School of Electronic &

Electrical Engineering

FACULTY OF ENGINEERING

ELEC 3662 –– Embedded  
 Systems

Project report  
ELEC3662\_PROJECT\_7\_2

|  |  |
| --- | --- |
| **Name**: Tariq Alqawasmi | **SID**: 201059872 |
| **Name**: Angus Moore | **SID**: 201038110 |

Contents

[Introduction 3](#_Toc534988415)

[Aims and objectives 4](#_Toc534988416)

[Hardware implementation 5](#_Toc534988417)

[Software implementation 5](#_Toc534988418)

[Results 9](#_Toc534988419)

[Testing 10](#_Toc534988420)

[Conclusion 10](#_Toc534988421)

# Introduction

In this module, we developed skills relevant to the area of embedded systems in terms of software and hardware. We learned how to program a microcontroller using the C language, interacting with its on board peripherals as well as interfacing with external components.

The project we worked on tasked us with developing a calculator that could correctly interface with a 4x4 keypad matrix and a 16x2 HD44780 LCD screen to perform floating point calculations with correct operator precedence.

The microcontroller we worked with is the TM4C123GH6PM microcontroller. A microcontroller is a small computer contained in a single chip. At the heart of it, is a microprocessor and memory busses such as Random-Access Memory and Read-Only memory (RAM/ROM). It also contains a range of input/output pins (I/O Ports) that enable it to interface to the outside world. In our case, those pins allowed us to interface with the keypad and the LCD screen.

# Aims and objectives

The aim for the project was to develop a TM4C123GH6PM program that could take a floating-point infix expression, inputted via the keypad, evaluate it with compliance to BODMAS and output the correct answer to the LCD screen.

The work approach we agreed on was to correctly interface the keypad and the LCD first, then code a working calculator that performs basic operations. And finally if time allows that, think of extras that could be implemented.

# Hardware implementation

In this part, we will explain the hardware behind this successful project.

As we mentioned above, the microcontroller contains I/O Ports that allowed us to interface with the keypad and the LCD.

The keypad is a 4x4 matrix, we assumed that the rows are set as inputs on port E [0:3], and the columns are set as output on port D [0:3]. They occupy 8 pins on the breadboard and are arranged as shown in the table below.

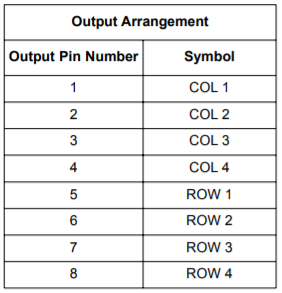


Table 1- Keypad output arrangement

The LCD runs on 5V and needs a potentiometer connected to the contrast pin to be able to control the brightness of the screen. The LCD is in 4-bit mode. This means only 4 pins of the microcontroller are used and not 8. Those are on port B [0:3] and are connected to the data bits DB [4:7] of the LCD. Also, other pins like digital enable and register select are connected to port A 2 and 3 respectively. The complete pin assignment is explained in the following table

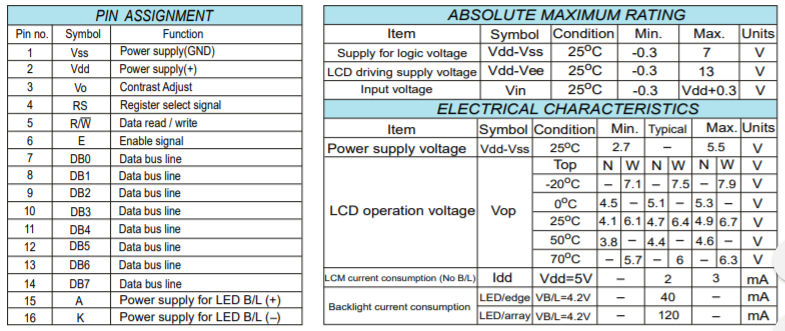
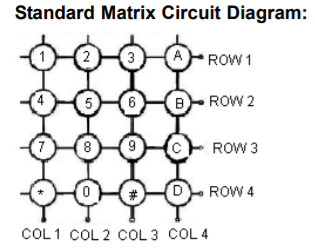
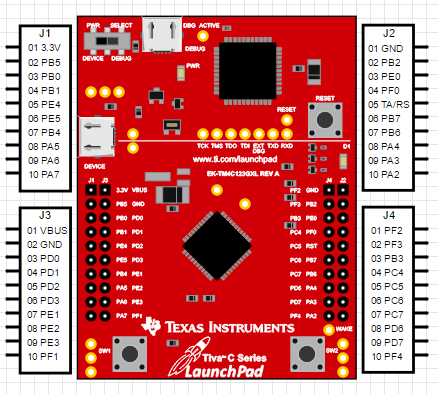
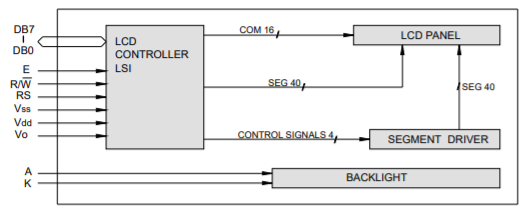


