Practical assignment of Digital Systems and Microprocessors

Course 2022-2023

Practice 2B

LSControl Tower

	Login	Name
Students	joaquin.bressan	Joaquin Bressan Nazar
	pau.coderch	Pau Coderch Masallera

	Board	Report	Grade
Delivery			

Date	

Copy for the students

Practical assignment of Digital Systems and Microprocessors

Course 2022-2023

Practice 2B

LSControl Tower

	Login	Name
Students	pau.coderch	Pau Coderch Masallera
	joaquin.bressan	Joaquin Bressan Nazar

	Board	Report	Grade
Delivery			

Date	

Cover of the report

INDEX

Summary of the statement

Port Menu:

First the user selects the tower by entering a port code using the keypad, which generates different sounds for each key press. The system follows SMS format and displays the entered code in real-time on the LCD. The (#) key sends the code to a Java interface, which confirms readiness. Once communication is established, the microcontroller sends the tower's name and displays the main menu.

Main Menu:

The main menu of this practice consists of five options, but due to the limited space on the 2-row, 16-column LCD display, only two options can be shown at a time. To navigate through the menu, a joystick is used to move up or down. The "#" key is used to select the option, which will always be the one displayed on the top row of the LCD.

Start Recording:

In the "Start Recording" mode of this practice, users can make voice recordings through a microphone to communicate with control towers. Analog samples are acquired at a constant sampling frequency and sent to the tower via a Java interface. The Java interface receives the samples and recording information, generating an audio track that is displayed in the program view. The user can select and play recordings either from the Java interface or the microcontroller. A USB-to-TTL converter and the USART of the PIC are used, and the desired baudrate needs to be configured. Each recording has a duration of 8 seconds, and an indicative message is displayed on the LCD when a new recording starts. The microcontroller sends the character 'D' to indicate the start of a recording. The communication protocol and data exchange between the microcontroller and Java interface need to be implemented, including the transmission of digital microphone samples, timestamp, and index. After recording, a 5-second

melody is generated using a speaker, and the user enters the recording name on the Java interface. The PIC stores the index and timestamp for future playback and ensures that the recordings are not lost by storing a copy in non-volatile memory, overwriting the oldest recording if necessary.

Play Recordings:

In the "Play Recordings" mode, a list of recorded tracks with their corresponding index and timestamp is displayed. Users navigate through the list using the joystick, and the (#) key selects the highlighted option. When a recording is selected, the microcontroller sends a 'P' to the Java interface, which responds with 'K'. The microcontroller then sends the recording index in ASCII format, followed by '\0', and the 8-second recording plays through the computer's speakers. After playback, the interface sends 'F' and a 5-second melody is generated using the microcontroller's speaker. If no recordings exist, a message prompts the user to make a recording, and the (*) key returns to the main menu.

Show Current Time:

In this mode, the LCD screen will display the minutes and seconds that have elapsed since communication was established with the port (since the main menu was first displayed after entering the port name). This time must be updated in real-time, so for every second that passes, the LCD should be updated. The format must be MM:SS, as shown in the picture. If the (*) key is pressed, the user will return to the main menu.

Modify Current Time:

In the "Modify Current Time" mode, users can adjust the current time of the system. The time is entered using the matrix keyboard on the second line of the LCD, following the format MM:SS. Pressing (*) returns to the main menu without saving the time. When

entering the minutes, the system displays a colon (:) before entering the seconds. Pressing (#) without completing the time keeps the user in this mode until the time is fully entered and (#) or (*) is pressed.

End Communication:

In the "End Communication" mode, when accessed, a farewell message "bye bye <TOWER>!" will be displayed for 2 seconds to indicate the end of communication with the control tower. Afterward, the access menu will be returned, allowing the user to establish communication with another tower by entering its name.

System Design

The code structure revolves around the main source file, which is responsible for handling the timer interrupt and initializing various components of the program. Within our main.c file, there is a while loop that controls the execution of different motors in our system. These motors include the audio motor, which manages the playback of music and sounds, the keypad motor, which handles user input from the physical keypad on the board. Additionally, the keypad motor also incorporates the SMS motor, which detects rapid key presses within a second to determine the user's intended action. Each motor serves a specific function in our system.

Among these motors, the menu motor plays a pivotal role as it acts as a central point for coordinating the flow of functionalities across different C files. It serves as a control mechanism for navigating through the various functionalities.

In summary, our system design is centered around the main source file, where the menu motor controls the overall flow of functionalities. Other C files are dedicated to handling specific aspects of the code, and when integrated into the menu motor, they collectively fulfill the required functionalities for this practice.

Configurations

Memory:

Throughout our project, we made use of both the Flash and EEPROM memories of the PIC18F4321.

Flash memory:

The flash memory is used to store the program code that runs on the microcontroller. It is where the instructions for the microcontroller's operation are stored. This memory being volatile, we had to use the EEPROM memory in order to store the necessary data regarding the recordings.

Non-volatile memory:

In the EEPROM of the PIC18F4321, we store the data of the recordings, the number of recordings, and the position within the array of recordings to ensure the FIFO functionality of the storage. Addresses 0x00 and 0x01 are used for the total number of recordings and the array position respectively.

Moreover, addresses from 0x10 to 0x17 are used for the indexes of the recordings received from the interface.

Finally, addresses from 0x20 to 0x27 and 0x30 to 0x37 are used for the minutes and seconds of the recording's timestamp respectively.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	08	09	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
10	08	01	02	03	04	05	06	07	FF	FF	FF	FF	FF	FF	FF	FF	....
20	0E	00	00	01	01	02	03	0D	FF	FF	FF	FF	FF	FF	FF	FF	....
30	01	11	22	0A	1C	3A	10	18	FF	FF	FF	FF	FF	FF	FF	FF	.."..:.. ....
40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
50	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
60	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
80	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
90	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....
F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	....

Clock:

Initially, we started the practice using a frequency of 10 MHz. After implementing most of the required functionality, we realized that we needed to switch to a frequency of 40 MHz to enable the recording section. To achieve this, we configured the oscillator (OSC) to HSPLL.

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

Configuration

Eusart:

This is the configuration that we used in order to be able to connect to the Java Interface.

