Practical assignment of Digital Systems and MicroprocessorsCourse 2021-2022

Practice 2B

LSSudoku

	Login	Nom
Students	Adrian.Luerssen	Adrian Luerssen Medina

	Board	Report	Grade
Delivery			

Date	01/07/2022

Practical assignment of Digital Systems and MicroprocessorsCourse 2021-2022

Practice 2B

LSSudoku

	Login	Nom
Students	Adrian.Luerssen	Adrian Luerssen Medina

	Board	Report	Grade
Delivery			

Date	01/07/2022

Contents

Summary of the Statement

The aim of this practice was to create a controller for a java based game of sodoku. The main goal of the controller was to be able to store several different users, to be able trigger a game to start, to be able to then save the resulting score of the game, and to be able to display various pieces of information to the user in the meantime.

On start up, the user is presented with 2 choices, log in or register. If the user registers a new user, they can not register under a username that is already being used, and when logging, the username and password must match that of one of the saved users. As we do not have infinite space to store data, we only store 8 users, and if a new user is registered after that, we overwrite the oldest user.

Once the user has logged in the user is presented with a menu with the following choices:

1. Play game
2. Modify time
3. Show general Top 5 scores
4. Logout
5. Show time

Play Game

To play a game the username is sent to the java interface through the 9600bps serial channel, once the game commences any moves done through the keypad and joystick are sent to the java interface through the 9.6kbps serial channel. The time remaining is then sent to a terminal through the 1200bps serial channel. As I chose the additional serial channel (RX) optional, if something is received through the serial channel (1200bps) it is then sent to the java interface.

Once the game ends, whether that is that the user runs out of time, or the user ends the game through the java interface or keypad, the errors are displayed on the screen until 3 seconds have passed since the last error shown. After this, we display the score and time remaining to the user and save the score to the EEPROM, and finally we wait for a # on the keypad to return to the main menu.

Modify Time

To modify the time properly one must input all 4 new values of the game time, then press the # key. If at any point during the process the * is pressed, the edit to the game time is not saved and you are returned to the main menu.

Show General Top 5 Scores

This option shows the top 5 saved scores and rotates through them using the marquee refreshing the display every second.

Logout

The logout will display a short message saying “BYE BYE <USERNAME>” for 2 seconds before returning to the login register screen.

Show Time

This shows the system up time, which updates in real time. It shows the user how long the system has been on for.

System Design

General Functioning

Inputs

Data

Top Scores

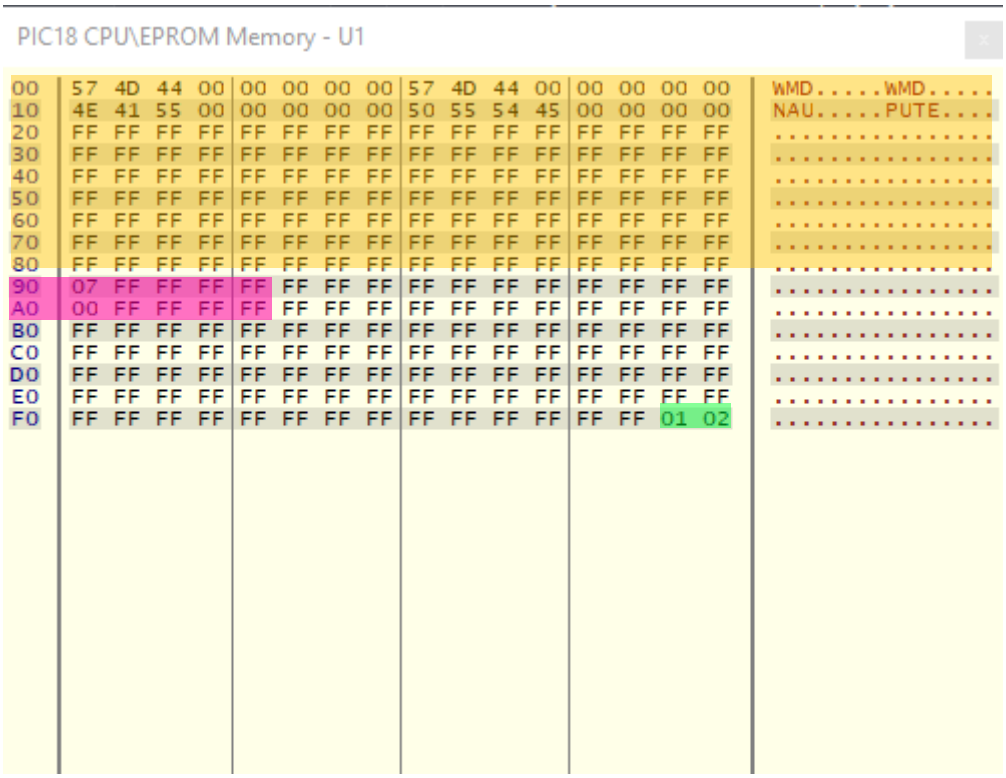
Here we store the general top 5 scores, we store the score value in positions 0x90 – 0x95, and the userNum in 0xA0 – 0xA5.

Position Pointers

These position pointers allow us to know what position the last user was to be written (circular queue), and the lastScore pointer allows us to know how many scores are currently saved. Position 0xFE being lastScore pointer, and 0xFF being the lastUser pointer.

User Information

The users are written to 16bits each the first 8bits are the username and the last 8bits are the password.



Microcontroller Configuration

Clock

The for the system clock I used the external 10MHz clock, however wanted to use an oscillation frequency of 40MHz for the timer. To do this we have to apply the PLL on the external oscillator, in this case we simply have to set the OSC to the HSPLL mode.

```
#pragma config OSC = HSPLL
```

Timer0

The timer0 interrupt time we used 0.833ms as for the additional serial transmission we need a baud rate of 1200bps, to do this we need a minimum timer0 interrupt time of 0.833ms. To achieve this we need to load the following values into TMR0L and TMR0H.

```
TMR0H = 0xDF;
TMR0L = 0x73;
```

ADC

For the ADC conversion, to allow us to do easy comparison and operations, we use left justification, this allows us to simply use the 8MSB and treat the conversion as 8bits rather than 10, without impairing the accuracy of the conversion too much.

```
ADCON0 = 0x01; // turning on adc conversion
ADCON1 = 0x0C; // setting AN2,AN1,AN0 to analogue
ADCON2 = 0x44; // setting conversion speed and left justification
```

EUSART

For the Eusart we need to transmit at 9600bps, to do this we need to configure it with the following values:

```
TXSTAbits.BRGH = 1; //highspeed baud rate
TXSTAbits.TXEN = 1; //enables transmit
TXSTAbits.SYNC = 0; //asynchronous

RCSTAbits.SPEN = 1; // enable serial port
RCSTAbits.CREN = 1; // enables receiver

BAUDCONbits.BRG16 = 0; // only 8bit SPBRGH

SPBRG = 255; // with FOSC=40MHz -> 9600bps
```

Electrical Schematic

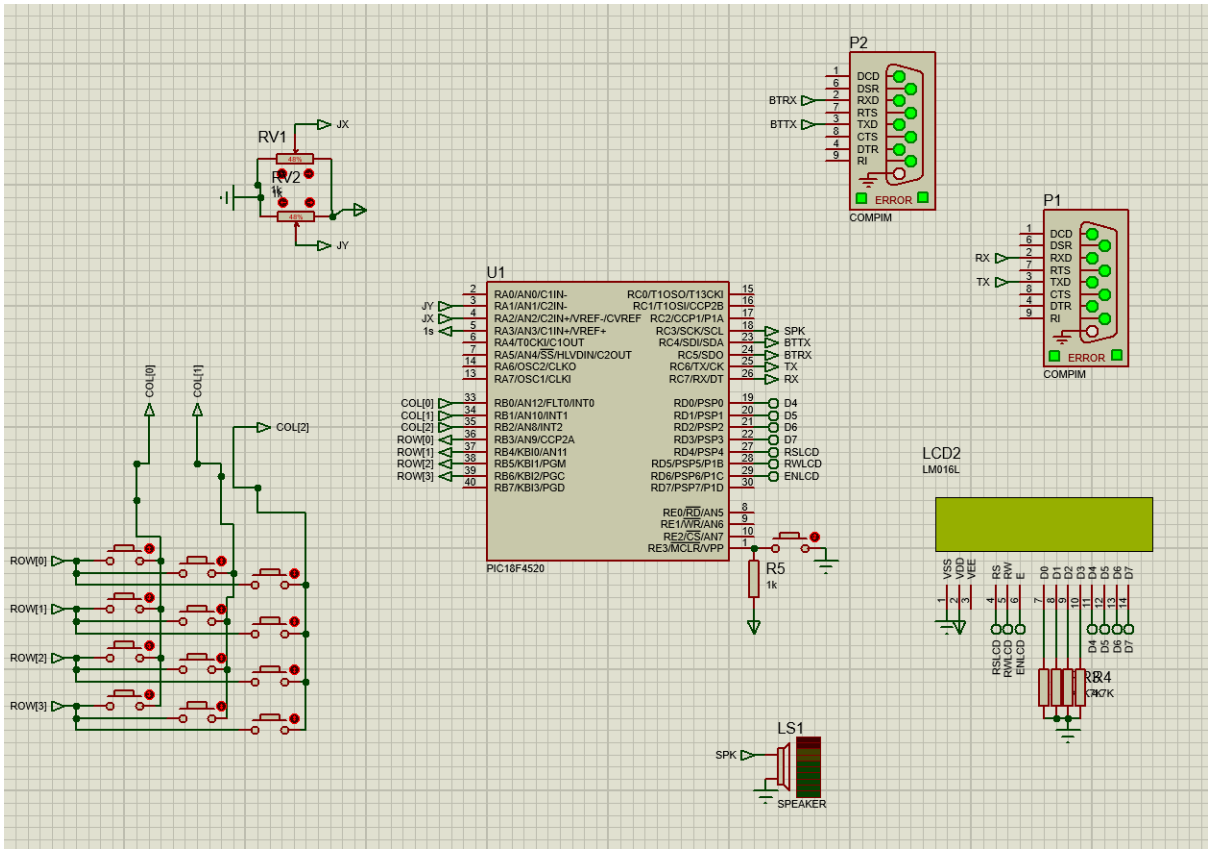


Figure 1 Electrical schematic

I had to use RA1 and RA2 as the joystick inputs as I had an issue with RA0 which was permanently set to ground.