Table 2- LCD pin assignment

Finally, this diagram represents the connections of the I/O ports used to connect the LCD and keypad





Pot.

# Software implementation

We structured our program in an object-oriented manner, having separate ‘.c’ and ‘.h’ files for each section of the program. This keeps the code orderly as well as allowing for easier debugging and future updates.

## Keypad and LCD

To get the keypad working it first needs to be initialised, by setting the appropriate registers we make the rows inputs and columns outputs. Then, we implemented a function called decodeKeyPress() to read values from the keypad. This checks for which column and row the high input intersects. Despite the method being straight-forward, the drawback of this method is that the code submitted was long.

Knowing what key has been pressed we can assign it to a character, we mapped the keypad as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| 1  ( | 2  ) | 3   ^ | +   \* |
| 4 | 5 | 6 | -   / |
| 7 | 8 | 9 | S |
| C | 0 | . | = |

Table 3- Keypad buttons

Since there are only 16 buttons available on the keypad and we require all 9 digits as well as parenthesis, addition, subtraction, multiplication, division and a power symbol, we implemented a shift button (S). This sets a flag in the calculate() function and if this flag is set to 1 and a dedicated button with an alternative character is pressed (as pictured in the above table), then the alternative character is entered into the infix string, displayed on the LCD and the flag is set back to 0.

On the other hand, the LCD also needed to be initialized. As indicated in the datasheet p.45, we first need to create a delay of at least 15 ms, then we need to set the pin connected to RS to 0 to select the instruction register and to be able to write data to the LCD. We then need to write 0011 to DB7-DB4 on the LCD. As we have connected these to PB0 - PB3 on the TM4C123 microcontroller, we need to write 0x3 to LCD\_DB. Once we have put the instruction on the port, we need to latch the command by pulsing the EN line with a pulse width of at least 450 ns. This is done using the LCD\_ENPulse() function which set the EN line high for this required width and then returns the EN line to 0. Finally, we need to send 0x3 another two times.

Then we implemented a set-up instruction to configure the LCD to behave as we wish such as set the cursor or turn the display on. The instruction is called LCD\_WriteCommand(unsigned char command). It is a byte long and require sending the high nibble first then the low nibble. Similarly, since we are writing data, we need to set RS to 0 and wait for 40 us. Regarding the first nibble, it needs to be written to LCD\_DB then shift those 4 bits and send the second nibble.

Additionally, we created LCD functions to assist us in controlling the LCD screen such as LCD\_ClearScreen(void) and LCD\_GoTo (int Row, int Col).

Finally, one of the most important function is the LCD\_WriteData(), because up until now all the other functions were created to send commands to the LCD whereas this function allows to send data to be displayed on the LCD. Since data is to be sent, we need to make RS 1 instead of 0 and wait a delay of 200us. This function is accompanied by LCD\_WriteRamString() which prints an array of characters on the LCD.

## Calculations

Our calculation code takes the input characters from the keypad and adds them to an infix expression string, this is done by placing the characters into a stack, in C there is no standard library functions for stacks, so we have followed common stack implementation code to create arrays that can be popped, pushed and read from. Infix notation is the most human readable and commonly used form of writing arithmetic expressions, an infix expression, however, is difficult to calculate programmatically as we need to account for operator precedence and parenthesis. Reverse polish notation (RPN), otherwise known as postfix, is an alternative way of writing an arithmetic expression. In RPN the operators follow their operands, it also eliminates the needs for parenthesis and proves simpler to evaluate programmatically through stacks. So, our methodology for evaluating an infix expression is to first convert it to RPN then evaluate this new, simpler to calculate expression.

