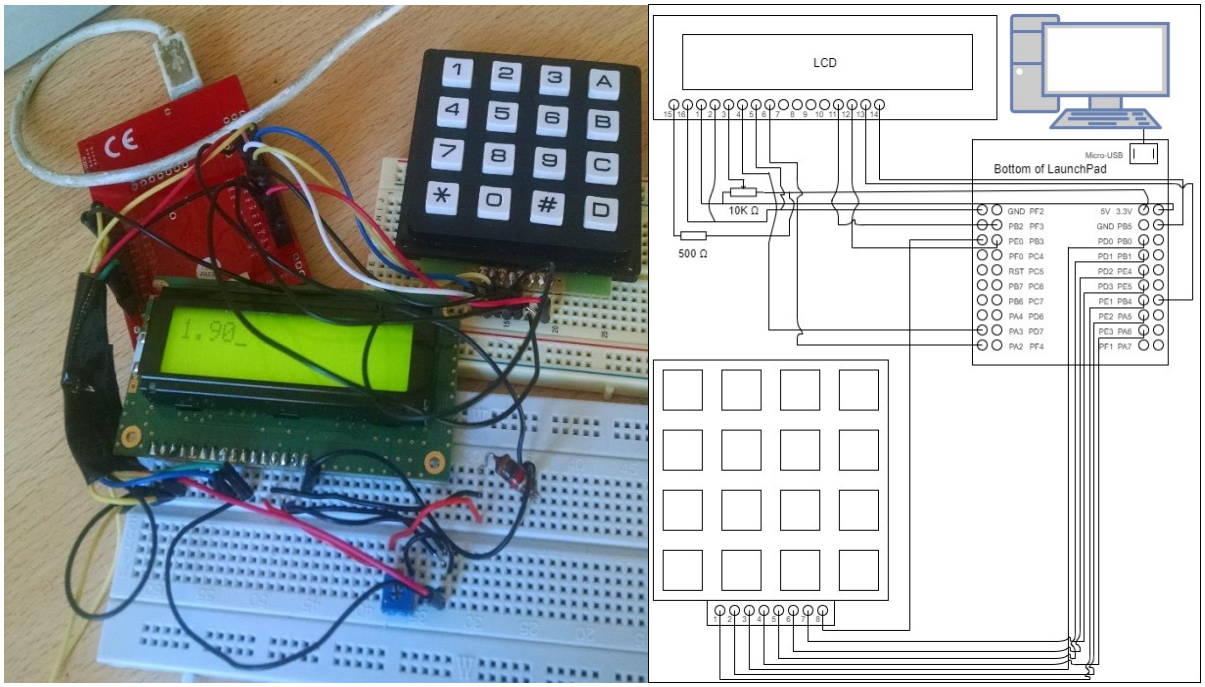
UNIVERSITY OF LEEDS

Calculator

An ELEC3662 Embedded Systems Project

Group Project Report



Project and report authors

Mateo Gomez-Randulfe 200970511

Konstantin Dragostinov 200978822

Group 10

Leeds, December 2017

1633 words

Contents

Introduction

Project Aims and Objectives

The aim of this project is to understand and interface a LCD (16x2 HD44780) and Keypad (4x4 AK-1607) with a microcontroller (Tiva TM4C123GH6PM) to program a calculator (using Keil uVision as the software for programming).

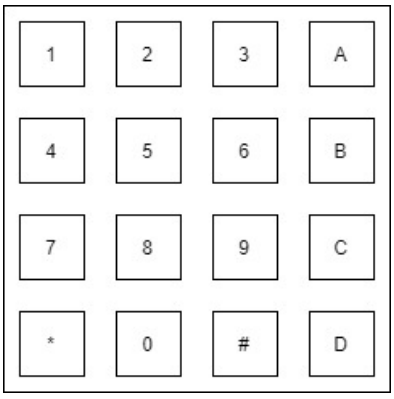
The following compulsory and optional objectives were set (C for compulsory and O for optional as extra feature):

Objectives
Interface Keypad Correctly **C**
Interface LCD correctly **C**
Display Results of Keypad input buttons in the LCD **C**
Calculator with simple functions (*, -, +, /) **C**
Floating point calculations **C**
Nested calculations (up to six operands) **C**
Password for accessing calculator functions **O**
Alternative functions (up to ten) **O**
Interfacing cursor shifting functions for avoiding to “write” out of the LCD **O**

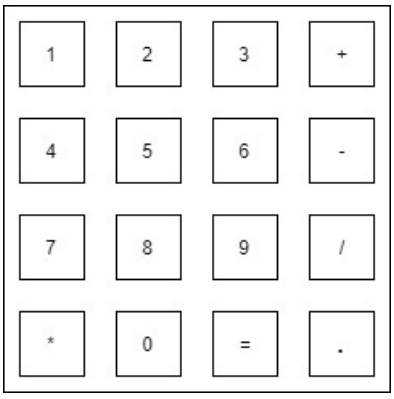
Table 1 Project objectives and which were completed

Once the calculator is turned on, a message asking for the password previously set was displayed. Until the user do not get the password right, he will not be able to access the calculator. For security, a star was displayed every time the user was inputting any number for the password.

Due to the small number of buttons of the Keypad, alternative functions were applied by using Shifts similar like the ones used in actual calculators. The main use of the buttons in the keypad was:

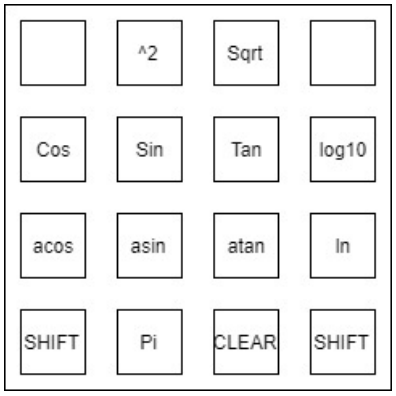


Original button labels



What they do

However, if “*” and then “.” were pressed the alternate functions were enabled (it is necessary to press two keys instead of one to be able to use the numbers 0-9 as for being able to use as many alternate functions as possible). Once alternate was activated, the buttons corresponding the functions were:



QUITAR SHIFT Y POWER

It is important to point out that for using these functions the desired number should have been previously introduced (for instance if the operation is cosine of 1, first it is required to press 1, then “*”, “.” and “4”).

Hardware Implementation

Following the datasheet instructions given by [1-3], the calculator was constructed as summarised by Tables 2 and 3 and Figures 3 and 4. The pins are denoted as: port, port name, pin number. For example, PA2 is the second pin of port A.

LCD-Tiva Interface			
Pin on LCD	Symbol	Pin on TM4C123GH6PM	Function
1	V_{ss}	GND	Common ground
2	V_{dd}	3.3V	(+) Power for the LCD
Pin 3 was connected to a 10 KΩ potentiometer, the latter being wired to 5V and GND on the Tiva.			Contrast adjust
4	RS	PA3	Register select signal
5	R/W	GND	As set always to ground the MCU will be always writing and never reading from the LCD
6	E	PA2	Enable signal
7-10	DB0-4	Unpinned	Data bus lines
11-14	DB4-7	PB2-PB5	
15	A	5V	(+) Power for the backlights
16	K	GND	Common ground

Table 2 LCD-Tiva pin correspondence

The LCD uses ports A and B. As the LCD was operated in 4-bit mode, 4 bits (nibble) were processed or sent at a time so only 4 pins were used for the data pins (from LCDB4-7

corresponding to PB2-5). For limiting current to the backlight and the LCD, a 500 Ω resistor was inserted between “A” and the “5V”. This avoids overheating. The potentiometer increased or decreased the intensity of the characters of the LCD.

The keypad used ports D as columns (outputs) and E as rows (inputs) which had activated their internal pull-down resistor.

Keypad-Tiva interface		
Pin on Keypad	Function and Symbol	Pin on TM4C123GH6PM
1	COL1	PD0
2	COL2	PD1
3	COL3	PD2
4	COL4	PD3
5	ROW1	PE0
6	ROW2	PE1
7	ROW3	PE2
8	ROW4	PE3

Table 3 Keypad-Tiva pin correspondence

The final disposition of the hardware would be like the following schematic:

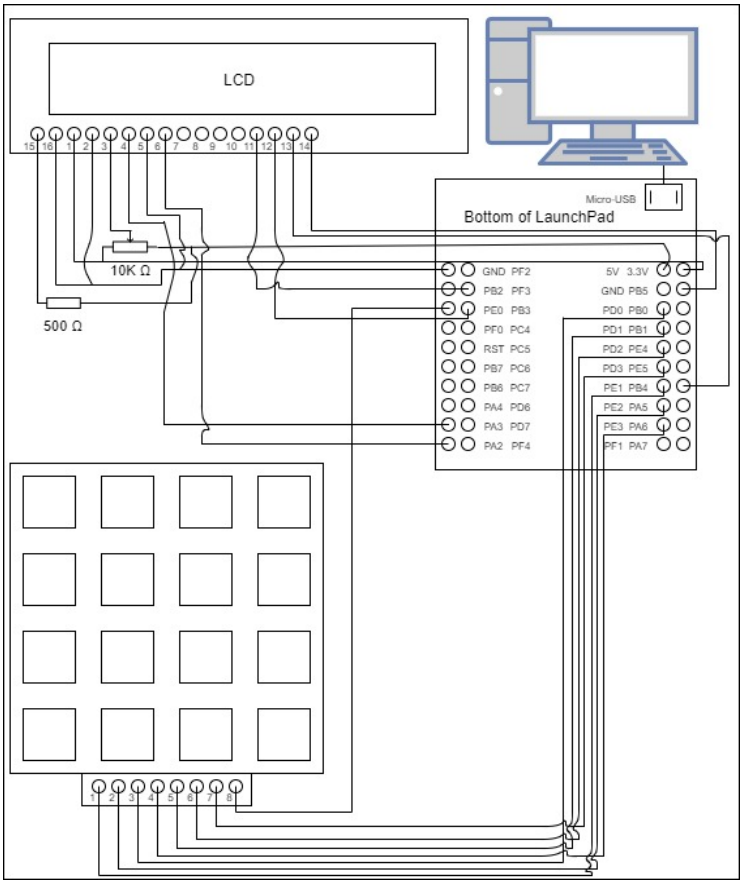


Figure 3 Circuit diagram of the calculator system. Diagram was made using draw.io

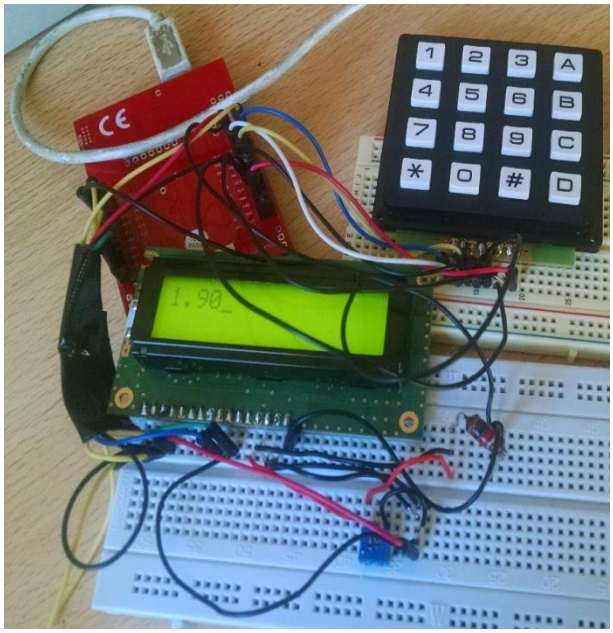


Figure 4 Photo of the system breadboard circuit

Software Implementation

Initialisation

The initialisation part spanned across all files of the program. The initialisation functions were grouped together into one function “Init()”:

```
void Init(void){
    PortInit();
    lcdinit(); lcdcommandInit();
    keypadInit();
}
```

Figure 5 Principal initialisation function

To use the pins for the LCD-Tiva-Keypad interface, the Tiva ports needed to be initialised. This was fulfilled using PortInit() and inside keypadInit(), which were very similar but the first one initialised the ports used by the LCD, whilst the other – the keypad ports. Note the steps taken to initialise each port below:

```
void keypadInit(void){
  unsigned long delay;
  //Init for port D (output,columns) 0-3
  SYSCTL_RCGC2_R |= 0x00000008;     // 1) D clock
  delay = SYSCTL_RCGC2_R;           // delay
  GPIO_PORTD_LOCK_R = 0x4C4F434B;   // 2) unlock PortD
  GPIO_PORTD_CR_R |= 0x0F;           // allow changes to PD3-0
  GPIO_PORTD_AMSEL_R &= 0x00;        // 3) disable analog function
  GPIO_PORTD_PCTL_R &= 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTD_DIR_R |= 0x0F;          // 5.2) PD3-0 Outputs
  GPIO_PORTD_AFSEL_R &= 0x00;        // 6) no alternate function
  GPIO_PORTD_DEN_R |= 0x0F;          // 7) enable digital pins PD3-0
  //Init for port E (input,rows) 0-3
  SYSCTL_RCGC2_R |= 0x00000010;     // 1) E clock;
  delay = SYSCTL_RCGC2_R;           // delay
  GPIO_PORTE_LOCK_R = 0x4C4F434B;   // 2) unlock PortE
  GPIO_PORTE_CR_R |= 0x0F;           // allow changes to PE3-0
  GPIO_PORTE_AMSEL_R &= 0x00;        // 3) disable analog function
  GPIO_PORTE_PCTL_R &= 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTE_DIR_R &= ~0x0F;          // 5.1) PE3-0 Inputs
  GPIO_PORTE_AFSEL_R &= 0x00;       // 6) no alternate function
  GPIO_PORTE_PDR_R |= 0x0F;         // enable pulldown resistors on PFE3-0
  GPIO_PORTE_DEN_R |= 0x0F;         // 7) enable digital pins PE3-0
}
```

```
void PortInit(void){
  unsigned long delay;
  //Init for PortB
  SYSCTL_RCGC2_R |= 0x00000002;     // 1) b clock
  delay = SYSCTL_RCGC2_R;           // delay
  GPIO_PORTB_CR_R |= 0x3C;           // allow changes to PB2-PB5
  GPIO_PORTB_AMSEL_R &= 0x00;        // 3) disable analog function
  GPIO_PORTB_PCTL_R &= 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTB_DIR_R |= 0x3C;          // 5.2) PB2-PB5 as OUTPUTS
  GPIO_PORTB_AFSEL_R &= 0x00;        // 6) no alternate function
  GPIO_PORTB_DEN_R |= 0x3C;          // 7) enable digital pins to PB2-PB5

  //Init for PortA2-3
  SYSCTL_RCGC2_R |= 0x00000001;     // 1) A clock
  delay = SYSCTL_RCGC2_R;           // delay
  GPIO_PORTA_CR_R |= 0x0C;           // allow changes to PA2-3
  GPIO_PORTA_AMSEL_R &= 0x00;        // 3) disable analog function
  GPIO_PORTA_PCTL_R &= 0x00000000;   // 4) GPIO clear bit PCTL
  GPIO_PORTA_DIR_R |= 0x0C;          // 5.1) PA2-3 OUTPUTS
  GPIO_PORTA_AFSEL_R &= 0x00;        // 6) no alternate function
  GPIO_PORTA_DEN_R |= 0x0C;          // 7) enable digital pins PA2-3
}
```

Figure 6 Initialising the ports

The pin mapping between the system components, as outlined in “Hardware implementation”, needed to be defined as well. Below the pins for the Keypad-Tiva interface were assigned:

```
//Define all the PINS in PortD individually
#define PORTD0      (*((volatile unsigned long *)0x40007004))
#define PORTD1      (*((volatile unsigned long *)0x40007008))
#define PORTD2      (*((volatile unsigned long *)0x40007010))
#define PORTD3      (*((volatile unsigned long *)0x40007020))
//Define all the PINS in PortE individually
#define PORTE0      (*((volatile unsigned long *)0x40024004))
#define PORTE1      (*((volatile unsigned long *)0x40024008))
#define PORTE2      (*((volatile unsigned long *)0x40024010))
#define PORTE3      (*((volatile unsigned long *)0x40024020))
```