In the Java Interface we set a baud rate of 57600.

```
SPBRGH = 0;
SPBRG = 172;
BAUDCONbits.BRG16 = 1;
BAUDCONbits.ABDEN = 0;
TXSTAbits.BRGH = 1;
TXSTAbits.SYNC = 0;
TXSTAbits.TXEN = 1;
RCSTAbits.SPEN = 1;
RCSTAbits.CREN = 1;

TRISCbits.RC6 = 1;
TRISCbits.RC7 = 1;
```

Configuration

ADC:

We use the ADC for the joystick functionality as well as the microphone's. The function was the same for both, we only had to modify the Analog Channel Select bits (CHS [3..0]) to set the AN channel for the conversion.

We use left justification.

```
ADCON0 = 0x03;
ADCON1 = 0x0C;
ADCON2 = 0x44;
```

Configuration in the code

Timer:

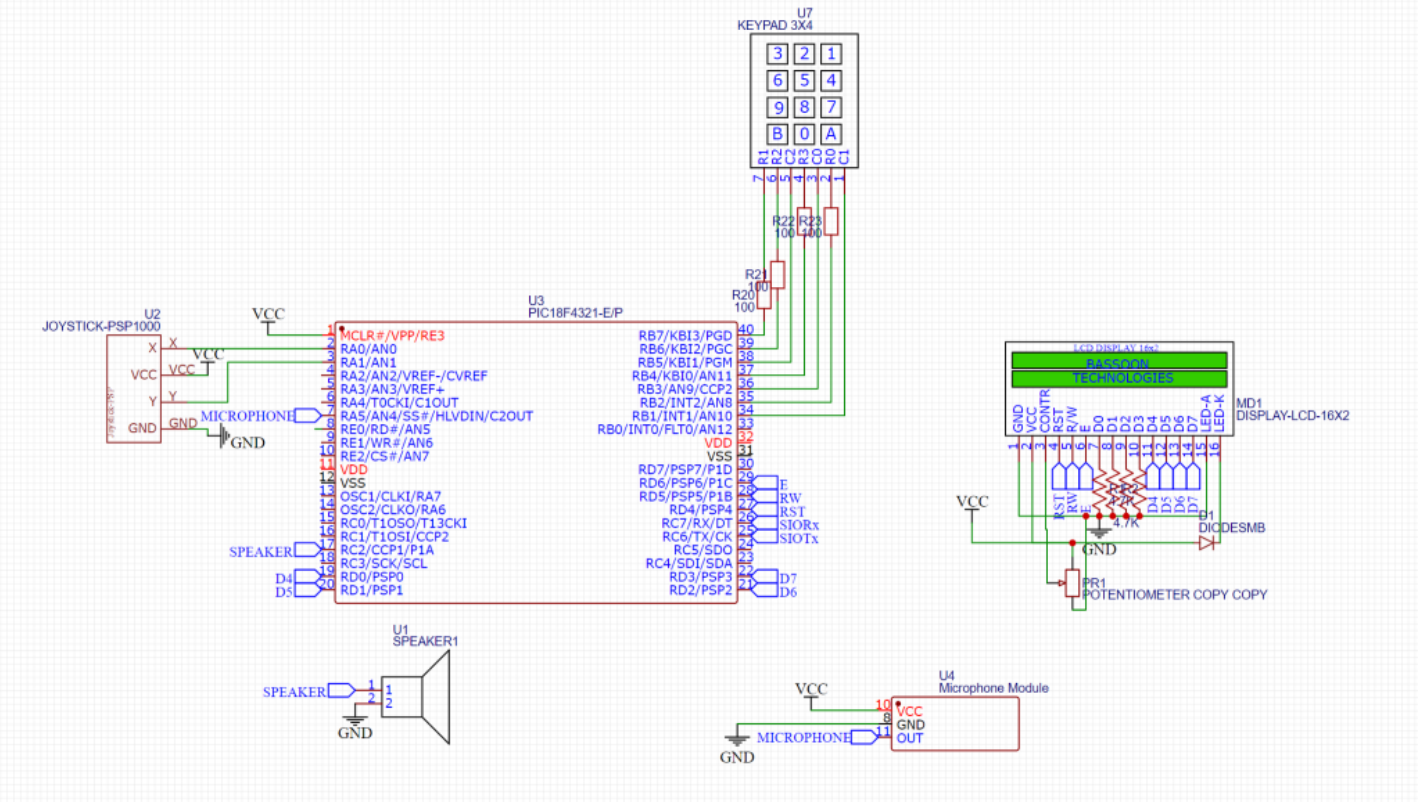
We reset the Timer Interrupt at the value of 65223, which corresponds to 0,25ms → 250µs.

For that we need to update the value of TMR0.

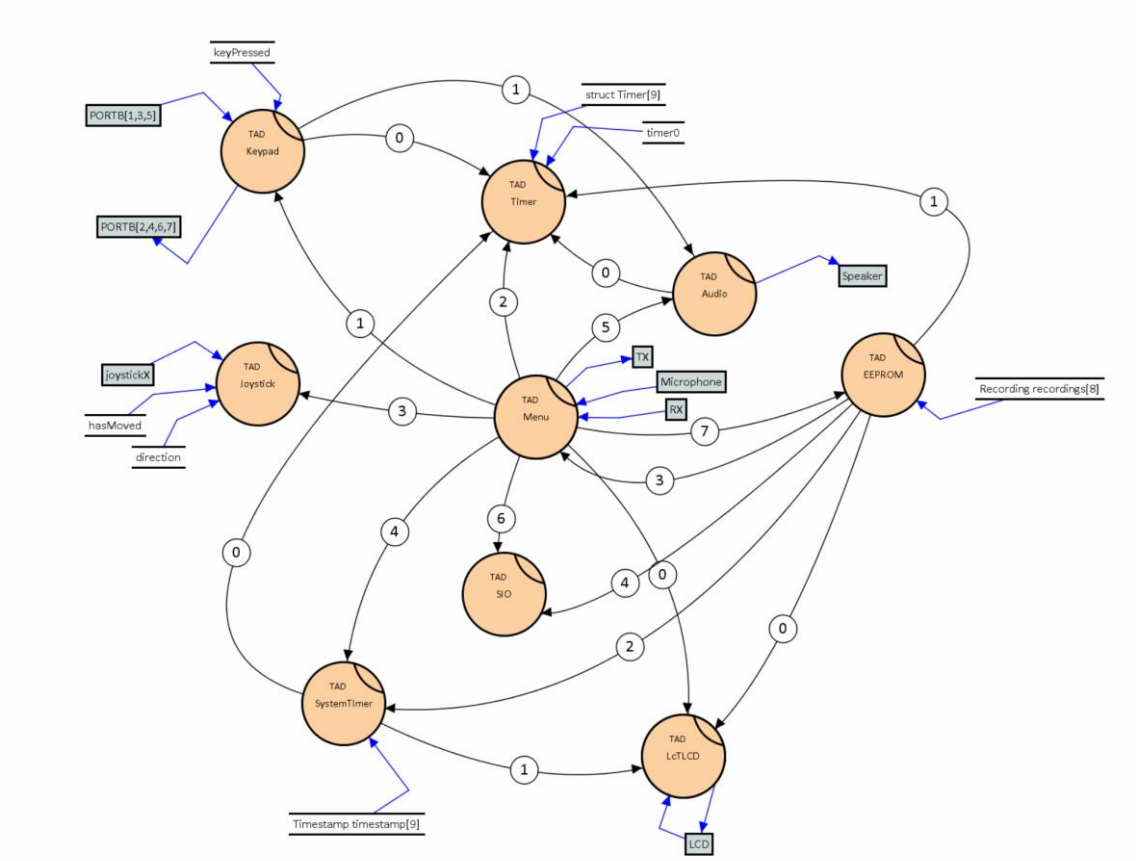
```
#define T0CON_CONFIG 0x82
#define RELOAD_TMR0 65223
```

Configuration

Electrical Schematic



TAD Diagrams



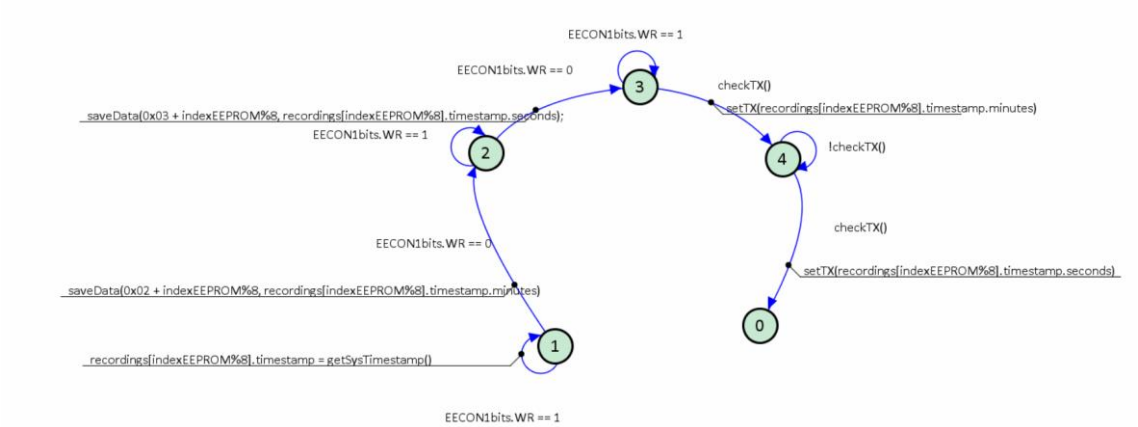
This is the TAD diagram of phase 2B. Here we can find the different peripherals used:

- Keypad for PORTB[1,7]
- Joystick X-axis analog value
- LCD display
- Speaker
- Microphone
- USB-to-TTL to communicate with the Java interface (TX and RX)

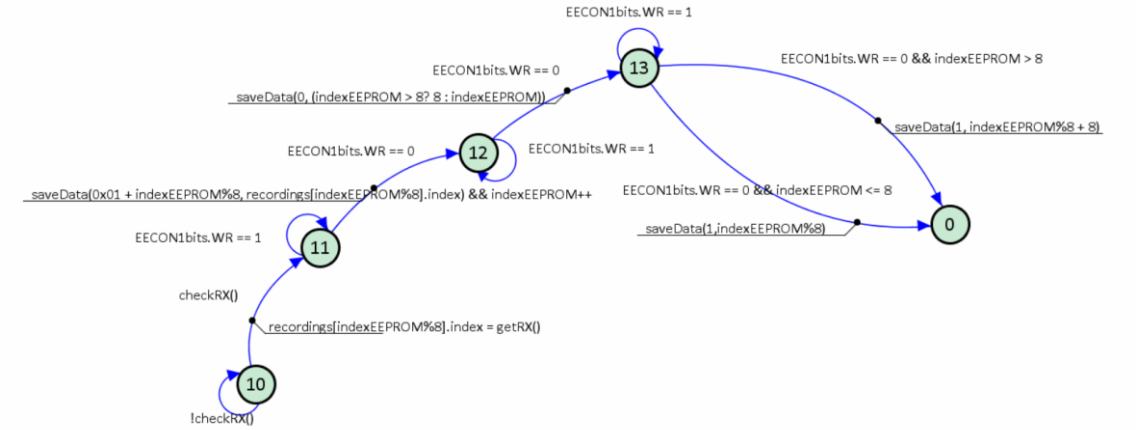
Moreover, we also listed the most important variables of the different ADTs. In total, we can find 9 ADTs, and within these ADTs, a total of motors (two for the keypad). It is important to notice how the TAD of the system revolves around the menu ADT. This can be seen in the code itself, where the menu plays the role of a controller, making use of the different files and methods in an efficient way.

EEPROM

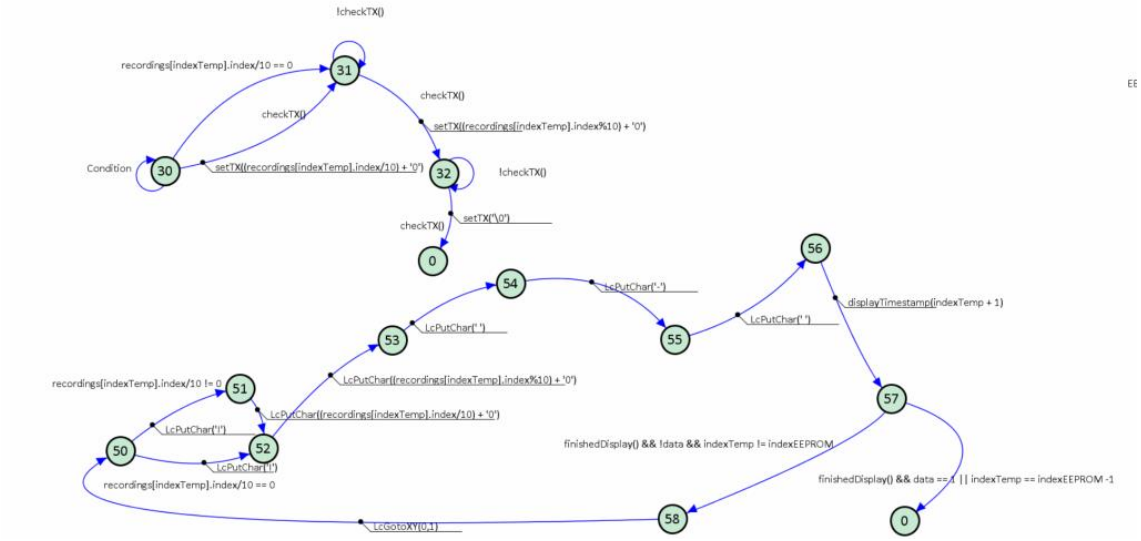
The EEPROM motor takes care of storing the necessary non-volatile data regarding the different recordings (indexes and timestamps), number of recordings, and position of the recording in the circular queue storing them. This queue is an array defined within the Flash memory of the PIC18F4321.



In this loop is handled the storing of the recording's timestamp within the EEPROM and the transmission of this same timestamp to the Java Interface. We first retrieve the system timestamp through the getSysTimestamp() method before storing it in the array of recordings. We then save it in the specific EEPROM's address before transmitting it to the Java Interface. The program then returns into the Idle state of the EEPROM motor.

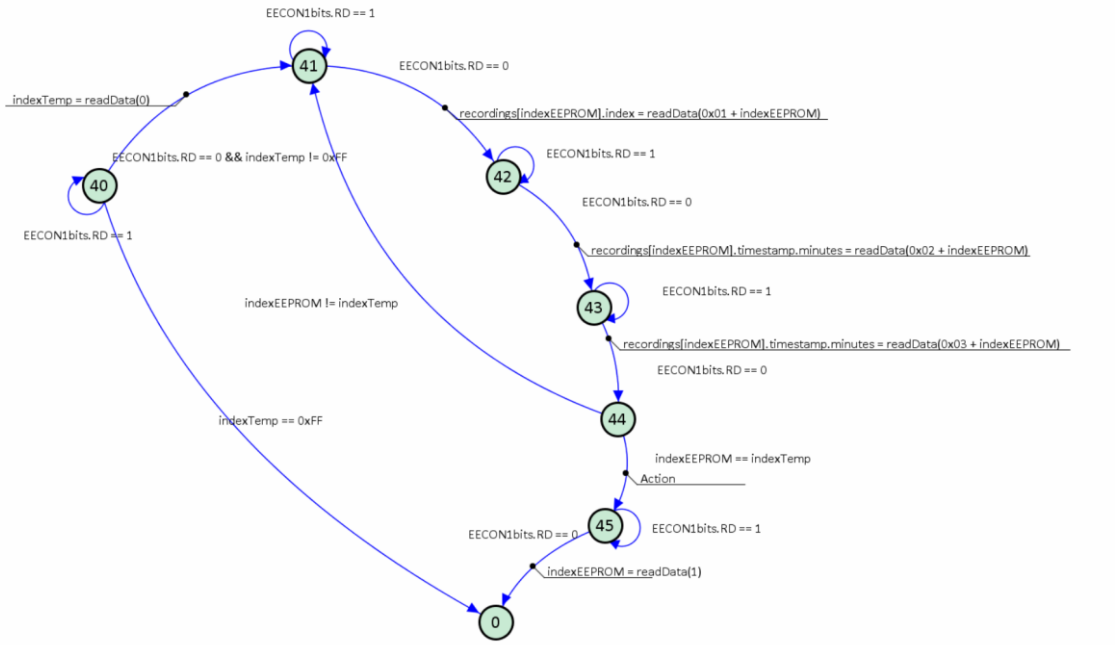


Here we first receive the index from the Java interface and store it within the array of recordings. Afterwards, we save the index's data within the EEPROM before updating the indexEEPROM indicating the number of recordings. We finally store the number of recordings and position within the array in the EEPROM memory and then return to the Idle state.



From state 30 to 32, we retrieve the specific index of the recording and transmit it to the Java interface in order to play the recording with that index before returning to the Idle state.

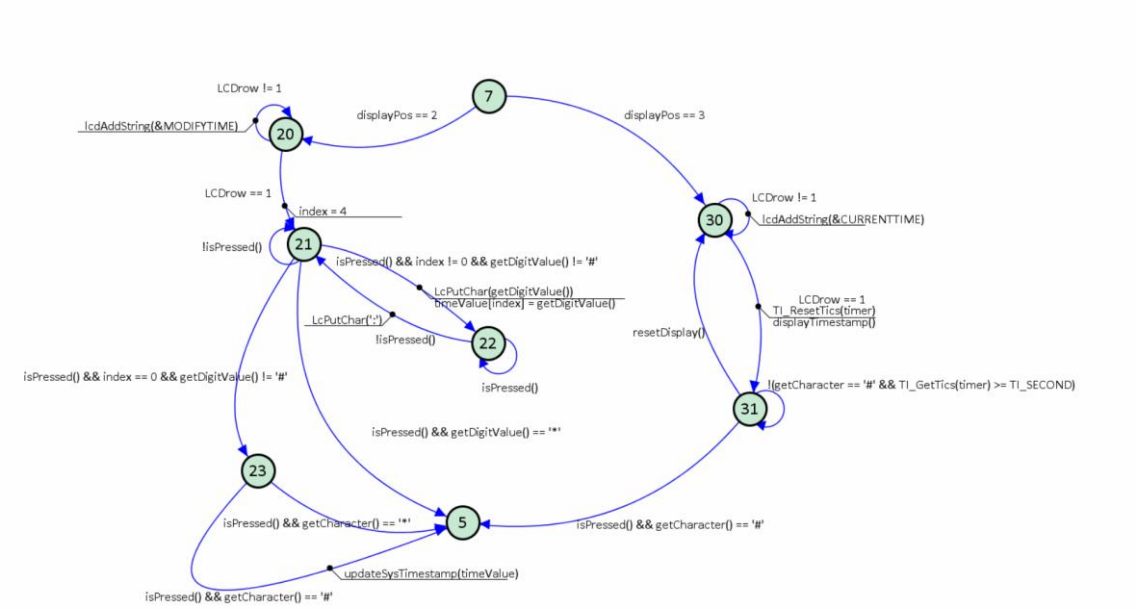
From states 50 to 60, we handle the display of the recordings (index and timestamp). We use methods of the LCD display, printing the necessary characters consecutively. State 58 is reached if two recordings' information have to be displayed. This does not happen in the case the user reached the last recording. Otherwise, we return to the Idle state of the EEPROM motor.



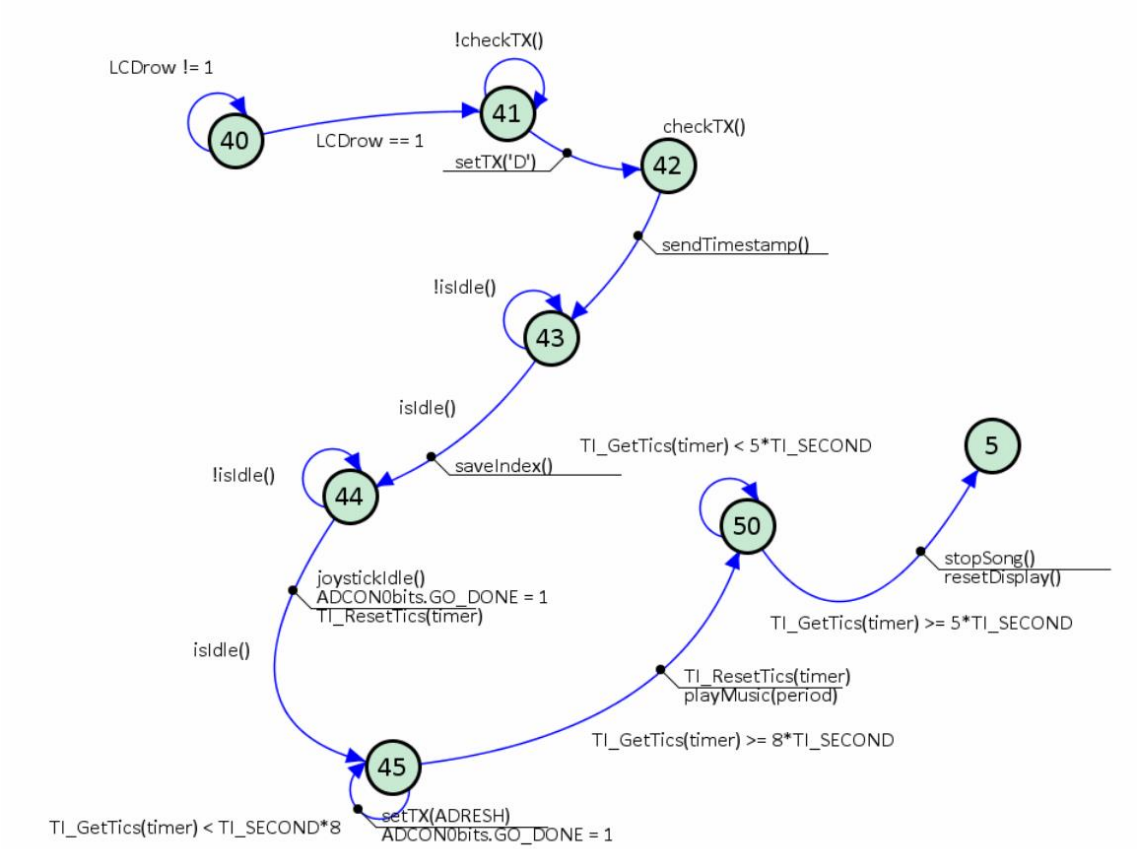
States 40 to 45 are implemented to read the EEPROM and store the information in the flash memory when the user is executing the program. This only happens thanks to the non-volatile memory of the EEPROM. We read and store from the EEPROM to the array of recordings. Whenever the temporary index reaches the total number of recordings retrieved in state 40, the motor returns to its idle state.

Menu

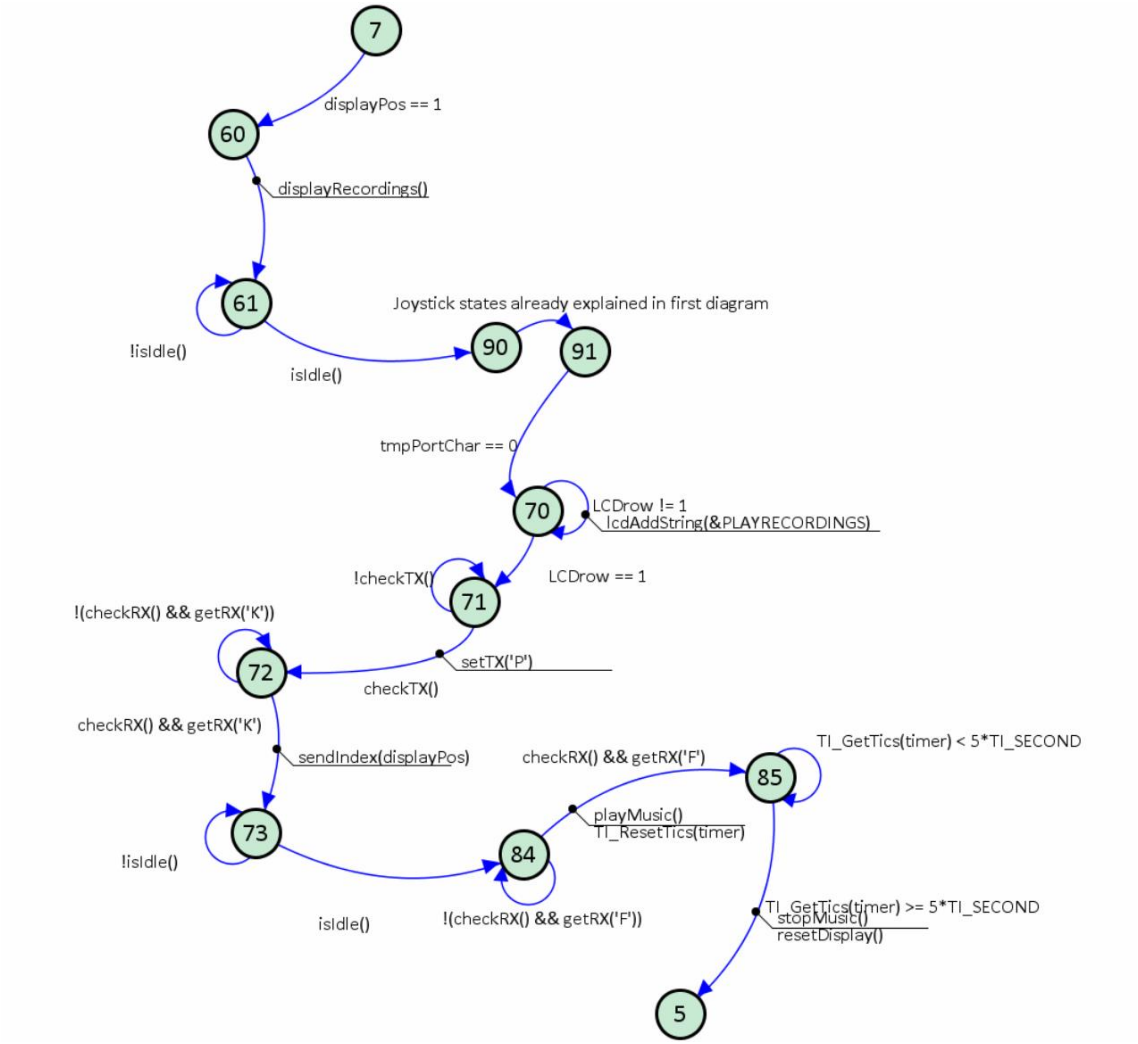
This ADT diagram represents the first part of the menu motor. The LCD display is reset before displaying the PORTNAMEMENU string through the lcdAddString() method. We verify that all the characters were printed through the LCDrow variable. The user is then allowed to enter characters through the keypad. The position in the LCD is consistently updated, staying in the same position or going to the next one depending on the keys pressed. After making sure the user entered 3 characters before pressing the '#' key, the port name entered is sent to the Java interface before displaying the menu in the same way as the port name string. After printing it, we verify the joystick is in the centre. If the joystick is then moved up or down, we update the displayPos variable, according to the menu options that need to be displayed. Otherwise, the scroll variable is updated every second for the marquee to take place. In the case where the joystick is not moved and the '#' key is pressed, the program is redirected to the specific state of the menu selected.



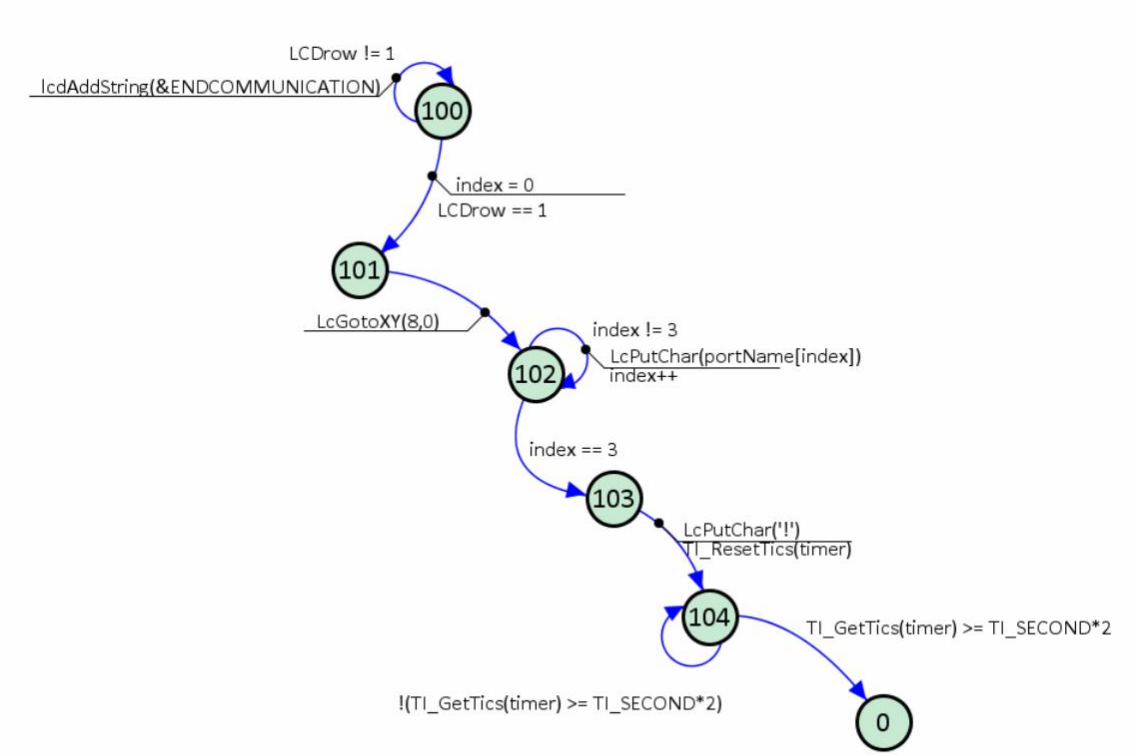
This diagram represents the functioning of options 3 and 4 of the menu: modify time and current time. In both cases, it starts by displaying the option's corresponding string. For the current time, the timestamp of the system is then displayed. The program changes states when a second passes by (it resets the display in order to update the time), or when the '#' is pressed, returning to the menu.
Regarding the modify time option, the program waits for the user's input. After inputting 2 digits, it prints the ':' character. It then waits for two more digits before changing state, waiting only for '#' or '*' characters. If '#' is pressed, the time is updated and the menu displayed back again. Otherwise, the same happens, except for the time update.



Here is where the recording takes place. We first display the recording string, send the timestamp to the Java interface and receive the new recording's index. When the EEPROM finished saving, we enter the recording loop: state 45. Previously reset, we wait for the timer to reach 8 seconds while sending the samples directly from the ADRESH register to the Java interface. After 8 seconds pass by, state 50 is reached. Music is played through the speaker with the playMusic() method for 5 seconds. Finally, the display is reset and the menu is displayed back again.
It is important to note that we disable the joystick for the recording in order to avoid any issue with the ADC.

