جامعة خليفة
Khalifa University

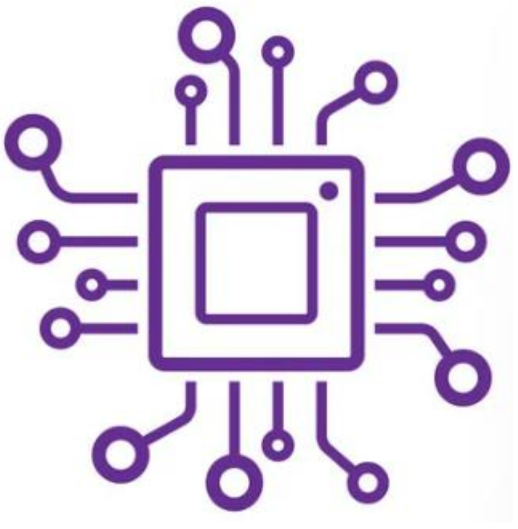
Department of Computer and Communication Engineering

MICROPROCESSOR SYSTEMS LAB – ECCE 316

Spring 2024

Project – Interfacing of Matrix Keypad

Section B4 – Group 2



Shayma Kulaib Alteneiji	100063072
Halima Aldarmaki	100058381

Table of Contents

Aim & objectives.

Aim:

To develop a key scanning program for the FRDM KL25Z microcontroller interfaced with a 4x4 matrix keypad, enabling recognition and response to key inputs, including alphanumeric, special characters (* and #), and generating patterns based on numeric inputs.

Objective:

The objectives of the project involve interfacing a 4x4 matrix keypad with the FRDM KL25Z microcontroller and developing a key scanning program. This program should recognize alphanumeric keys, display them on TeraTerm, and handle special characters (* and #), waiting for numeric input and generating patterns accordingly. The program will repeat this process for the # key. Clear feedback will be provided on TeraTerm for each key input, and error handling mechanisms will be implemented. Thorough testing, debugging, and comprehensive documentation will ensure the project's reliability and facilitate future replication.

Introduction.

4x4 Matrix Keypad

As seen in Figure 1, the 16 keys on the 4x4 matrix keypad are organized in 4 rows and 4 columns. Every time a button is pressed, the two wires (row and column lines) that connect each keypad switch are shorted. Normally, rows and columns are not connected when no switch is pressed. Connect the row lines to the output port to interface the keypad.
column lines to the microcontroller's input port as depicted in Figure 2. Figure 1 illustrates the fact that no line is attached to the ground directly. Nevertheless, by setting the output port pins to "0," the desired lines can be forced to the ground.

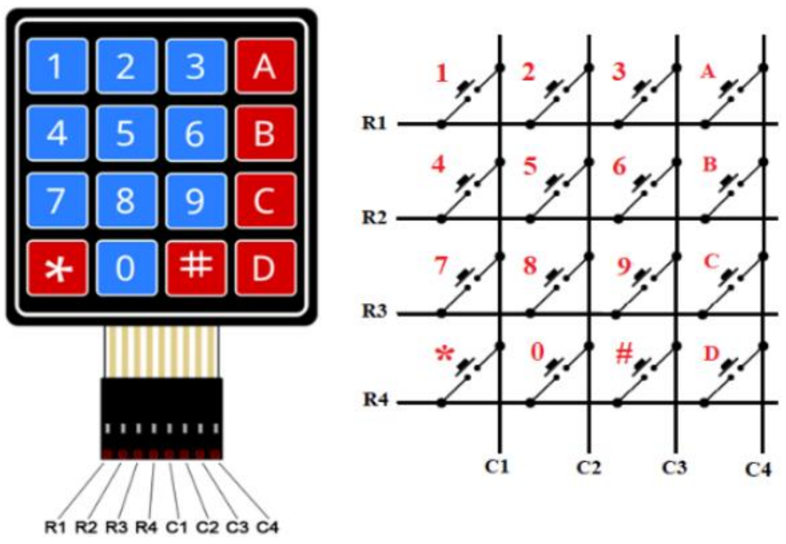


Figure 1: 4x4 matrix keypad

To detect the key that is being pressed, a row is alternately grounded, and the data input register of port E is read. If a bit corresponding to either of the pins (PTE0-PTE3) is zero, the microcontroller then knows that the key corresponding to (Ri, Cj), (i,j = 1,2,3,4) is pressed; where Cj is the column corresponding to the PTEi pin inputting a zero, and Rj is the row that is being grounded when the columns inputs are read. For instance, if row R2 and column C2 are both "0," then key 5 has been hit. It is possible to create the keypad scanning algorithm using this reasoning. The bit combinations in Table 1's row and column lines demonstrate how to determine which key has been pressed.

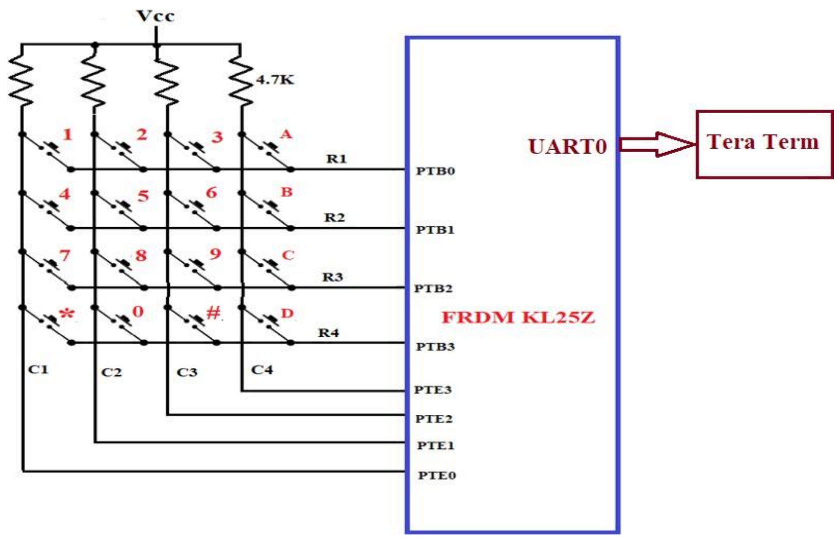


Figure 2. Interfacing of 4x4 matrix keypad and 7-segment display.

Table 1

Port B	PTB3	PTB2	PTB1	PTB0	Port E	PTE3	PTE2	PTE1	PTE0	Key pressed
Row	R4	R3	R2	R1	Column	C4	C3	C2	C1	
0x0E	1	1	1	0	0x0E	1	1	1	0	**1**
					0x0D	1	1	0	1	**2**
					0x0B	1	0	1	1	**3**
					0x07	0	1	1	1	**A**
0x0D	1	1	0	1	0x0E	1	1	1	0	**4**
					0x0D	1	1	0	1	**5**
					0x0B	1	0	1	1	**6**
					0X07	0	1	1	1	**B**
0x0B	1	0	1	1	0x0E	1	1	1	0	**7**
					0x0D	1	1	0	1	**8**
					0x0B	1	0	1	1	**9**
					0x07	0	1	1	1	**C**
0x07	0	1	1	1	0x0E	1	1	1	0	*****
					0x0D	1	1	0	1	**0**
					0x0B	1	0	1	1	**#**
					0x07	0	1	1	1	**D**

Elimination of Key Bouncing

When a push button key is pressed or released, the metal contacts undergo a rapid series of making and breaking contact, resulting in a bouncing effect before settling into a steady state input, as depicted in Figure 3. Typically, this bouncing phenomenon lasts for approximately 20 to 30 milliseconds. This issue, known as key bouncing, can lead to fast devices like microcontrollers interpreting a single key press as multiple presses. Therefore, it's crucial to prevent the bouncing of the key from being registered as input. Key debouncing techniques are commonly employed to address this problem, which can be implemented either through hardware (utilizing flip-flops) or software (employing a delay). The software-based key debouncing method can be easily executed by introducing a delay in the key scanning program. This delay ensures that when a key closure is detected, the microprocessor pauses for 20 to 30 milliseconds before considering the key as a valid input.

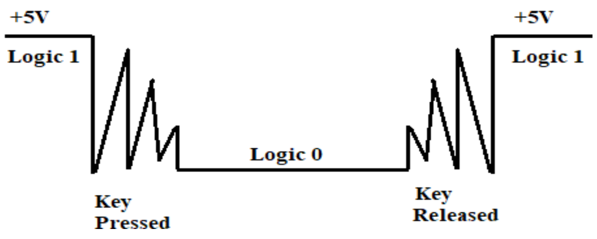


Figure 3 Debouncing effect on signal read.

Hardware Design.

The interfacing circuit of the microcontroller and the keypad is shown in figure 4, the circuit schematics is shown in figure 5. The circuit compromises the microcontroller, a keypad, four pull-up resistors, and jumper wires. The eight pins of the keypad correspond to its four columns and four rows.

As mentioned in the introduction, the rows R1-R4 are connected to the microcontroller through pins (PTB0-PTB3) which are configured as output pins; while the columns pins are connected to the microcontroller through pins (PTE0 – PTE3) which are configured as inputs.

