Embedded Systems and Development  
600085  
Students Individual Report

Student ID: 201602395  
December 2019

Due Date: 13 December 2019

Word Count:

Contents

[Introduction 3](#_Toc27124276)

[Artefacts Produced 3](#_Toc27124277)

[Button Matrix 3](#_Toc27124278)

[Buzzer 4](#_Toc27124279)

[EEPROM 4](#_Toc27124280)

[LCD 4](#_Toc27124281)

[Push Buttons 4](#_Toc27124282)

[Thermometer 4](#_Toc27124283)

[Testing Performed 4](#_Toc27124284)

[Critical Evaluation 6](#_Toc27124285)

# Introduction

The task for this project was to create a GreenHouse Control system using the PIC QL200 development board and other necessary hardware components. Towards the beginning of the project it was realized that to complete this task, a series of drivers would need to be created to interact and use different hardware components available. From this understanding, it was decided to assign each group member with the creation of a set of drivers.

The list of drivers needed are as follows:

* Button Matrix
* Buzzer
* EEPROM
* LCD
* Push Buttons
* RTC
* Thermometer

Once all drivers were created and tested, it was then possible to start putting these components together with the right logic to produce the GreenHouse system requested.

# Artifacts Produced

This section will detail the purpose and features of each driver created by this group member. The following are the allocated drivers to be created and tested:

* Button Matrix
* Buzzer
* EEPROM
* LCD
* Push Buttons
* Thermometer

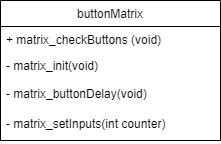
To complete the creation of each of these drivers, it was decided that each driver would be created in its own “prototype” project folder so that it could be experimented with verified for completeness before it was implemented into the main project.

Each driver was created and designed so that a user would have full functionality over a certain piece of hardware while using as little methods from a driver as possible. This also results in a more fluent and easier to understand the system.

### Button Matrix

The Button Matrix driver was created to interact with the 4x4 button matrix using port C. As this matrix has 16 possible outputs with only 8 port bits to interact with, a series of input/output configurations were developed to gain full use out of the matrix. Half of the bits were set to output and half of the bits were set to input. One by one each input bit was set to low and every output bit was read. Using this configuration allowed for a possibility of 16 different combinations of bytes across the C Port. This value is returned from the driver as an integer from 0 – 15.

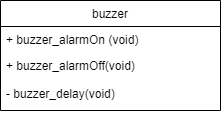
##### Class Diagram



### Buzzer

The buzzer driver was created to interact with the buzzer using bit 0 on Port E. By default the buzzer on this development board using bit 2 on Port C, however, due to the driver Button Matrix driver also using port C, a jumper cable is on the board to redirect this input bit. The buzzer is used to create an alarm sounding noise to indicate to the user a certain parameter has been met, and attention towards the system is needed.

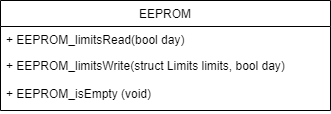
##### Class Diagram



### EEPROM

The EEPROM driver is used to read and write data to and from the EEPROM. The EEPROM is used to save the temperature limits of the system to memory so that they are saved and not reset upon every reset of the system.

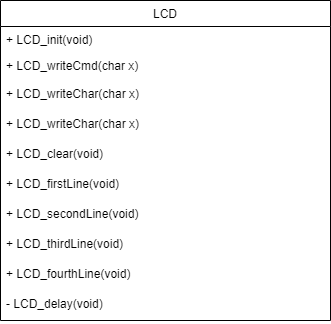
##### Class Diagram



### LCD

The LCD driver is used to display useful system information to the user using port A and Port D. Port A is used to tell the LCD when to be pulling data from Port D. The LCD provides the main source of output for this system allowing the user to navigate through a series of menus and see various information of the system.

##### Class Diagram



### Push Buttons

The Push Buttons driver is used to navigate between the different menus of the system using port B. This driver simply sets port B to output and checks for any signal being low from any of its first 4 bits.

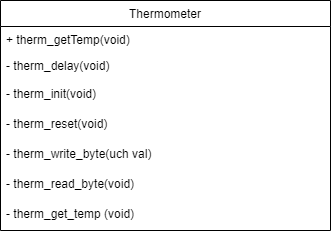
##### Class Diagram



### Thermometer

The Thermometer driver is used to interact with the DS18B20 thermometer using port A. The thermometer provides the main input into the system, the temperature, and is, therefore, being checked and its temperature updated on the screen constantly in the system.

##### Class Diagram



# Testing Performed

Towards the beginning stages of creating the drivers, it was realised that due to the nature of this project of using embeddable hardware, there was no simple way to simply display an output of our system such as using a console or a debugger. A debugger was available that was used to test the logic of the system, however, could not be used to test the functionality of the hardware as it existed purely within the IDE.

Therefore a 7-Segment Display driver was used to print out a maximum of 6 hexadecimal characters to the 7-segment display on the development board which was used to test the remaining drivers. Eventually, the LCD driver was created, and this would replace the 7-Segment Display driver as it provided more detailed outputs and conflicted with the pin usage as both drivers use port C.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Driver Functionality Testing** | **Test Description** | **Expected Outcome** | **Actual Outcome** |
| 1 | Button Matrix. Pressing of the buttons | The button matrix driver returns an integer 0-15 depending on the button. Each number was pressed, and its number printed to the 7-segment display. | With each button pressed, a different number between 0 and 15 would display. | As expected. |
| 2 | Buzzer. Activation of the buzzer. | A loop was created that turned the buzzer on, conducted a long delay, turned the buzzer off and conducted another long delay. | The buzzer should beep for a period, then turn off for the same period. | As expected. |
| 3 | LCD. Displaying a series of 4 characters | A series of 4 characters were written to the display. The system would then enter an endless loop to prevent a reset. | The LCD should display the given series of characters in order horizontally across the display | As expected. |
| 4 | LCD. Displaying characters on different lines. | The same series of characters were written to the display but instead, one character was written on each line. | The LCD should display the given series of characters in order vertically down the display. | As expected. |
| 5 | LCD. Clear command | Test 3 was repeated; a long delay was conducted then the clear command was issued. | The LCD should display a series of characters, then the LCD should completely clear itself. | As expected. |
|  | *NOTE. With the creation of the LCD driver, it will be used in the following tests as a source of output.*  *Additionally. EEPROM by default has all bits set to 1. Therefore, all bytes read as FF.* | | | |
| 6 | EEPROM. Reading from and writing to memory | The first 8 bytes of EEPROM was read. The LCD was set to display the contents read.  A series of 8 characters were written to EEPROM. The board was then reset. | The LCD should display 8 FF’s.  Upon reset of the board, the LCD should display the 8 characters written in the write sequence. | As expected. |
| 7 | Push Buttons. Pressing of the buttons | The push buttons driver returns an integer 0-3 depending on the button. Each button was pressed, and its number printed to the LCD | With each button pressed in order, the numbers 0 to 3 appeared on the LCD. | As expected. |
| 8 | Thermometer. Reading the temperature | The temperature of the room was recorded and the temperature of our thermometer was displayed on the LCD. A user’s finger and thumb were placed around the thermometer to increase its temperature. | The temperature displayed on the LCD should resemble the room temperature with a margin for error. The temperature should be seen to rise once heated by the user. | As expected. |

# Critical Evaluation

Using the PIC 16F87 and each hardware component datasheets a finished Green House solution was created. Although long, an appreciation has grown towards these datasheets as the student's ability to read them and understand them has greatly increased.

The process of creating each API first before implementing the logic of the system allowed for an easier flow of work and delegation of tasks. Through the use of modular testing, it was a straight forward process to combine each element together to create a finished system. Additionally, it provided an easy method of maintaining the system and fixing any errors that arose.

If this project was completed again, a greater focus on the fairness of delegation of tasks would be conducted as this project was somewhat hindered by the outweighed workload separation between group members.