ADTs

ADT Diagram

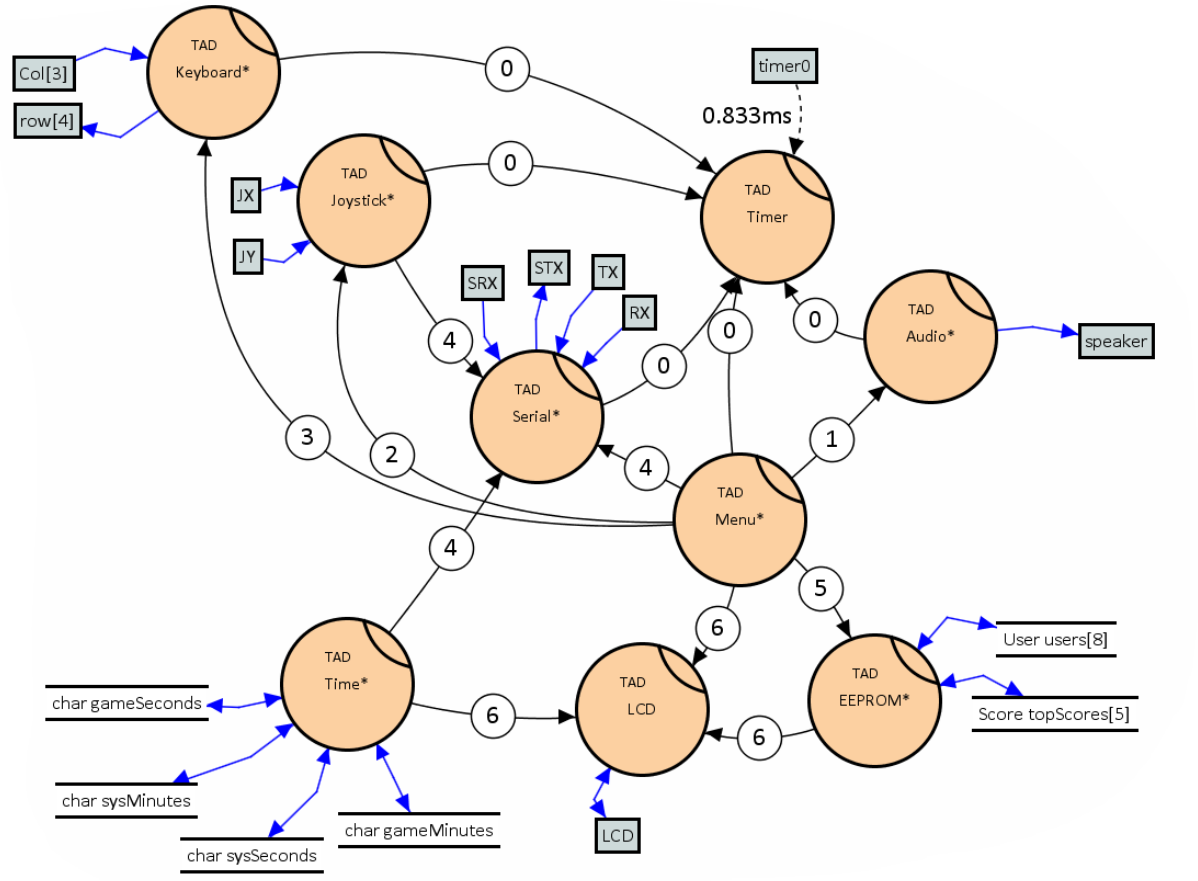


Figure 2 ADT diagram

This is the final ADT diagram, in the end I used 9 ADTs with 7 of which being motors. I created a ADT for each of the inputs and outputs to try to keep them as separate as possible. To this end the 'centre' of the design, and controller of the system is the menu ADT.

Motors

EEPROM

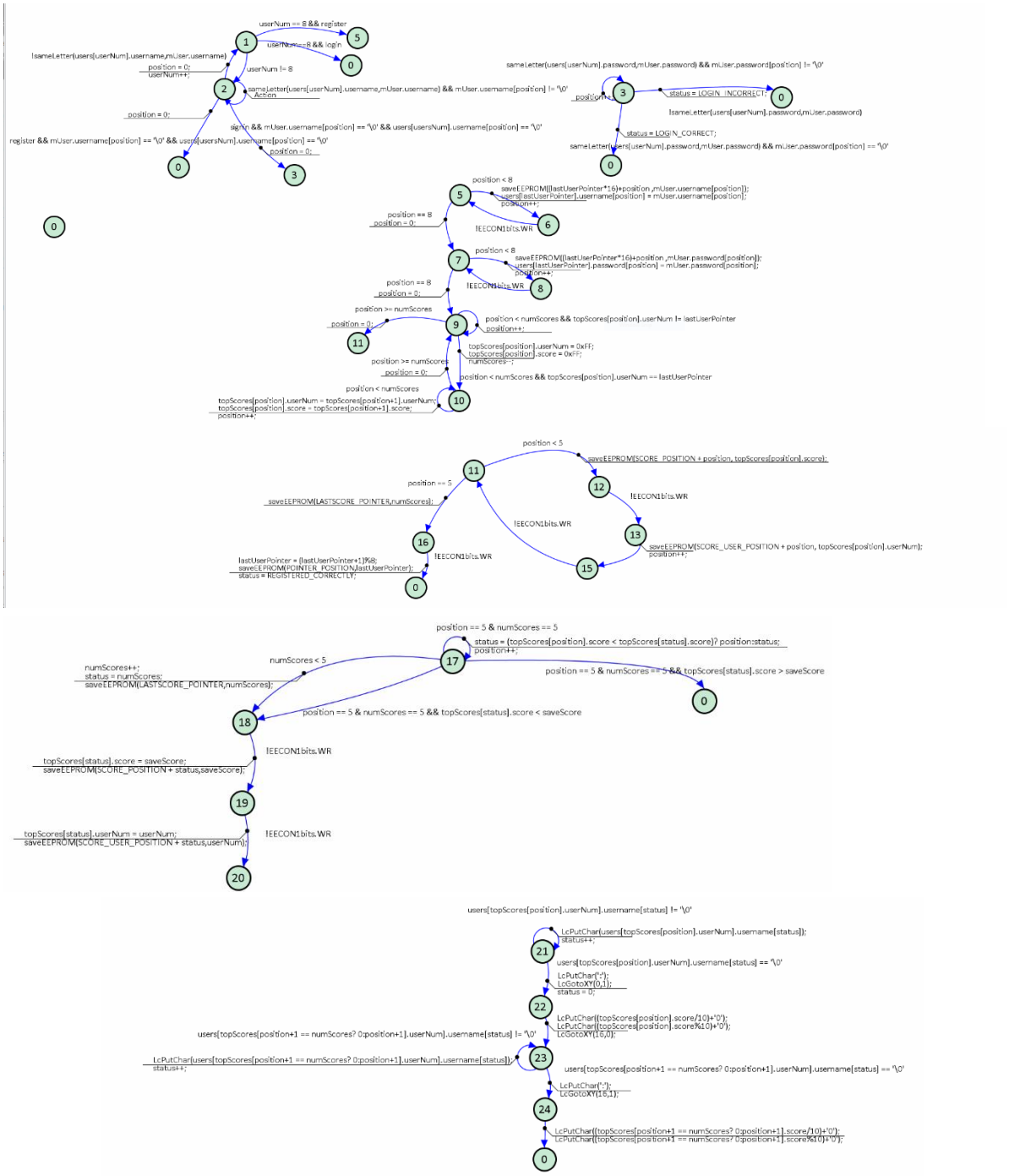


Figure 3 EEPROM motor

The EEPROM motor is one of the most complex of all of the motors, it takes care of writing newly registered users to the file, and also tests whether usernames and passwords are correct or already bieng used. In addition to this, when we write a new user, we check if the user that we are overwriting has any saved scores, if they do, we first delete them from the array of scores saved in the RAM, then write the entire array of scores back to the EEPROM. In addition to this, when we write a new score,

we need to overwrite the lowest score, we also take care of this by finding the lowest saved score, only if we have saved fewer than 5 scores previously.