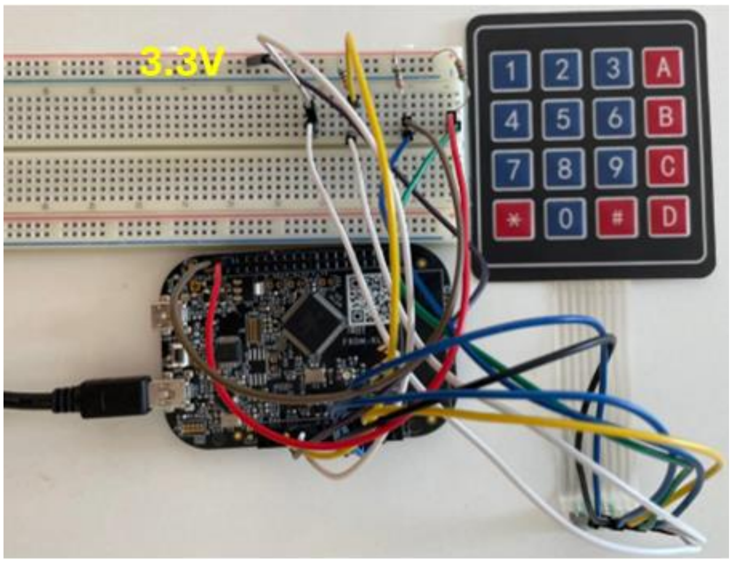


Figure 4. Program main flow chart.

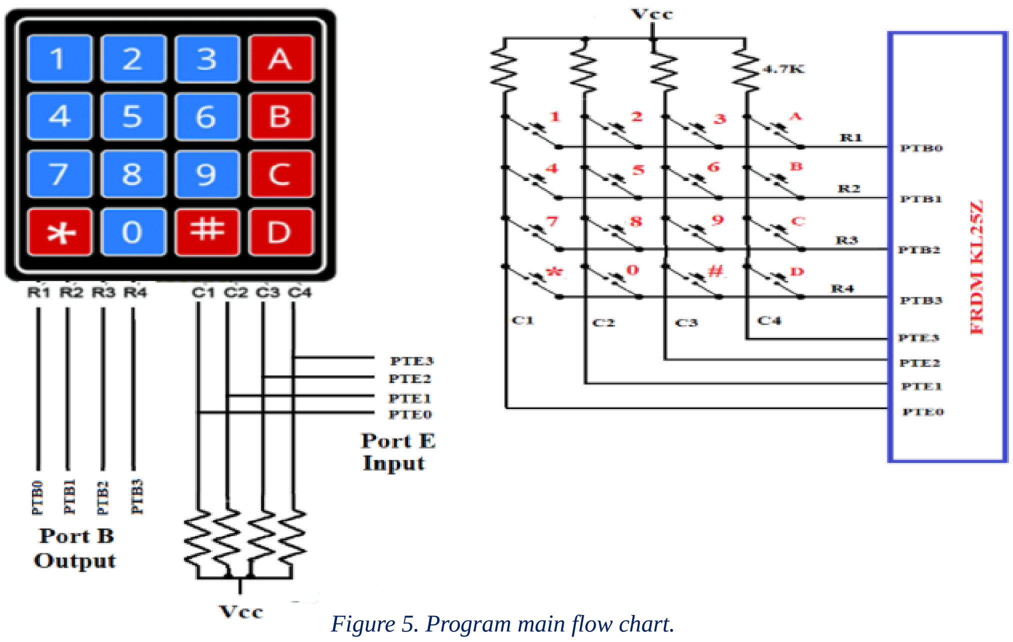


Figure 5. Program main flow chart.

The columns of the keypad are pulled high by pull-up resistors and connected to the microcontroller pins PTE0- PTE3 as mentioned; this is so that the input read by the microcontroller is always high (1) and only when a key is pressed (in the rows that being grounded), the input read by the microcontroller is zero. The circuit diagram of the pull-up resistors in terms of the given is shown in figures 6(a), and (b).