This is done via the Shunting Yard algorithm; this algorithm is commonly used for converting an infix string to reverse polish notation whilst retaining the correct operator precedence and discarding any parenthesis. It does this by iterating through the input string and moving operators and parenthesis into an operator stack depending on their precedence and left/right associativity, operands are pushed to output stack and operators are moved to output queue accordingly, either when parenthesis dictate (these are then discarded from the output stack) or the whole input string has been iterated through. Following this process gives us an output RPN string, though the issue is that when inputting floating point or multi-digit numbers the resulting RPN string is ambiguous as to which numbers were entered, for example an input string of **23+4.2** would result in an output string of **234.2+**, as you can see this is ambiguous as to what the original operands were. To solve this, we add spaces at appropriate points in the algorithm so that every operand and operator in the resulting RPN string has spaces between them, now an input string of **23+4.2** would become **23 4.2 +**.

Now when we come to evaluate the RPN string we know what operands to perform the operator calculations on.

To evaluate the RPN string we iterate through the expression and if the current token is a digit or decimal place, we push it to a double conversion stack. This is necessary for performing the correct calculation as this RPN string is made of character data. When the current token is a space and the previous token is not an operator, we know that an operand string is now in the conversion stack. Using the atof() function we then convert this character string to the represented double value; the conversion stack is then cleared and this value is pushed to an evaluation stack. When the current token is an operator, we perform that operation on the top two values of the evaluation stack. This process results in the correct evaluation of the RPN string, then when the equals button is pressed, we print the result to the bottom row of the LCD via the afore mentioned LCD\_WriteRamString() function. Pressing the clear button will clear the LCD and all stacks ready for a new calculation.

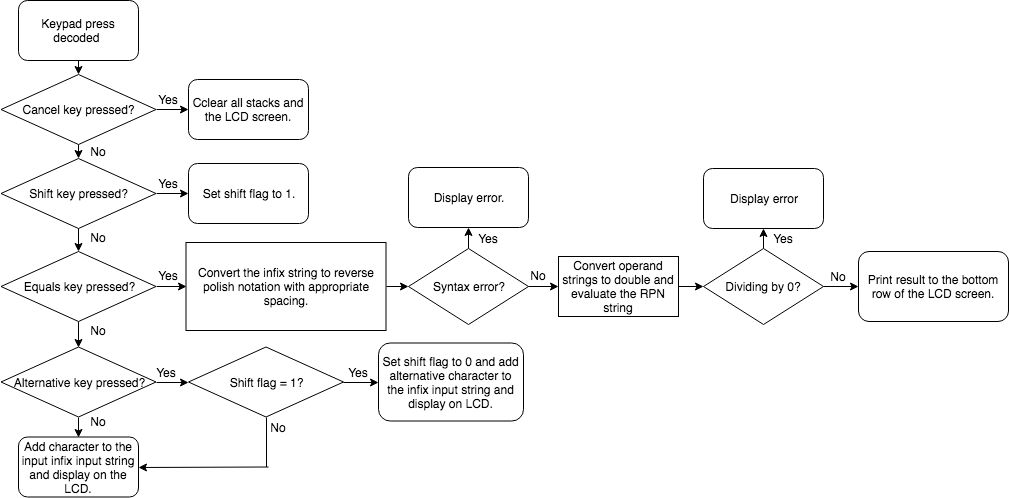
Below is a flow chart describing the software process:

Figure 1- Flow chart of the code software

# Results



At the end, our project was successful as we managed to get a working calculator that does basic operations with multiples operands:

 Also, we managed to add extra features like powers and parenthesis priorities using the shift button:



# Testing

To test the evaluation of the infix string we created a testing function that would input different infix strings with a known correct answer and compare what the programs output was to this. There are also a number of printf() errors which also print to the LCD throughout the calculation process that indicate any syntax errors in the infix string, if there are any stack overflows or divisions by 0. We also tested the calculator by comparing it to the same infix expressions calculated on a Casio fx-83GT.

# Conclusion

In conclusion we feel this project was a success, we developed a calculator for the TM4C123GH6PM microcontroller that could correctly evaluate any infix string with the presented operators, it took an input from the keypad and correctly displayed the relevant information on the LCD. There is however, room for improvement with this calculator, we did not manage to implement additional features and the calculator could perhaps be extended, for example to work with trigonometric functions. However, we feel the project met the specifications and executed the presented task in a logical and informed manner.

Notes:

The tables showing the pin arrangement of the lcd and the keypad as well as their circuit, was taken from their datasheets.

Tiva diagram was taken from:

<https://www.google.com/search?q=tiva+tm4c123g&safe=strict&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjUmb2QmO7fAhXTUBUIHTjqDiAQ_AUIDigB&biw=1536&bih=754&dpr=1.25#imgrc=ZvAkhpX-se1nZM>: