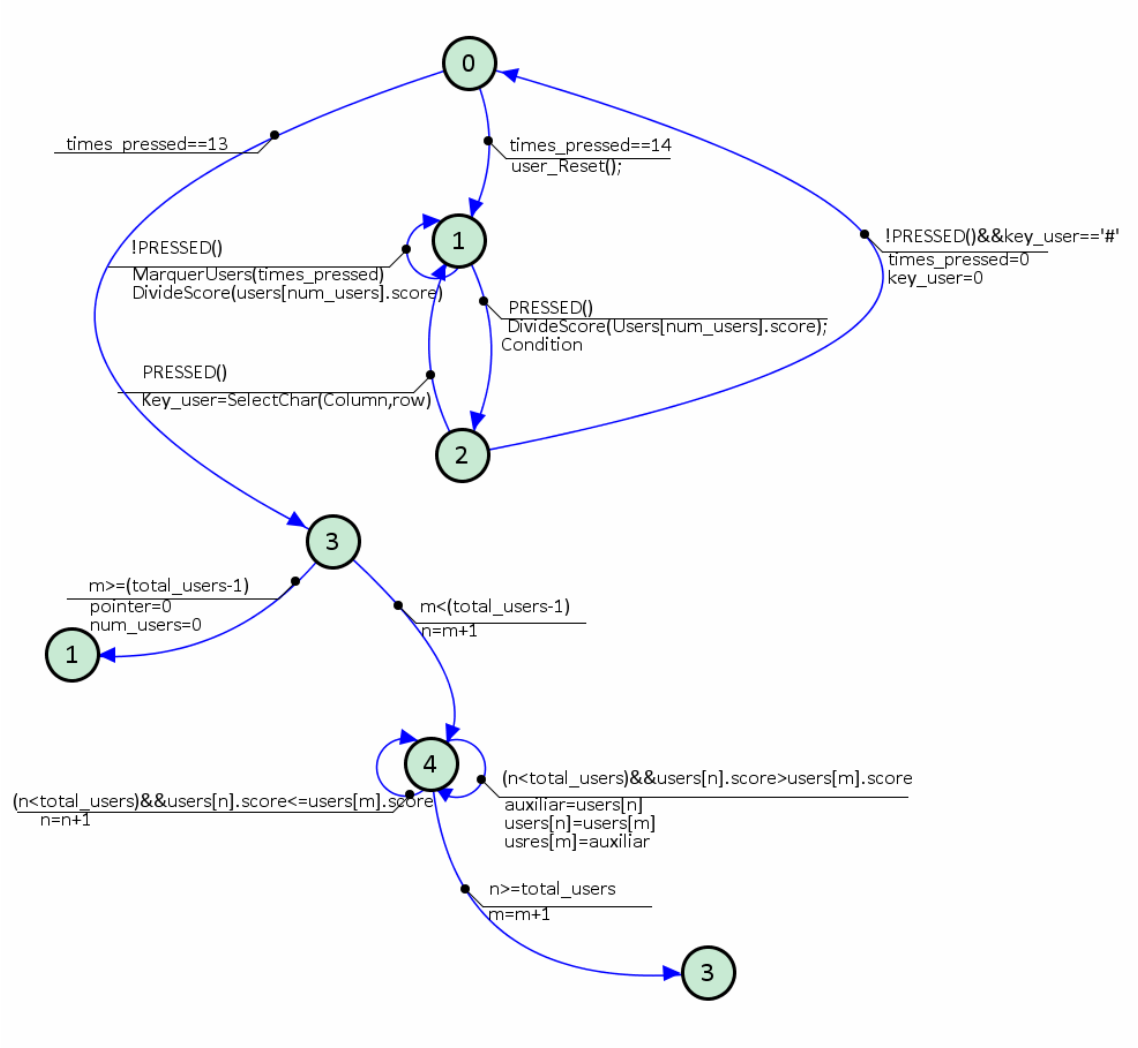
For sorting the users, we will sort the array users directly. We will use the bubble method which is checking all positions and, by means of an auxiliar struct, change the struct of place if the following value is greater than the previous. The process will be performed for all users until we reach *total_users* and the program, by means of two variables, *m* and n (*m* pointing to number 0 struct and *n* to number 1 struct.

After the sorting, the program will reinitialize the *num_users* value to 0 (since it has been changed while looping around the array of structs) and redirect the motor to state 1, where we will perform the display of the users and scores.

The display will be a little bit different in the two options in terms of computation. In the case of option 2, we will only require *num_users* and *num_users3* variables since we will only represent the users in the first row. In the case of option 3, since we have to show one user each row, we will need two sets of pointers*, num_users* and *num_users3* in one side and *num_users2* and *num_users4* on the other side. These sets of pointers will indicate the first user and the next user to be displayed, respectively.

The way we are going to perform the marquee will be similar to the one of the menu: we will use the *i[], r[], k[],* and we will print the whole row but always, incrementing in 1 the *i[],* so the text seems to be moving. If *i[]* reaches the end(>15), we increment in 1 the *num_users* and *num_users3* variables and, also, we reinitialize the *i[].*

On following the queueUser motor is shown:


0
times_pressed==13
times_pressed==14
user_Reset();
1
!PRESSED()
MarquerUsers(times_pressed)
DivideScore(users[num_users].score)
!PRESSED()&&key_user=='#'
times_pressed=0
key_user=0
PRESSED()
DivideScore(Users[num_users].score);
Condition
PRESSED()
Key_user=SelectChar(Column,row)
2
3
m>=(total_users-1)
pointer=0
num_users=0
1
m<(total_users-1)
n=m+1
4
(n<total_users)&&users[n].score<=users[m].score
n=n+1
(n<total_users)&&users[n].score>users[m].score
auxiliar=users[n]
users[n]=users[m]
usres[m]=auxiliar
n>=total_users
m=m+1
3

SMS

SMS is the motor that will enable the user to enter data inside the board. This is, the name and the hour.

In the case of the name, the keypad will behave as the SMS we all know: depending on the times the user presses the key, one or other character will be saved. For instance, if the user presses the key 2 3 times, he will obtain character b since the characters the key will offer to the user will be :'2-a-b-c'.

If the user wants to enter a new key and store the already placed one, he has to wait about 500ms to enter the next key. In that time, the program will store the key entered(which will come from an array of chars called *SMS[8][7])* in the array of structs *users*. Each time we exit SMS mode and the user presses '*' we will increment in one *total_users* and *num_user*, so as to know where the storage has to be done in the next case.

Regarding the character the user enters, each time he user presses the same key a pointer tells us the key to print(*digit*). If the pointer reaches the total amount of characters for that key, *digit* is reseted and shows the first char of the string.

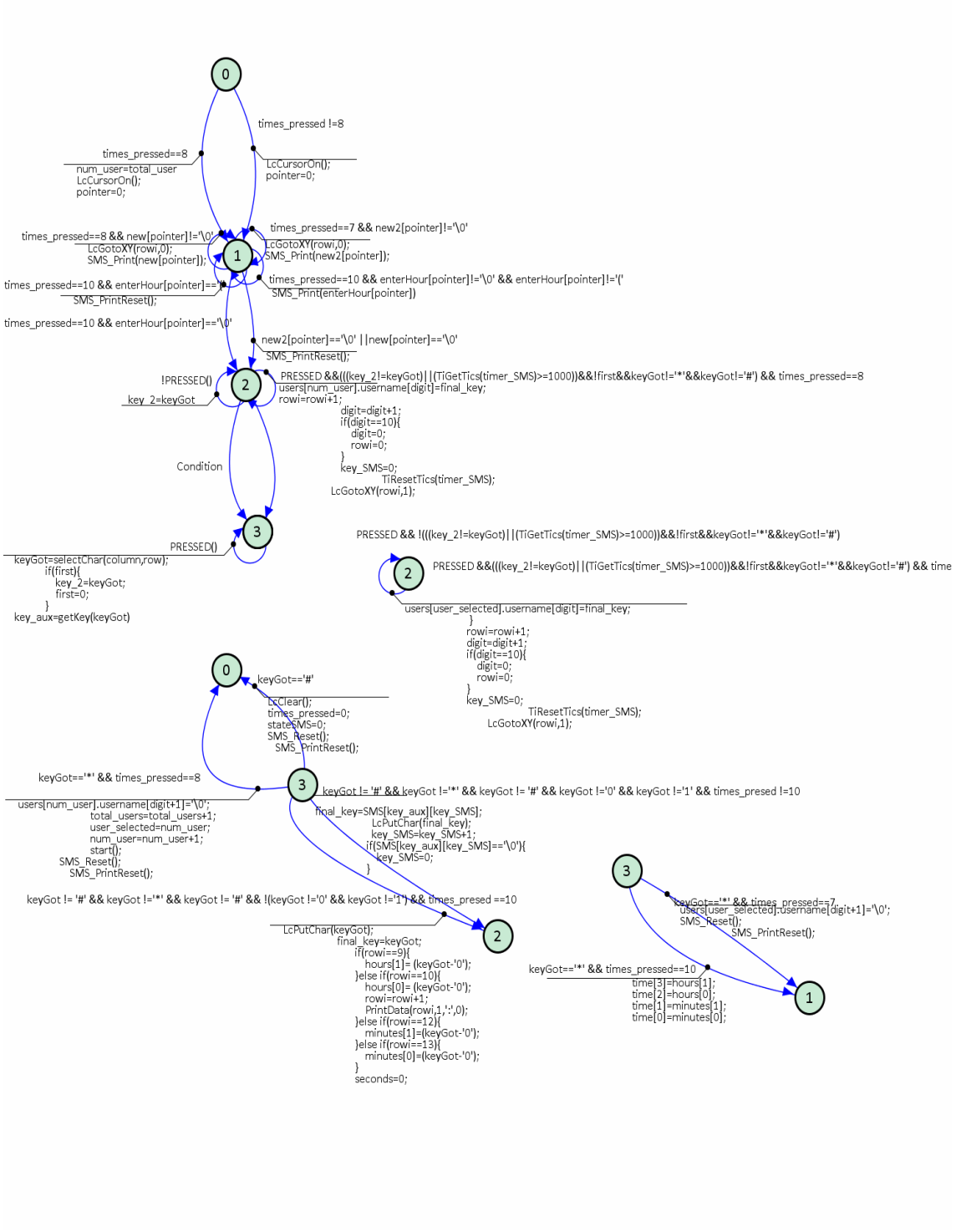
On the other hand, in the same place where we compare the time between pressings, if the user presses a different key than before(we store the previous key in *key2*, so we know the previous char), the cursor will move and the previous key will automatically be saved.

The user will keep doing the process until he presses '*'. Nevertheless, if he presses '#', *num_user* will not increment and, therefore, the name will not be saved.

On the other hand, if the user presses the asterisk, the motor will automatically activat start() function from *data.c* where we will set *startGame='Y'* and *times_pressed=15.* Therefore, we will directly to *EUSART.c* motor.

Nevertheless, *SMS.c* motor can perform different options and, therefore, we will have to differentiate the different functions of options it can perform(different values of *times_pressed* will be detected by the motor). Therefore, if *times_pressed*==8, we will be in New User mode, where we will have to print a different title from when *times_pressed*==7, the edit name mode.

This will also happen for modify time, *times_pressed*==10, where we will also enter another title too. This differentiation will be performed in case 0 of our motor. So the first thing to do before the name introduction will be printing the title of the process until '\0' is detected, in the same way as we did the other times.


0
times_pressed !=8
LcCursorOn();
pointer=0;
times_pressed==8
num_user=total_user
LcCursorOn();
pointer=0;
times_pressed==8 && new[pointer]!='\0'
LcGotoXY(rowi,0);
SMS_Print(new[pointer]);
1
times_pressed==7 && new2[pointer]!='\0'
LcGotoXY(rowi,0);
SMS_Print(new2[pointer]);
times_pressed==10 && enterHour[pointer]=='\0'
SMS_PrintReset();
times_pressed==10 && enterHour[pointer]!='\0' && enterHour[pointer]!='('
SMS_Print(enterHour[pointer])
times_pressed==10 && enterHour[pointer]=='\0'
new2[pointer]=='\0' ||new[pointer]=='\0'
SMS_PrintReset();
!PRESSED()
key_2=keyGot
2
PRESSED &&(((key_2!=keyGot)||(TiGetTics(timer_SMS)>=1000))&&!first&&keyGot!='*'&&keyGot!='#') && times_pressed==8
users[num_user].username[digit]=final_key;
rowi=rowi+1;
digit=digit+1;
if(digit==10){
digit=0;
rowi=0;
}
key_SMS=0;
TiResetTics(timer_SMS);
LcGotoXY(rowi,1);
Condition
3
PRESSED()
keyGot=selectChar(column,row);
if(first){
key_2=keyGot;
first=0;
}
key_aux=getKey(keyGot)
PRESSED && !(((key_2!=keyGot)||(TiGetTics(timer_SMS)>=1000))&&!first&&keyGot!='*'&&keyGot!='#')
2
PRESSED &&(((key_2!=keyGot)||(TiGetTics(timer_SMS)>=1000))&&!first&&keyGot!='*'&&keyGot!='#') && time
users[user_selected].username[digit]=final_key;
}
rowi=rowi+1;
digit=digit+1;
if(digit==10){
digit=0;
rowi=0;
}
key_SMS=0;
TiResetTics(timer_SMS);
LcGotoXY(rowi,1);
0
keyGot=='#'
LcClear();
times_pressed=0;
stateSMS=0;
SMS_Reset();
SMS_PrintReset();
keyGot=='*' && times_pressed==8
users[num_user].username[digit+1]='\0';
total_users=total_users+1;
user_selected=num_user;
num_user=num_user+1;
start();
SMS_Reset();
SMS_PrintReset();
3
keyGot != '#' && keyGot !='*' && keyGot != '#' && keyGot !='0' && keyGot !='1' && times_presed !=10
final_key=SMS[key_aux][key_SMS];
LcPutChar(final_key);
key_SMS=key_SMS+1;
if(SMS[key_aux][key_SMS]=='\0'){
key_SMS=0;
}
keyGot != '#' && keyGot !='*' && keyGot != '#' && !(keyGot !='0' && keyGot !='1') && times_presed ==10
LcPutChar(keyGot);
final_key=keyGot;
if(rowi==9){
hours[1]= (keyGot-'0');
}else if(rowi==10){
hours[0]= (keyGot-'0');
rowi=rowi+1;
PrintData(rowi,1,':',0);
}else if(rowi==12){
minutes[1]=(keyGot-'0');
}else if(rowi==13){
minutes[0]=(keyGot-'0');
}
seconds=0;
2
3
keyGot=='*' && times_pressed==7
users[user_selected].username[digit+1]='\0';
SMS_Reset();
SMS_PrintReset();
keyGot=='*' && times_pressed==10
time[3]=hours[1];
time[2]=hours[0];
time[1]=minutes[1];
time[0]=minutes[0];
1

Hour motor

Finally, we will talk about the last motor of our system, the Hour motor. In this motor we will perform the timing calculation while the board is active.

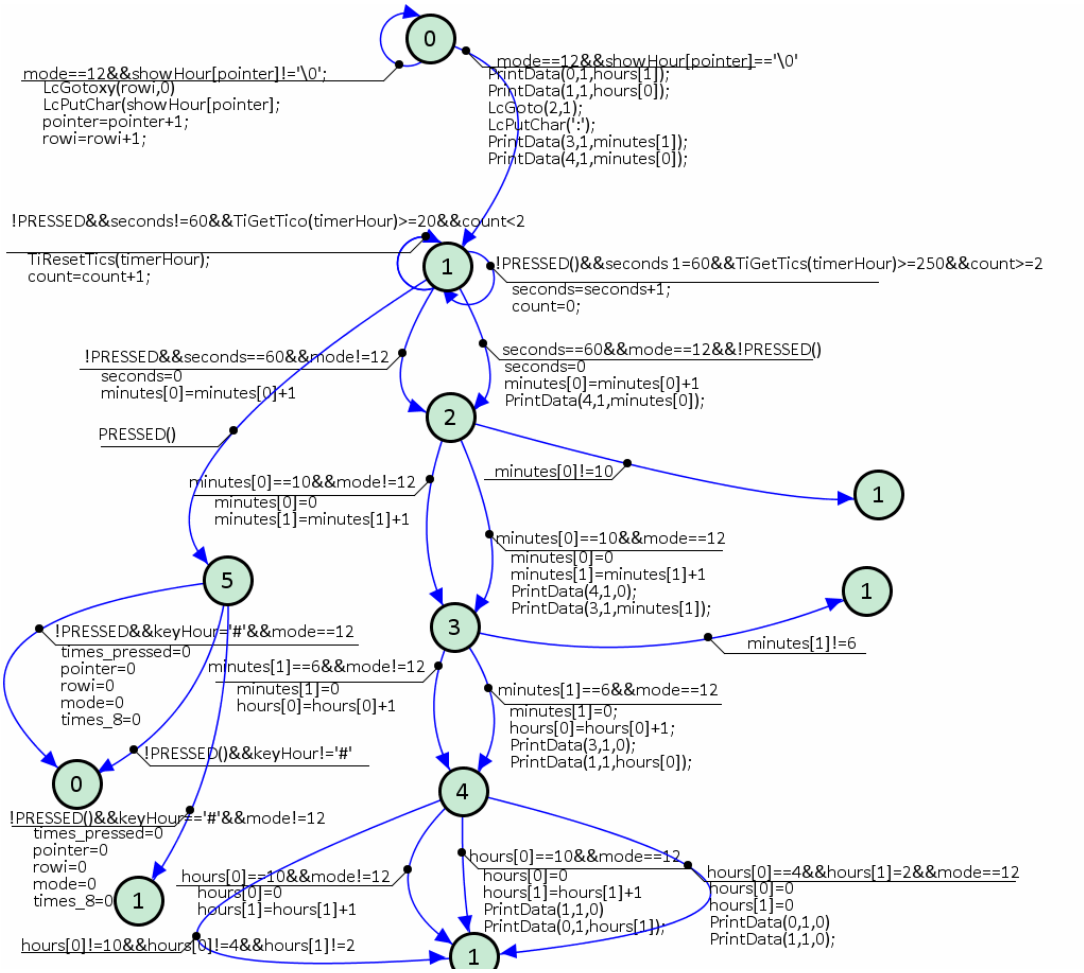
So as to do so, we will perform similar to the *EUSART.c* timing calculation, but instead of minutes and seconds, we will perform in hours, minutes and seconds.

Therefore, each time our timer reaches 1 second(2000 tics) we will increment *seconds* in 1. When *seconds* ==60, we will increment the position 0 of the the array *minutes[]* in 1 and reset seconds, and reset seconds. When *minutes[0]*=10, we will reset it too and increment *minutes[1].*

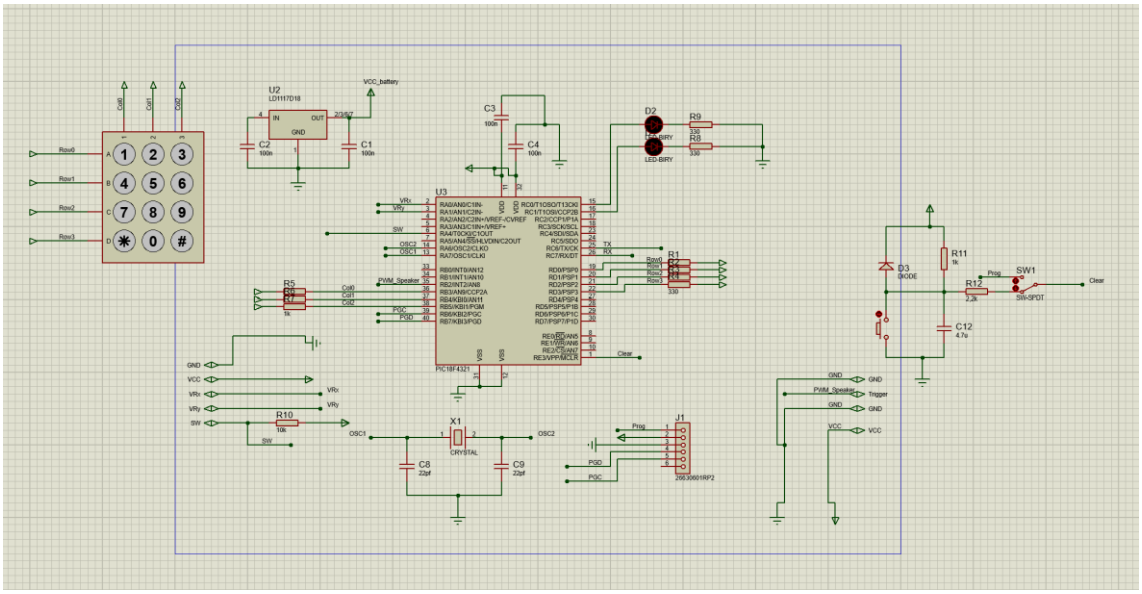
In this case, the whole process will be done again until *minutes[1]*=6, time when an hour will have passed and we will have to increment the position 0 of the hour. For the hour, the position 0 will increment from 0 to 9 but, on the case of *hours[1],* if it reaches 2 and *hours[0]* reaches 4, the whole hour will be reseted, since a new day would have began.

The whole process will be printed by means of the function *PrintCouples()* and *PrintData(),* always taking into account that, since we are not storing constants, we will have to add a '0' so as to print it in the LCD.

It is important to mention that this motor will be always operative in terms of time calculation. Nevertheless, the printing will be only performed when *times_pressed* acquires a value of 12, which will mean that we are in show time mode, we will check it in the beginning of the motor.



ELECTRICAL SCHEMATIC OF THE BOARD



As it is visible in the picture above, the electrical schematic for phase B is really similar to phase A one. The thing is that I have took advantage of some circuits that were similar for the devices in phase B (as sonoscanner's circuits for speaker's one) or changed the way the ports are connected to the devices but not the ports itself(as it happened with the ports of the seven segments(In phase A) that now are connected to the rows of the matrix keyboard(or the columns of the keyboard, which before were pushbuttons).

PROBLEMS FACED

LSSnake is a game that has a lot of features and particularities that a programmer has to take into account when developing a game like that. It is usual, therefore, that many times some problems arise while confectioning the final result. In this section of the report I am going to explain from my own experience the problems that I had when creating my board and how I dealt with them.

The first problem I experienced was when starting the menu configuration. When I was trying to make the menu stop when the user was entering a '*', the menu reappeared again and again and this fact was because I was not limiting its operation. This is, I was not setting a variable that indicated to the program that the menu should not be executed at that time. Is for this reason why I used the *times_pressed* variable, which indicated at each moment which motor/option had to be carried on.

Another problem emerged was with the SMS. When I was entering the next key with a difference of time greater than 500ms, the program did not changed the cursor of place immediately. This was a problem of where I was setting the condition of the time. The problem is that I was doing it at the end of the state, when I should do it just after the pressing. The program was placing first the next char and, after that, move the cursor but I wanted to move it directly so I had to do it before the printing of the char. From this problem emerged another one linked with the *data.c* . The problem I had is that, since the check was done after the printing, the SMS was not storing the correct key in my structure. Therefore, when I was displaying the users, they were wrong saved. With the previous solution, I was able to solve that problem too.

Next, I had problems with the marquee in the sense that I was not making the transition of the increment of the user at the proper time and, therefore, the text passed from the end to the beginning of the row in one change. From that problem and also from the speaker configuration, I discovered that I was not adjusting my timer correctly, since I was using a value that was giving me a period of 2ms, not of 0.5ms. Finally, I adjusted the value and all worked correctly.

Another problem I had when doing the board was with the EUSART. I did not realize that I was using HS instead of HSPLL so my configurations matched with HSPLL and not with HS. Therefore, the serial port was not working because of that. I changed some configurations regarding the TXSTA register and all worked correctly.

Last problem I had in this project was the matrix keyboard. Since I had not left so much space for it, I had to manage its connections with pins and sometimes misconnections occurred. I had to change the matric several times but finally, the last one worked correctly. Now the keypad is okay.

CONCLUSIONS

After finishing the board for phase B for the second semester of Digital Systems and Microprocessors, I have extracted some conclusions.

The first thing I realized after finishing is that the PWM is a tool that is so helpful when working with physical motors as we saw in phase A but also for daily life devices as it is a speaker. In some way, with the same command, we can do a lot of different things, producing music, moving a motor, etc.

The second conclusion extracted is that the division of our code into modules makes the code more understandable and also more optimized since it is easier to see the relation between the modules and use the functions more appropriately. Also, having the code divided in modules allows us to work with global variables, thing that is so helpful if we want to use the same variable in different ADTs.

In third place, I have learnt how we can make a code that we learned in programming 1, with loops and fors, be cooperative with motors and if/else. This makes my board, a device with little memory to work as our actual computer does with no effort. By staying in one state or in another and, when necessary, operating the transition needed to advance, we act like we were in a loop in a sample of c code.

At last, I have learnt that some actions that were done in the previous practice in Assembly are not that different when you work with motors and c language. The commands are different but the code is nearly the same and, also, as we have been noticed this year, what our compiler does is exactly passing our c program to assembly. Here comes the little differences.

Finally, so as to end with the conclusions, I have to say that working in this project has made me and I think all Digital Systems and Microprocessors students evolve as the developers we want to be in the future since, we have learnt that a complex game and code can fit into a little device like the PIC18F4321 if you program it efficiently and correctly. In the end, us, as engineers what we want are the solutions that work and are cheap. In this sense, we are not working with money but we are working with memory and this program , in some way, works and is not expensive in memory terms.

PLANNING

Expected planning

Since I was able to deliver phase A over ten, my planning was following a little bit the path I followed for that phase.

In the first week I was planning looking deeply at the statement and do a little of research so as to drive the problem in the right direction.

In the second week, I would solder the parts of the board that had to be soldered and maybe start the code or make an sketch of it.

Until a week before the delivery, I was planning on doing the code and testing the board so, in the last week, I could do the report and deliver over 10 again.

Nevertheless, I believe I decided wrong because I was applying the same method to a board that had nothing to do with the previous one: was more complex and during the period of confection we had the finals and the other deliveries of the degree. Therefore, the previous plan was not suitable for both the project and the situation when the project had to be done. It is not the same making a project in the beginning of the semester than at the end of it, apart from, as logical, I was doing it alone so, in this case, I had the work doubled.

WEEK	SKETCH	SKETCH	SKETCH	SKETCH	SKETCH	SKETCH	SKETCH
1st April	SOLDERING	SOLDERING	SOLDERING	SOLDERING	SOLDERING	SOLDERING	SOLDERING
2nd April	SOLDERING	SOLDERING	SOLDERING	SOLDERING	CODE	CODE	CODE
3rd April	CODE	CODE	CODE	CODE	CODE	CODE	CODE
4th April	CODE	CODE	CODE	CODE	CODE	CODE	CODE
1st May	CODE	CODE	CODE	CODE	CODE	CODE	CODE
2nd May	CODE	CODE	CODE	CODE	CODE	CODE	CODE
3rd May	CODE	CODE	CODE	CODE	CODE	CODE	CODE
4th May	CODE	CODE	CODE	REPORT	REPORT	REPORT	REPORT

Outcome

As happened, I could not finish the board over 10 because of the bad management of time and the overall work that come out.

The first and second weeks were on track but the code took longer than expected so, since the finals came out, I decided to make sure I passed the other subjects and left the board after the finals.

Since I had part of the code done, and all documented, it was easier for me to retake that again and, therefore, I was able to deliver it now.

Week	SKETCH	SKETCH	SKETCH	SKETCH	SKETCH	SKETCH	SKETCH
1st April	SOLDERING	SOLDERING	SOLDERING	SOLDERING	SOLDERING	SOLDERING	SOLDERING
2nd April	SOLDERING	SOLDERING	SOLDERING	SOLDERING	CODE	CODE	CODE
3rd April	CODE	CODE	CODE	CODE	CODE	CODE	CODE
4th April	CODE	CODE	CODE	CODE	CODE	CODE	CODE
1st May	CODE	CODE	CODE	CODE	CODE	CODE	CODE
2nd May	CODE	CODE	CODE	CODE	CODE	CODE	CODE
3rd May	DELIVERIES	DELIVERIES	DELIVERIES	DELIVERIES	DELIVERIES	DELIVERIES	DELIVERIES
4th May	FINALS	FINALS	FINALS	FINALS	FINALS	FINALS	FINALS
5th May	FINALS	FINALS	FINALS	FINALS	FINALS	FINALS	FINALS
1st June	FINALS	FINALS	FINALS	FINALS	FINALS	FINALS	FINALS
2nd June	CODE	CODE	CODE	CODE	CODE	CODE	CODE
3rd June	CODE	CODE	CODE	CODE	CODE	CODE	CODE
4th June	RETAKES	RETAKES	RETAKES	RETAKES	RETAKES	RETAKES	RETAKES
1st July	CODE	CODE	CODE	CODE	CODE	CODE	CODE
2nd July	CODE	CODE	CODE	REPORT	REPORT	REPORT	REPORT