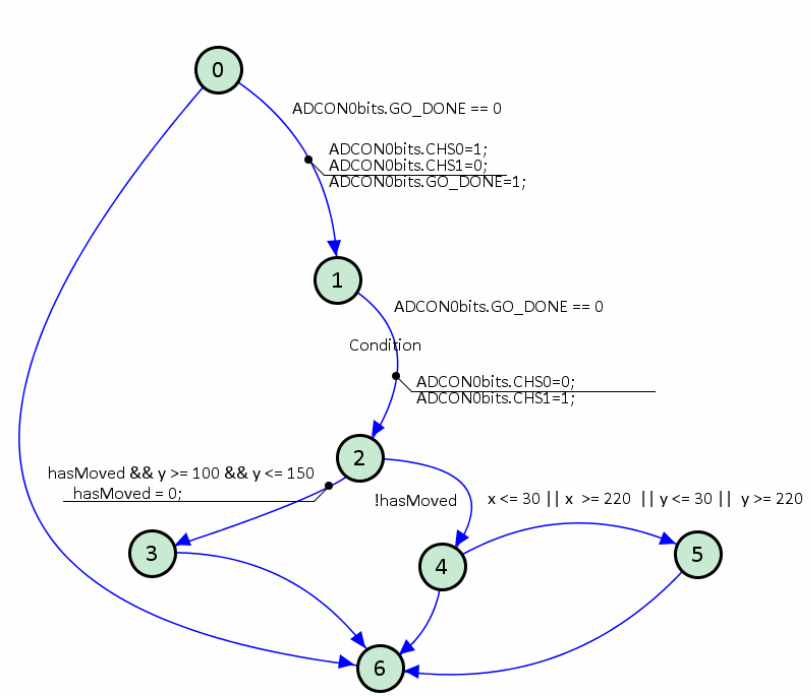


This diagram represents the functioning of the option 2: play recording. The corresponding string is printed in the LCD before printing all the saved recordings (their timestamp and index). Then, a recording is selected using the joystick and '#' key. Afterwards, the index of the selected recording is sent to the Java interface to play it on the central tower (the computer). After the recording is played, 'F' is sent by the interface and received by the microcontroller. This indicates the end of the reproduction. Finally, music is played for 5 seconds in state 85 before returning to the main menu and stopping the music.



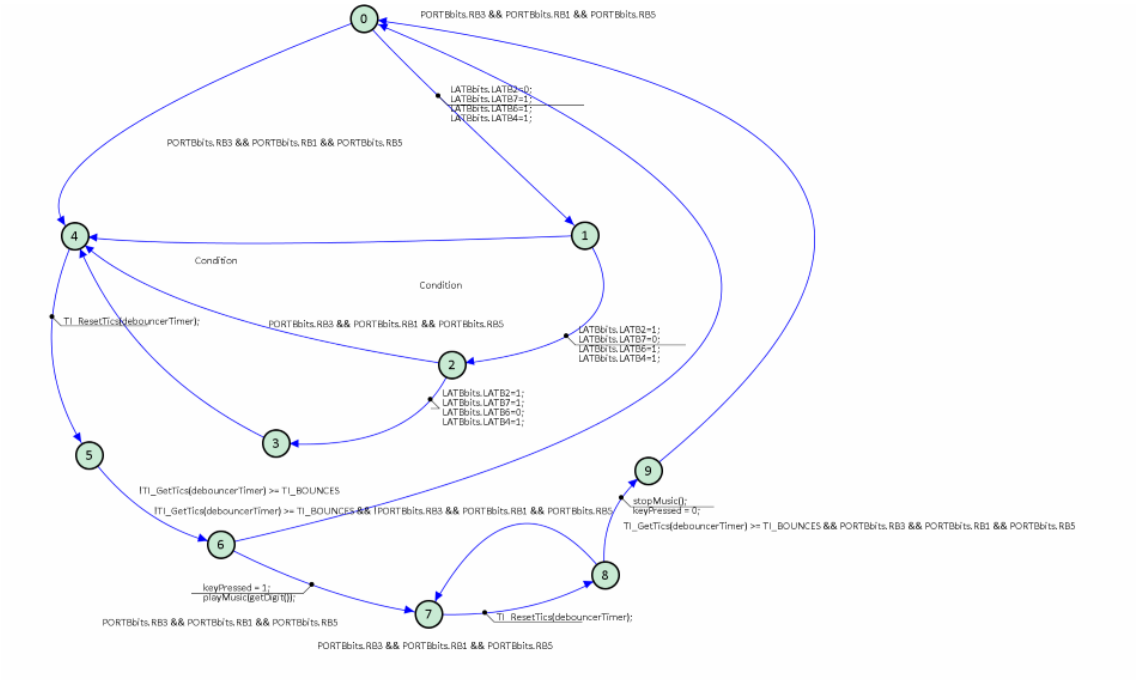
Finally, when the fifth option is selected, the motor accesses state 100. between states 100 and 103, it displays the end communication message: bye bye <TOWER> !. It leaves this message in the LCD for 2 seconds before going back to state 0, asking the user for the port's name, restarting the whole motor.

Joystick



The ADC conversion ADT is responsible for converting analog signals to digital values. It not only handles the conversion process but also manages the output of the conversion. Additionally, it acts as a controller to ensure that the joystick is in motion when it is not centred or when it is pointing in a specific direction.

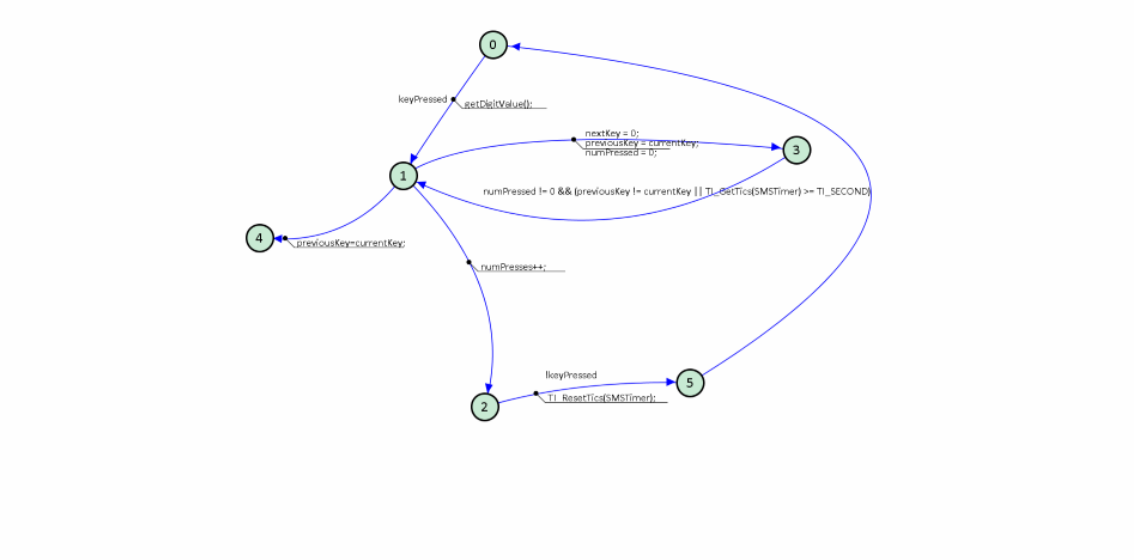
Keypad



The ADT responsible for the keypad motor serves several functions. It controls the detection of pressed keys on the keypad, handles debouncing to prevent erroneous inputs, and manages certain operations such as playing music in specific scenarios. Essentially, it acts as the controller for the keypad system.

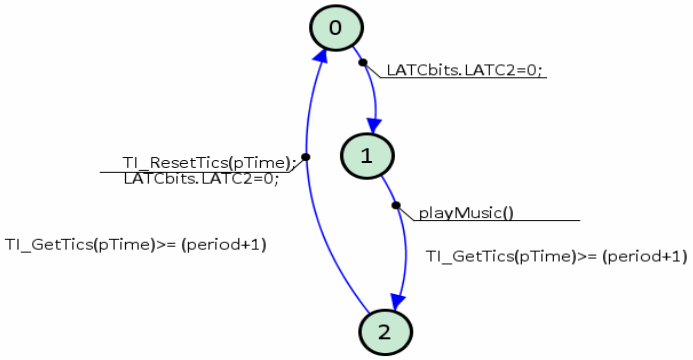
Additionally, there is another motor known as the SMS motor that relies on the keypad motor, although they operate independently.

Sms Keypad



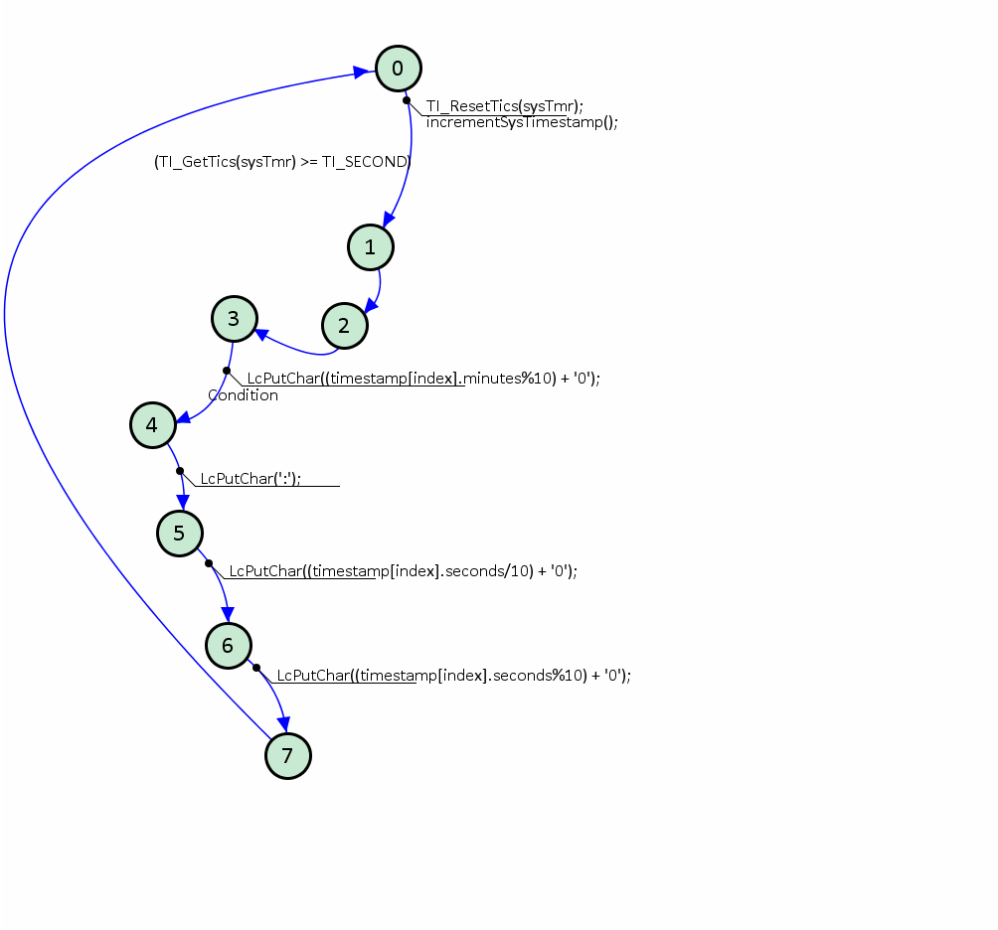
The SMS Keypad is responsible for handling the selection of specific characters on the keypad. It efficiently manages the timing between key presses, facilitates the rotation of characters and digits, and takes care of most of the writing and error handling tasks for the LCD display.

Audio



The audio ADT primarily relies on two functions and a simple core loop, as illustrated in the diagram. Its main purpose is to facilitate audio playback. The playMusic() function activates the audio and plays the specified period, while the stopMusic() function is called after the playMusic() function completes to turn off the audio.

System Timer



Observed Problems

Throughout the practice, we faced several challenges that tested our problem-solving skills. Initially, we struggled to become familiar with the C environment in MPLAB, but with perseverance and experimentation through test projects, we gradually gained a better understanding.

As we proceeded to implement the code for the joystick and keypad, we encountered relatively smooth progress. However, when it came to programming the LCD, we faced a significant setback as it refused to turn on. Despite investing an entire day troubleshooting the issue, we eventually discovered that the problem stemmed from a simple oversight - we had forgotten to connect the potentiometer.

Another major problem happened when attempting to establish a connection with the Java interface using the EUSART. This particular challenge consumed two full days of our efforts. Ultimately, we identified that the enable pin TXEN had not been properly set to 1, leading to the connection issue.

In essence, the time-consuming obstacles we encountered during the practice were primarily due to minor oversights or mistakes.

Conclusions

This practice presented significant challenges, particularly due to the limited time we allowed ourselves for completion. We waited until the final week to submit it, leaving a small margin for error. Despite this time constraint, it proved to be an interesting and valuable learning experience. We gained a deep understanding of memory usage and the importance of managing limited resources effectively. Furthermore, we acquired knowledge on working with various components and establishing connections with external programs. Overall, this practice served as a valuable lesson in time management, resource optimization, and expanding our technical skills.