Figure 7 Defining the pins individually, where 4000/4002 are the port addresses and the sum of the last three numbers define which pins are to be accessed

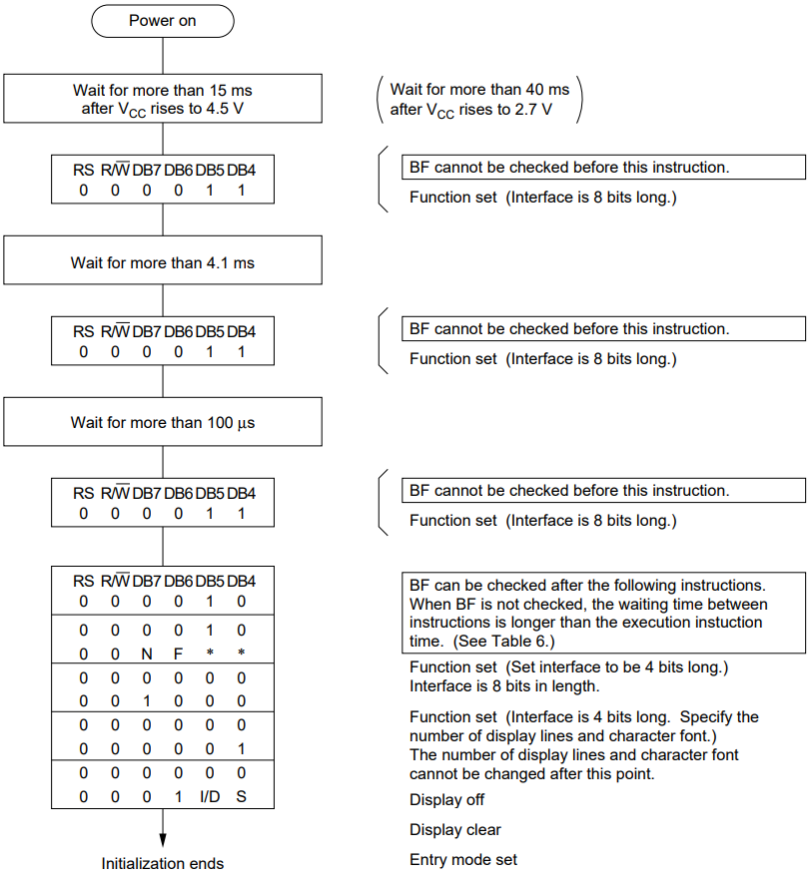


Figure 8 LCD initialisation algorithm, as specified by [4], page 46

Following the steps in Figure 8, the initialisation for a 4-bit interface was performed using the functions lcdinit() and lcdcommandInit(), which wrote commands to the pins (DB4-7). After each such write, the EN line needed to be pulsed for 450ns. Pulsing means setting a line high from an initial low for some period, then returning to low. The first function “wakes up” the LCD and the second one applies desired settings (see second line of Figure 9) and this is how the initialisation finishes. Since commands are a byte long, they need to be sent in two parts, where first the high nibble is written to lcdDB by using bit shifting and masking (for example, 0x2 (<<2)) and then the low nibble Is sent similarly.

```
void lcdcommandInit(void){
  //We need 4-bit data, 2 lines, 5x8 font
  lcdWriteCommand(0x28); SysTick_Wait(delay50ms); lcdWriteCommand(0x00); //sets lcd to 4 bit data, 2 lines, 5x8 font
  lcdWriteCommand(0x06); SysTick_Wait(delay50ms); lcdWriteCommand(0x00); //entry mode: cursor to increment and display not to shift
  lcdWriteCommand(0x0F); SysTick_Wait(delay50ms); lcdWriteCommand(0x00); //sets display pn, cursor on, blink on
  lcdWriteCommand(0x04); SysTick_Wait(delay50ms); lcdWriteCommand(0x00); //entry mode: cursor to increment and display not to shift
}
```

Figure 9 Modifying the LCD settings as per line 2

Time management

Phase-locked loop is an oscillator matching the operating frequency in the Tiva with the frequency of inputs. In the project, System Timer (SysTick) was a counter used to make the Tiva "sleep" for some time. The two circuits were initialised (PLL_Init() and SysTick_Init()) and SysTick_Wait() was called to set delays where appropriate with an argument defining the delay. Since a clock frequency f of 80 MHz was used, the period T_{clock} was:

$$T_{clock} = \frac{1}{f} = \frac{1}{80x10^6} = 13x10^{-9}\ ns\ (1)$$

Generating an time delay T_D of 200us using SysTick_Wait() would be calculated by dividing an appropriate value by 13 ns. If we want a delay of 200 us, then the delay argument of SysTick will be 16000, as calculated by:

$$delay = \frac{T_D}{T_{clock}} = \frac{200x10^{-6}}{13x10^{-9}} = 15384.61 \approx 16000\ (2)$$

Calculations

The sequence for reading rows and columns described in "Hardware implementation" was fulfilled in a function readKeypad(), whose algorithm looks like:

```
PORTD0=0x00;PORTD1=0x02;PORTD2=0x00;PORTD3=0x00; //Read 2nd column
if(PORTE0==0x01){return '2';} //Read 1st row
else if(PORTE1==0x02){return '5';} //Read 2nd row
else if(PORTE2==0x04){return '8';} //Read 3rd row
else if (PORTE3==0x08){return '0';} //Read 4th row
```

Figure 10 Checking row inputs at the second column

In the example above example, the second column is read. If the user pressed a key to one of the rows, they changed the hex value written to the row. The function checked whether a row matched that hex value. If it did, the corresponding key was returned.

The functions for processing the input number and the password input were almost identical, in that they read the input (readKeypad()) as a string and returning it as a char or int.

```
char returnNumber(char a, char b, char c){
    //takes a reading from the keypad in a,
    //concatenates the reading to b and copies it to c
    if(a == '0') {strcat(&b,"0");strcpy(&c,&b);}
    if(a == '1') {strcat(&b,"1");strcpy(&c,&b);}
    if(a == '2') {strcat(&b,"2");strcpy(&c,&b);}
    if(a == '3') {strcat(&b,"3");strcpy(&c,&b);}
    if(a == '4') {strcat(&b,"4");strcpy(&c,&b);}
    if(a == '5') {strcat(&b,"5");strcpy(&c,&b);}
    if(a == '6') {strcat(&b,"6");strcpy(&c,&b);}
    if(a == '7') {strcat(&b,"7");strcpy(&c,&b);}
    if(a == '8') {strcat(&b,"8");strcpy(&c,&b);}
    if(a == '9') {strcat(&b,"9");strcpy(&c,&b);}
    if(a == '.') {strcat(&b,".");strcpy(&c,&b);}
    return b;
}
```

```
int password(char input) {
    //concatenates the inputs of readKeypad() into a string,
    //which is converted to int before it's returned
    if(input == '0') {strcat(passEntry,"0");strcpy(passQuery,passEntry);}
    if(input == '1') {strcat(passEntry,"1");strcpy(passQuery,passEntry);}
    if(input == '2') {strcat(passEntry,"2");strcpy(passQuery,passEntry);}
    if(input == '3') {strcat(passEntry,"3");strcpy(passQuery,passEntry);}
    if(input == '4') {strcat(passEntry,"4");strcpy(passQuery,passEntry);}
    if(input == '5') {strcat(passEntry,"5");strcpy(passQuery,passEntry);}
    if(input == '6') {strcat(passEntry,"6");strcpy(passQuery,passEntry);}
    if(input == '7') {strcat(passEntry,"7");strcpy(passQuery,passEntry);}
    if(input == '8') {strcat(passEntry,"8");strcpy(passQuery,passEntry);}
    if(input == '9') {strcat(passEntry,"9");strcpy(passQuery,passEntry);}
    if(input == '.') {strcat(passEntry,".");strcpy(passQuery,passEntry);}
```

Figure 11 Processing the inputs

The password() function then constantly compared the input password with the actual one.

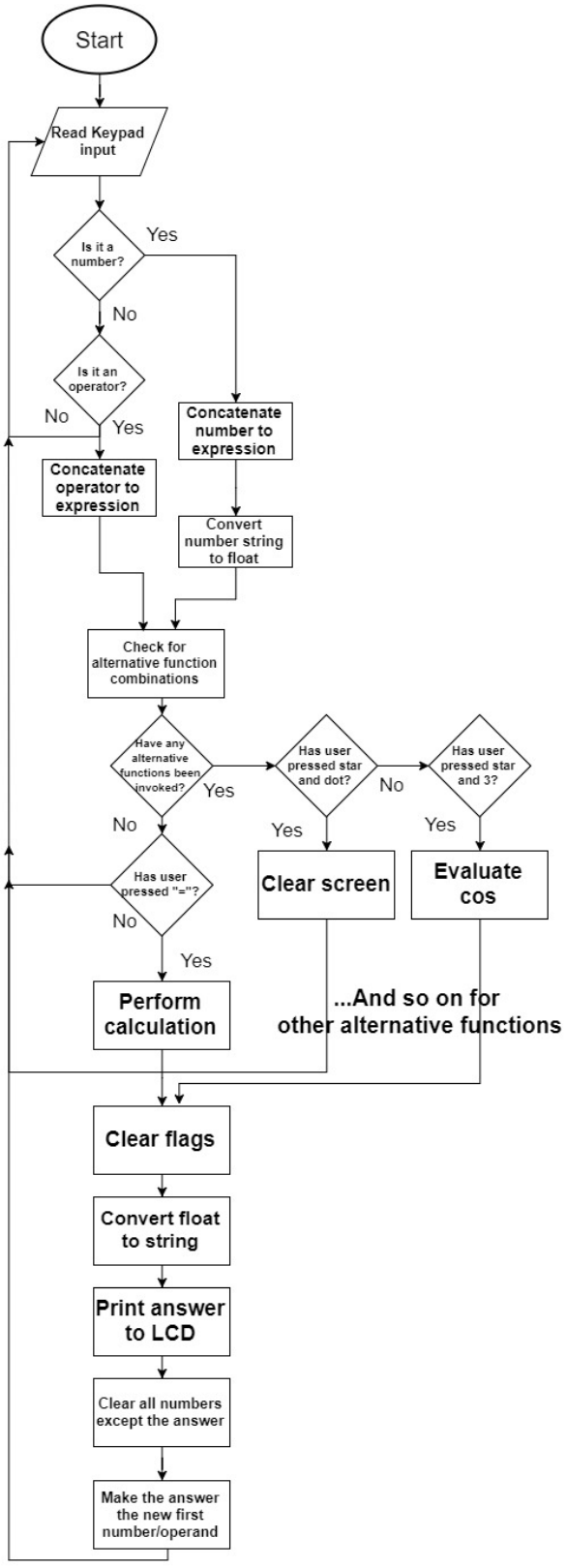


Figure 12 Calculator algorithm. Flowchart was made using draw.io

It was made so that pressing each key would result in a flag raise corresponding to that key. Conversion from a number to a string was done using the function itoa(). The inverse was performed using sprintf(). The final numbers were reset using memset() since they were stored in arrays and the flags were reset by equalling them to 0. Operators {+, -, *, /} were

made to correspond to {1, …, 4}. Every time the user input an operator, the array “pos[]” index was incremented and the number at that index equalled to the corresponding operator (e.g. pressing “-“ would set pos[0]=1 and then pos[0] would be incremented to pos[1]). Later, a check was made at the zeroth index of pos[] which operator had been chosen:

```
if(pos[0]==1){y=firstNumber+secondNumber;}
if(pos[0]==2){y=firstNumber-secondNumber;}
if(pos[0]==3){y=firstNumber*secondNumber;}
if(pos[0]==4){y=firstNumber/secondNumber;}
```

Figure 13 Checking for operators

“y” was then transferred in another similar operator check but for pos[1] and the cycle was repeated until “=” was pressed or the array indices were exhausted (up to pos[5]).

Workflow, Testing, Results

Whilst both project authors worked individually at times, most of the code was written while working as a pair. Problems were not explicitly set for each member, but they would research tasks in terms of algorithms and implementations and then would frequently meet to program together and figure out a solution. What prevented the project from working successfully at times were either poor algorithms or poor implementation (in code) of such (e.g., figuring out how to concatenate “1” and “2” into “12”).

For testing, each main and alternative function were tried, and they worked as intended in terms of expected result. The calculator could not handle division by zero, as it displayed random numbers as the answer. Instead of displaying zero (because of 4-4, for example), the screen cleared instead of displaying “0”.

The cursor shift would go six places forward, but not back and was failed to be implemented properly due to:

- “When 4-bit length is selected, data must be sent or received twice” – which was failed to be considered [4] for the cursor implementation, although done for the LCD initialisation, as shown in Figure 14.
- Since R/W was always connected to GND and therefore, = 0, the current cursor address could not be extracted (using “Read busy-flag and address counter” instruction).

```
void lcdInit(void) {
  PLL_Init(); SysTick_Init(); //init time functions
  lcdEN =0x00<<2; //set enable to 0
  lcdDB =0x00<<2; //set data to 0
  SysTick_Wait(delay50ms); //50 ms wait

  lcdRS=0x00<<2; //RS to 0

  lcdDB =0x3<<2; //waking up instructions
  lcdENPulse(delay4500ns); //latch enable line for 450 ns

  SysTick_Wait(delay50ms);
  lcdDB =0x3<<2;
  lcdENPulse(delay4500ns);
  //designated times to wait from datasheet
  SysTick_Wait(delay2000us);
  lcdENPulse(delay4500ns);

  SysTick_Wait(delay2000us);

  lcdDB =0x2<<2;
  lcdENPulse(delay4500ns);

  SysTick_Wait(delay2000us);
}
```

Figure 14 lcdinit() with the accent here being that lcdDB needs to written to twice in 4-bit mode

Storing numbers as “char” arrays of 20 elements (20*1 bytes = 160 bits), seems to have been a wrong choice since the largest number the calculator could process was composed of 12 numbers and not 20. This meant storing even small numbers like 1 as a char of 20 bytes, which is inefficient.

Finally, before submitting the code, a “Find and replace” was performed on most variables and some functions to give them more descriptive names. In terms of results, which where demonstrated during the project presentation, it can be observed from Table 1 that most of the objectives were met.

Bibliography

[1] Powertip, “PC 1602-F”, HD44780 Datasheet, Available here

[2] Texas Instruments, “Tiva™ TM4C123GH6PM Microcontroller”, TM4C123GH6PM Datasheet, Available here

[3] J. Valvano, R. Yerraballi, “Embedded Systems - Shape The World”, TM4C123 Reference Material, Available here

[4] Hitachi, “HD44780U (LCD-II)”, HD44780 Manual, Available here

[5] “Standard Matrix Circuit Diagram”, AK-1607 Keypad Datasheet, Available here

Now that the system is assembled, how does it know what key has been pressed? The microcontroller powers each keypad column in sequence, whilst voiding of power the rest. How fast this happens depends on the operating clock frequency, which is 80 MHz for the project. In Figure 2, the sequence of which column is powered when can be seen. When a user presses a button (Figure 2.5), they make an input to the system and create a short circuit between a row and a column. This is how the microcontroller knows what value corresponding to the key pressed should be returned and why the columns are known as “outputs” and the rows as “inputs”.

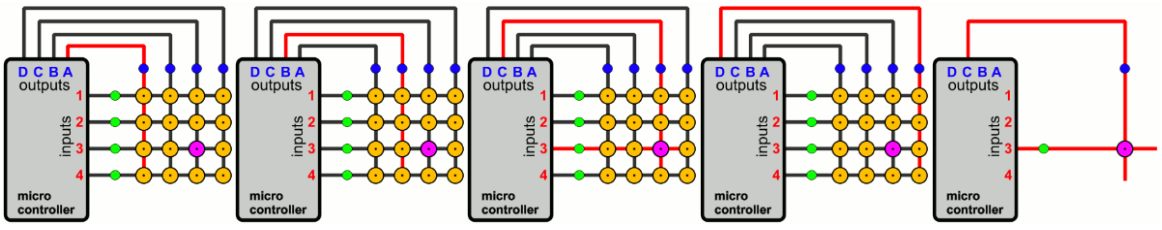


Figure 2 How the keypad inputs are read: Powering each column (1-4) and detecting an input (5)