Joystick

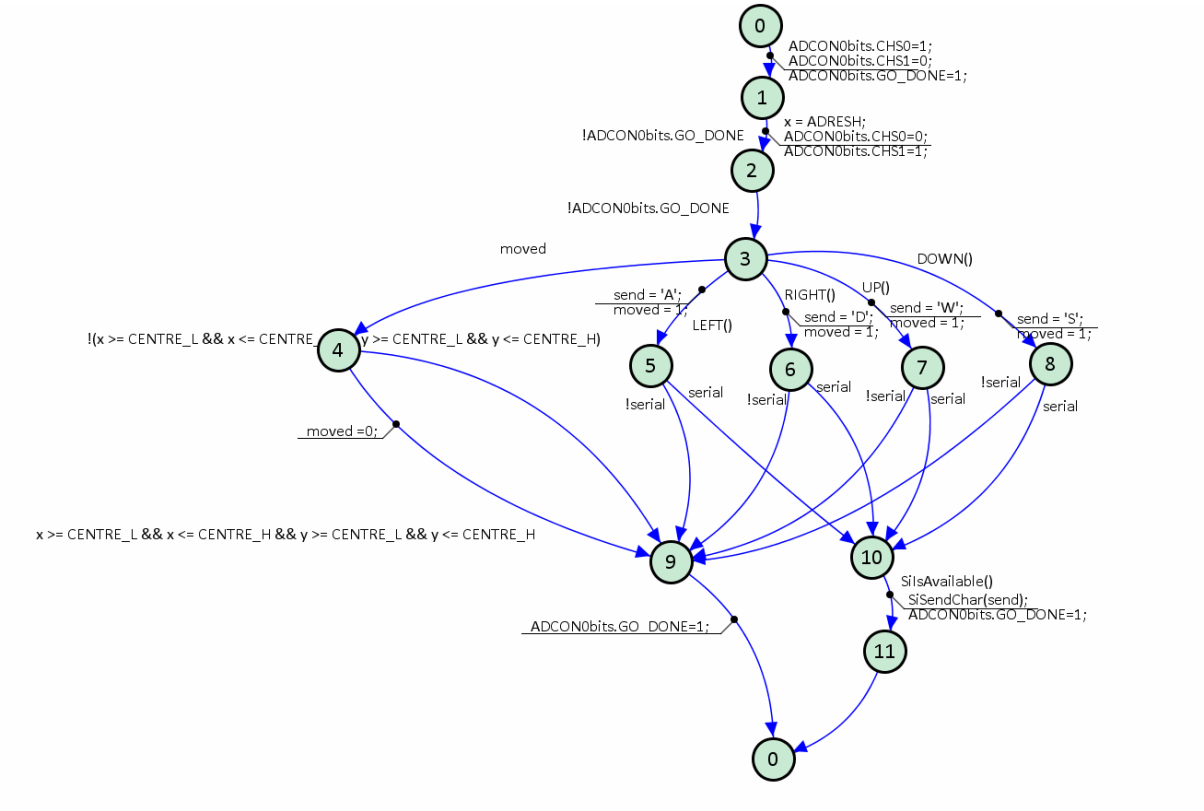


Figure 4 Joystick Motor

The joystick ADT takes care of the ADC conversion as well as controlling what happens with the resulting results of the conversion, making sure that before the joystick can be read in a different direction, that it has returned to the centre. This motor also controls the serial transmission of the direction of the joystick when in the game mode.

Keypad

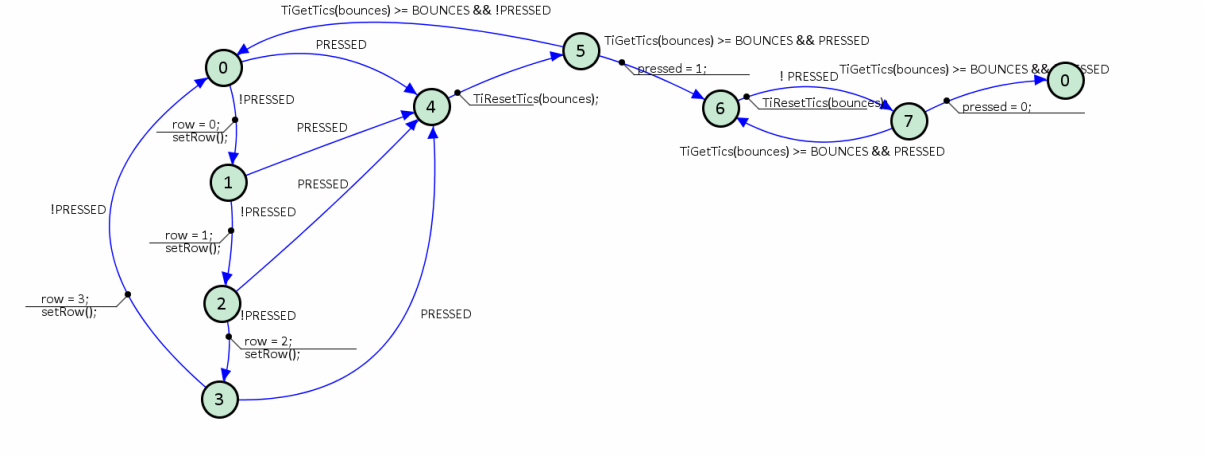


Figure 6 Keyboard Presses Motor

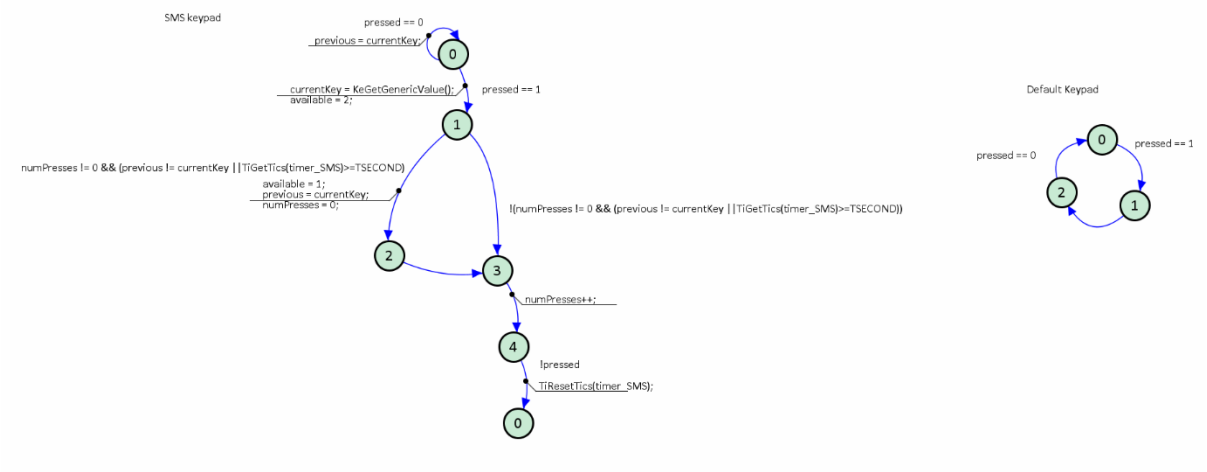


Figure 5 SMS Keyboard and default keyboard

The keyboard ADT motors are split into three separate motors, one is the motor that rotates through all of the rows and reads the key that is pressed and debounces any button presses.

There are also motors for the SMS and regular modes. If we are in the SMS mode, we rotate through the characters and set flags (available) to let the user know what state the keypress was in (1 = new press, 2 = rotated press). The regular keypad mode doesn't have any of this as we do not wish to rotate through the characters as we are playing the game.

Serial

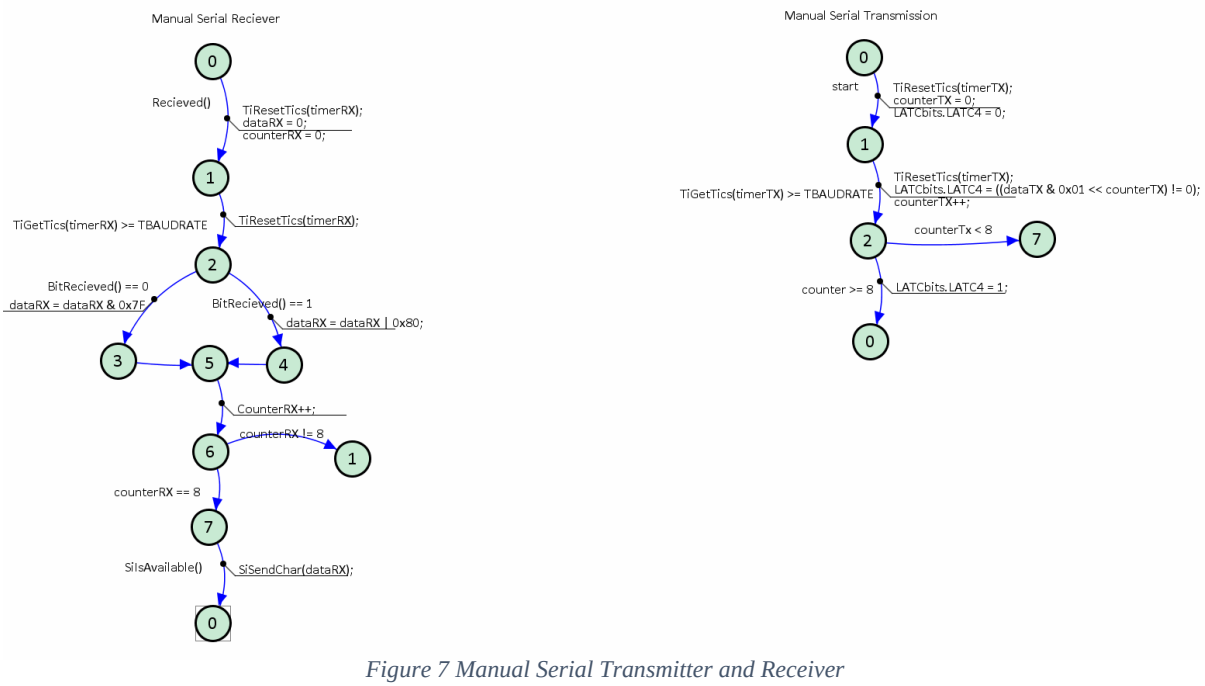


Figure 7 Manual Serial Transmitter and Receiver

These are two separate motors that take care of reading and transmitting the data through the serial channels not managed by the EUSART, when we receive a character, we send it to the serial channel managed by the EUSART, as that is what optional 'Additional Serial Transmission' asked of us.

Audio

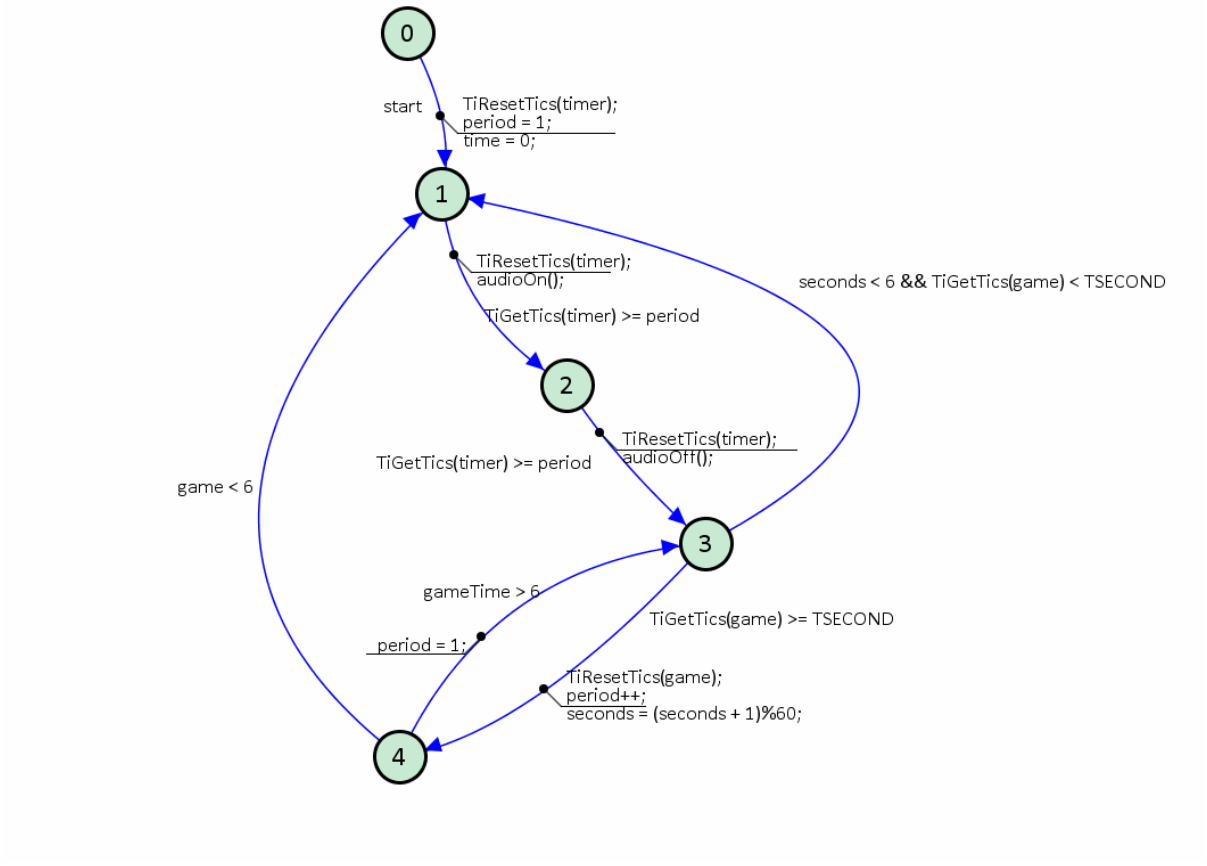


Figure 8 Audio Motor

This motor is for the audio, the aim is to have it activated through the startSong() function, then we play 5 notes, each second the notes period changes, after 5 seconds we then stop playing the song until 1 minute has passed.

Time

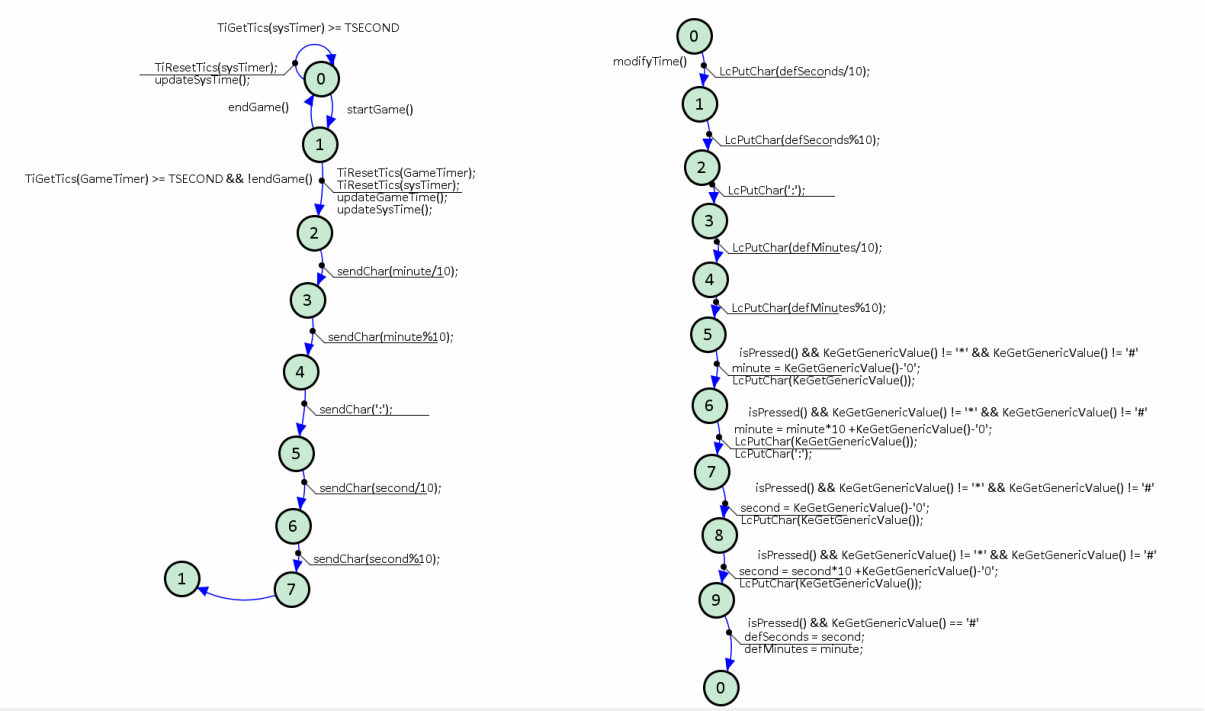


Figure 9 Time motor for game time and modify time

The time motor takes care of how much time has passed in the game and also the system time, it also takes care of sending it through the serial channel and displaying it on the display when necessary. However, we also have another motor which is for the modification of the time.

Menu

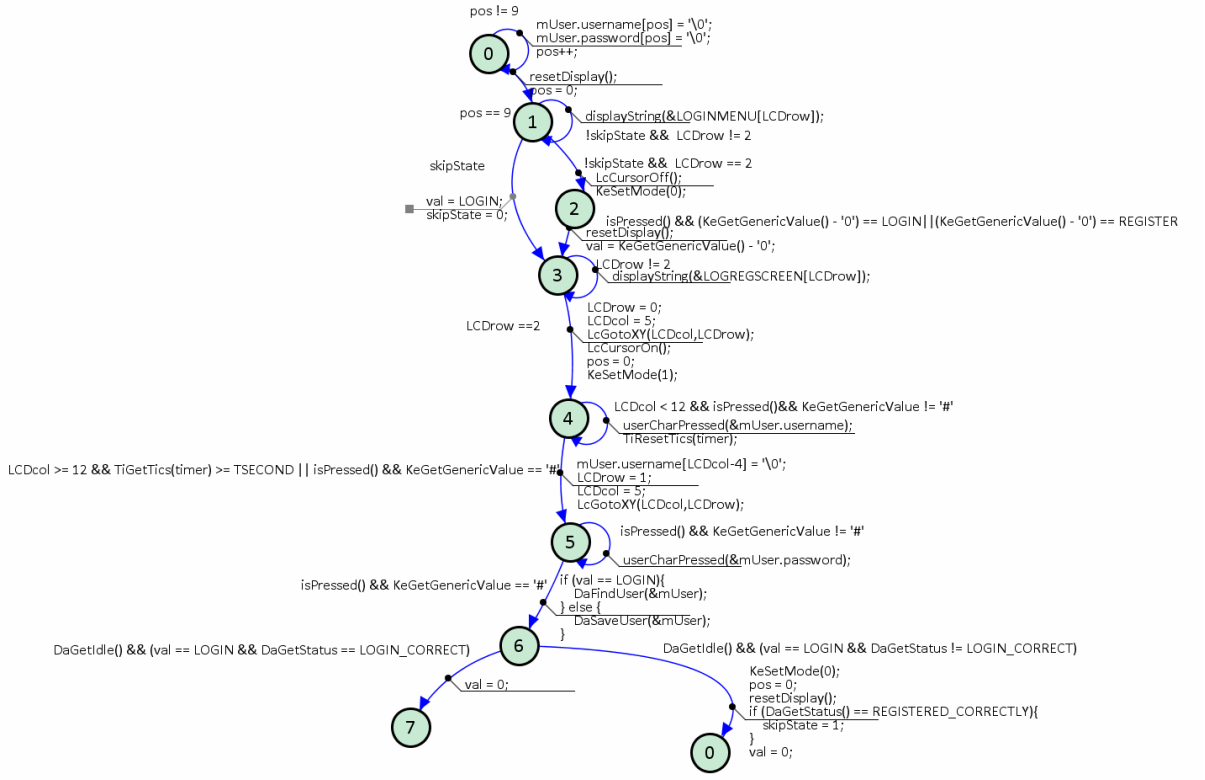


Figure 10 Menu Motor (Login/Register Part)

This section of the motor takes care of how I do a login and how we register a user, it uses the EEPROM functions to check the status of the register/login, and also uses the functions of the keyboard to get the keys pressed.

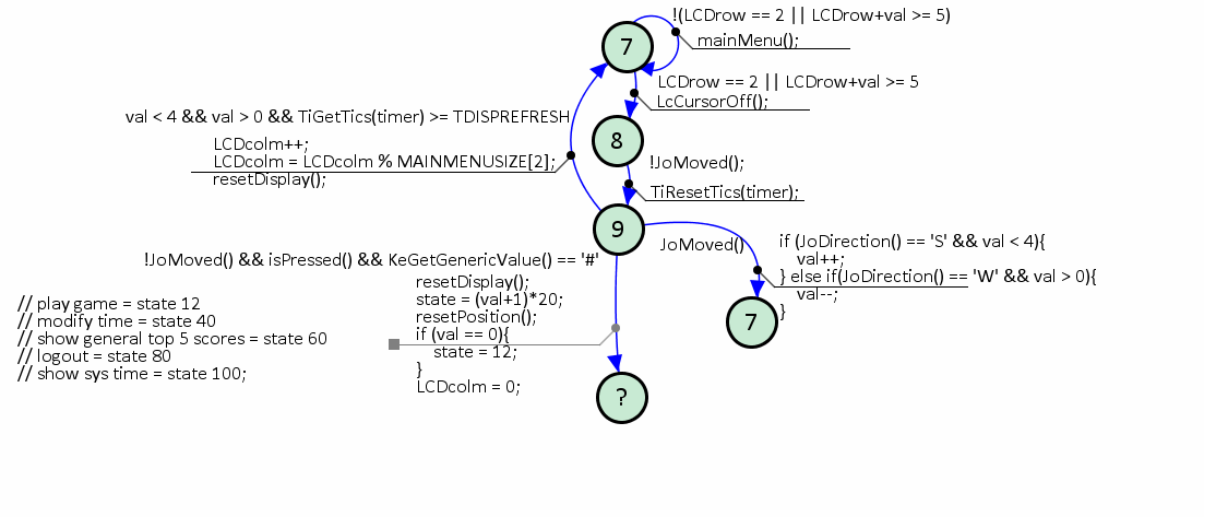


Figure 11 Menu Motor (Main menu Part)

These are the states that are used for the main menu and the navigation through it, it also contains the state '?', this is because I use arithmetic to calculate the next state depending on the choice the user made, this allows me to reduce the number of instructions as I do not have to do an if for each case. I

use state 12 for the game as I needed more space. This section communicated with the joystick ADT to ask for its current state.

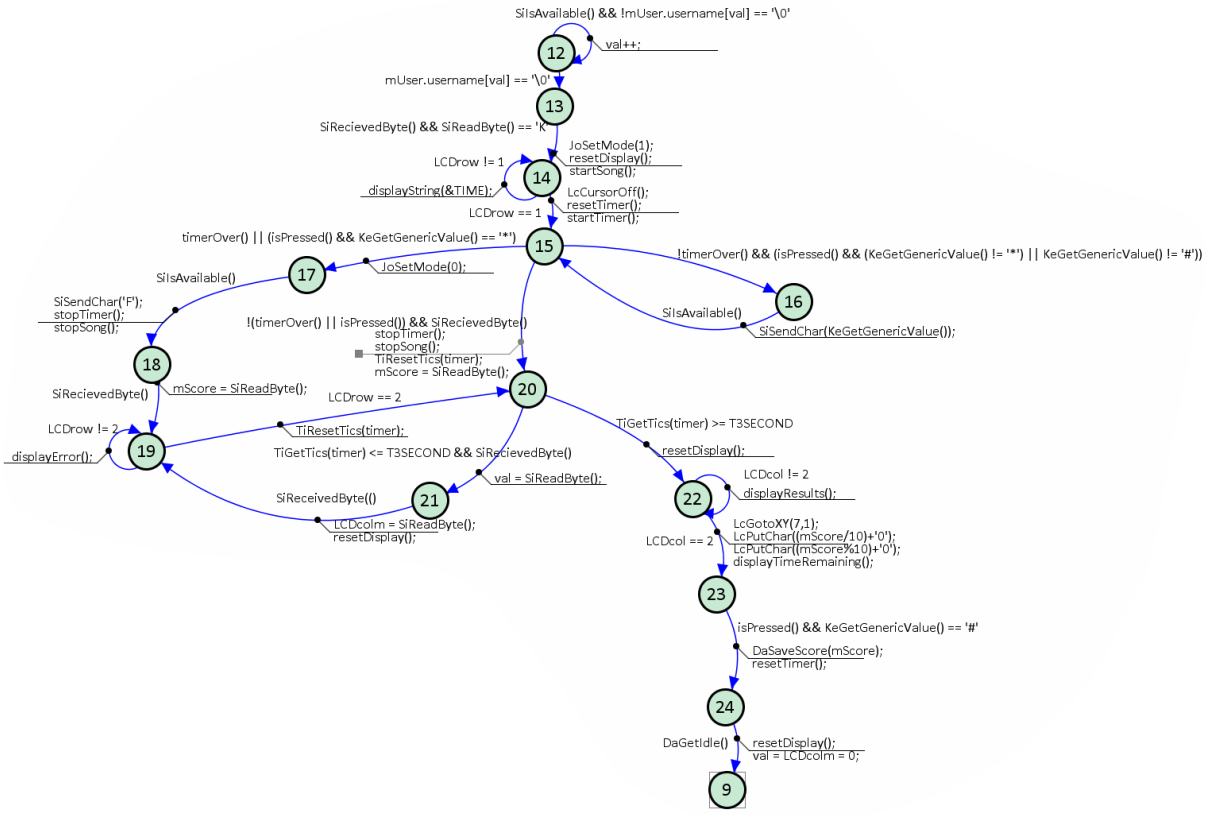


Figure 12 Menu Motor (play a game and save results part)

This is the motor to be able to play a game and show the errors, then subsequently return to the menu and save the score in the EEPROM and score structure. Here we also use the Timer and Audio ADTS as we want to play the song as we are playing the game, and we want the time to count down on the display and be sent to the serial channel.

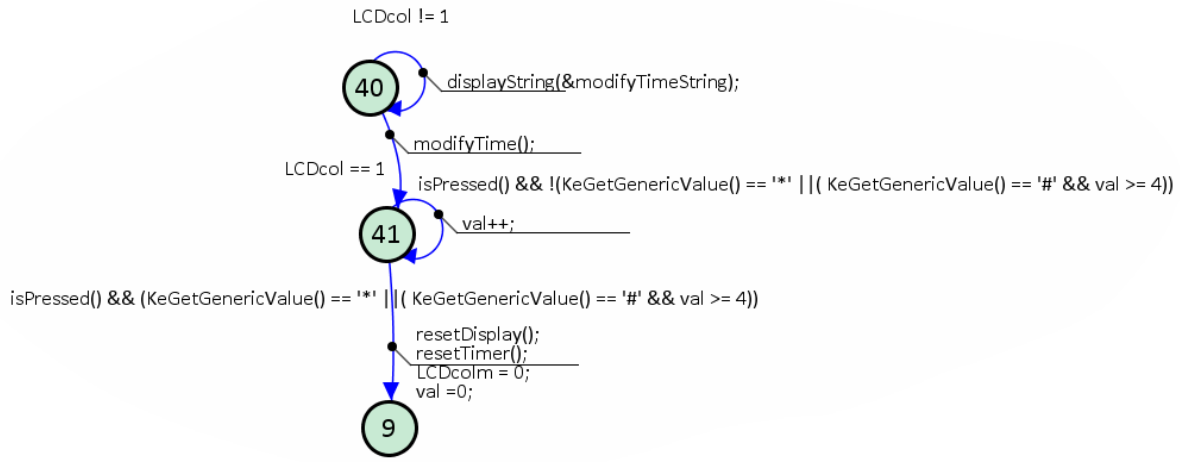


Figure 13 Menu Motor (modify the game time part)

In this part of the main menu motor is used to allow the user to modify the time of the game, this uses the time ADT for the majority of the logic.

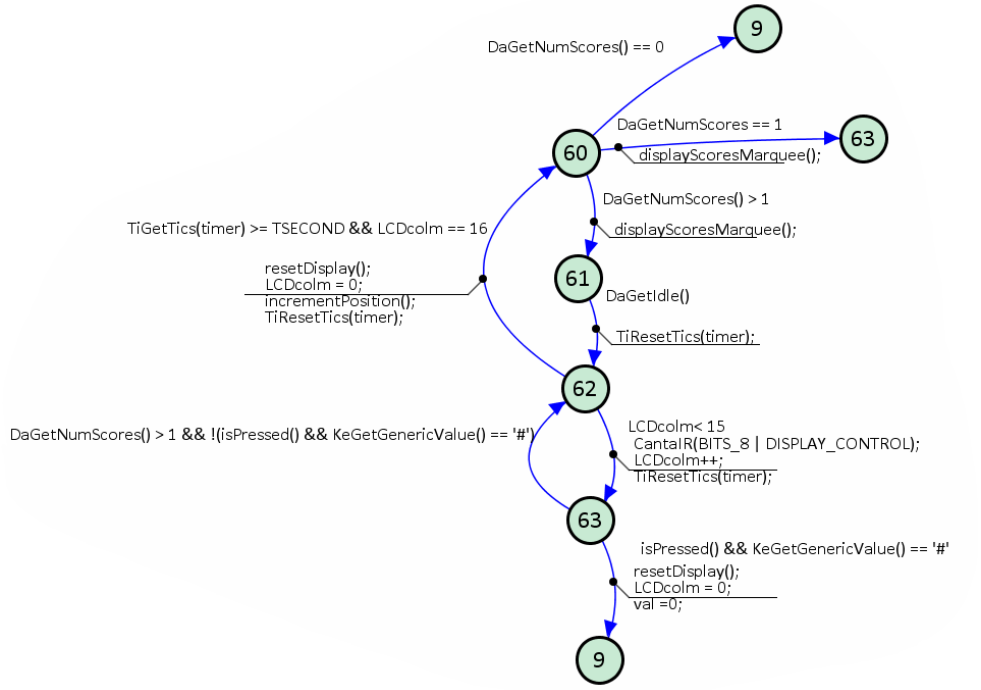


Figure 15 Menu Motor (display top 5 scores part)

In this section we allow the EEPROM ADT put the data into the Lc buffer, and we simply shift the display when necessary. We also do not allow the display to turn on if there are no scores saved, and if only one is saved we simply leave the display static and don't do the marquee.

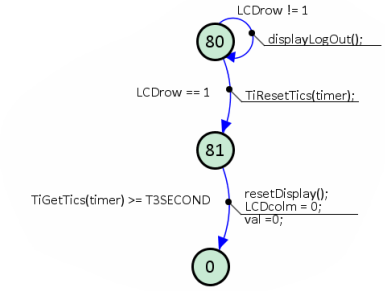


Figure 14 Menu Motor (log out part)

This is a fairly simple section of the motor, which displays the goodbye message and then returns to the login and register menu.

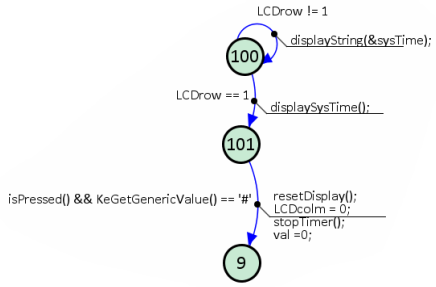


Figure 16 Menu Motor (display system time part)

This part of the motor again was simple as the Time ADT took care of the updating of the display.

Observed Problems

One of the main issues that I encountered was to do with memory issues, this issue arrived close to the end of the project when I was in the debugging stages, I found myself having to go back and edit old code that I had done as it was highly inefficient. This problem was mostly caused by my confidence and lack of awareness of how little program space the Pic had. I managed to solve this problem by reusing states or creating functions of commonly used instructions.

Planning

Expected Time Planning

	April			May			
	11-17	18 - 25	25-30	1 - 7	7 - 14	14 - 21	21 - 28
Planning							
Designing ADT + Motors							
Implementation							
Debugging							
Report							

Actual Time Planning

	May		June					July
	15-22	22 - 28	1 - 7	7 - 14	14 - 21	21 - 28	29 -31	1 - 3
Planning								
Designing ADT + Motors								
Implementation								
Debugging								
Report								

Reason for Deviation

I severely underestimated how much work this project would be, especially undertaking it alone. The time deviation also had to do with personal health issues, but also with other classes and their projects. These are the reasons that I started the project so late. The large gap from the 18th to the 25th of june is due to the trip to San Francisco, due to the activities and lack of resources, it was difficult to progress much at the time.

Conclusions

This practice was very challenging, especially as I had to tackle it alone. I feel that I learned a lot of positive things in this project, especially that I should be very mindful of how I approach a system and to always take into account that we do not have infinite resources. This alongside the segmentation of the problem were my two main take aways from the project. It had a steep learning curve, but it was enjoyable to complete as it was easy to see that you were making progress.

The use of a speaker, LCD screen and Bluetooth module were all new concepts to me and it was fun to work with all of them in one project. The additional serial channel was one of the more challenging parts conceptually, as it took me a while (with help) to find the correct way of reading and writing the data properly from the serial channel.