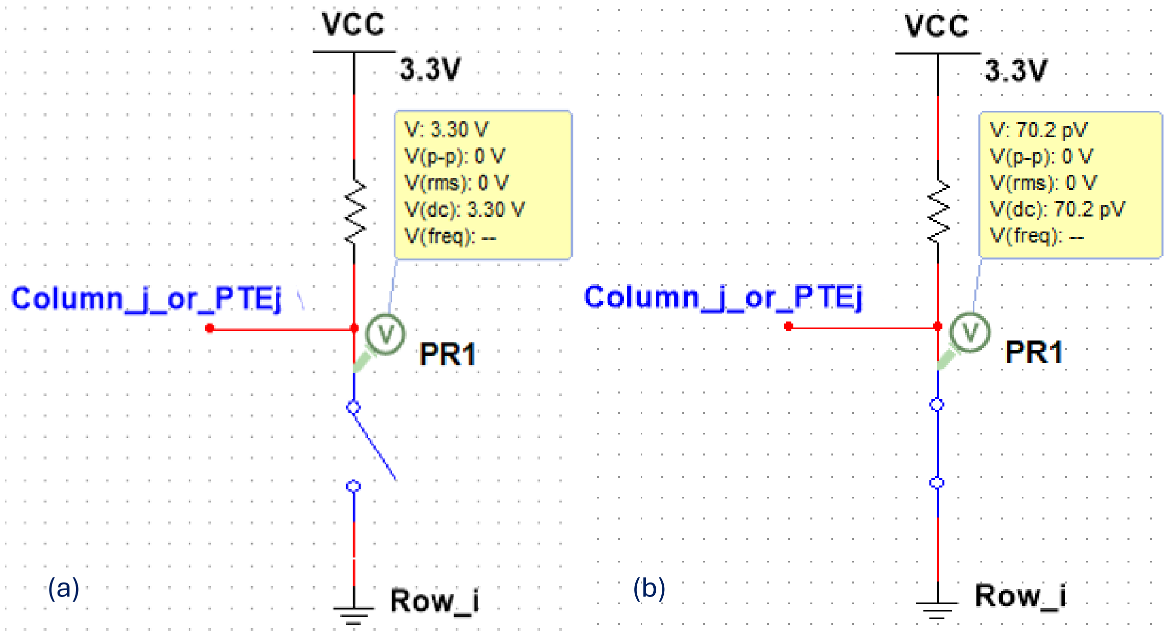


Figure 6. Pull-up resistor circuit, (a) input read is 1, meaning that the key in (Ri, Cj) is not being pressed. (b) input read is 0, meaning that the key in (Ri, Cj) is being pressed.

Software Design.

In writing the code of the program, the following simple flow chart, figure 7, summarizes the entire code program code given in appendix A. The process is repeated infinitely many times with no termination. A delay of 100ms is used to eliminate bouncing and to obtain a single output on Tera Term per key pressed.

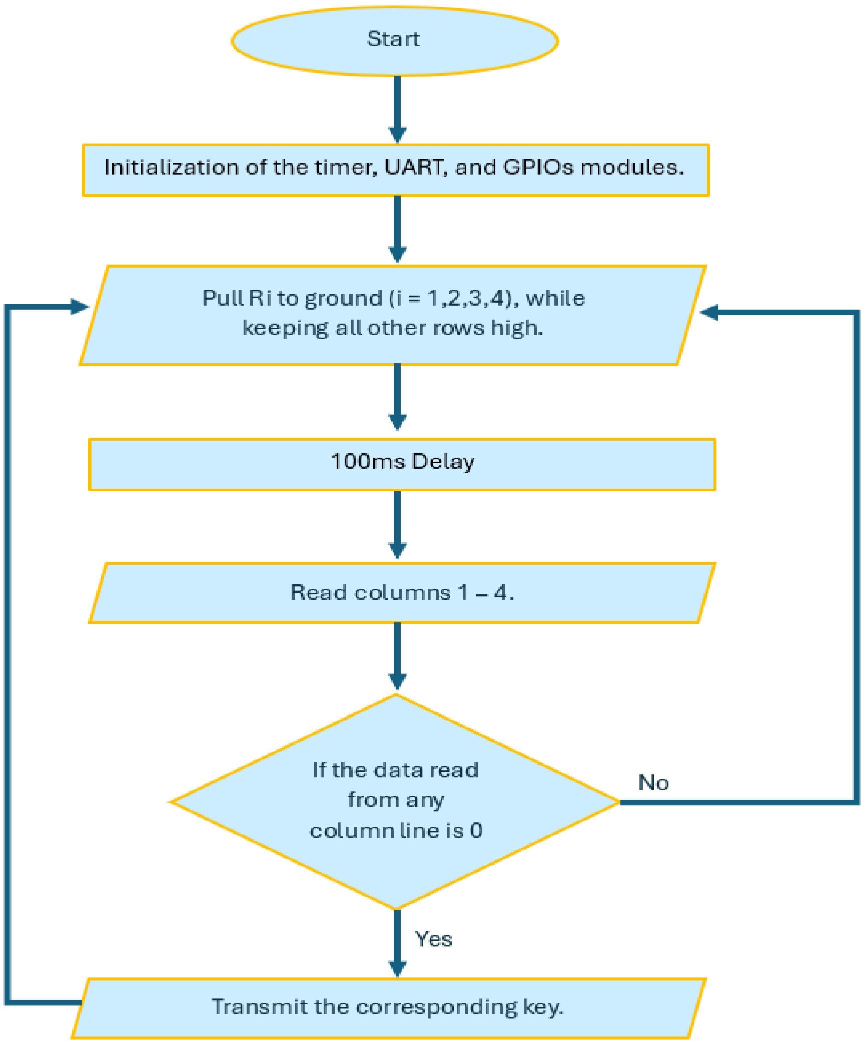
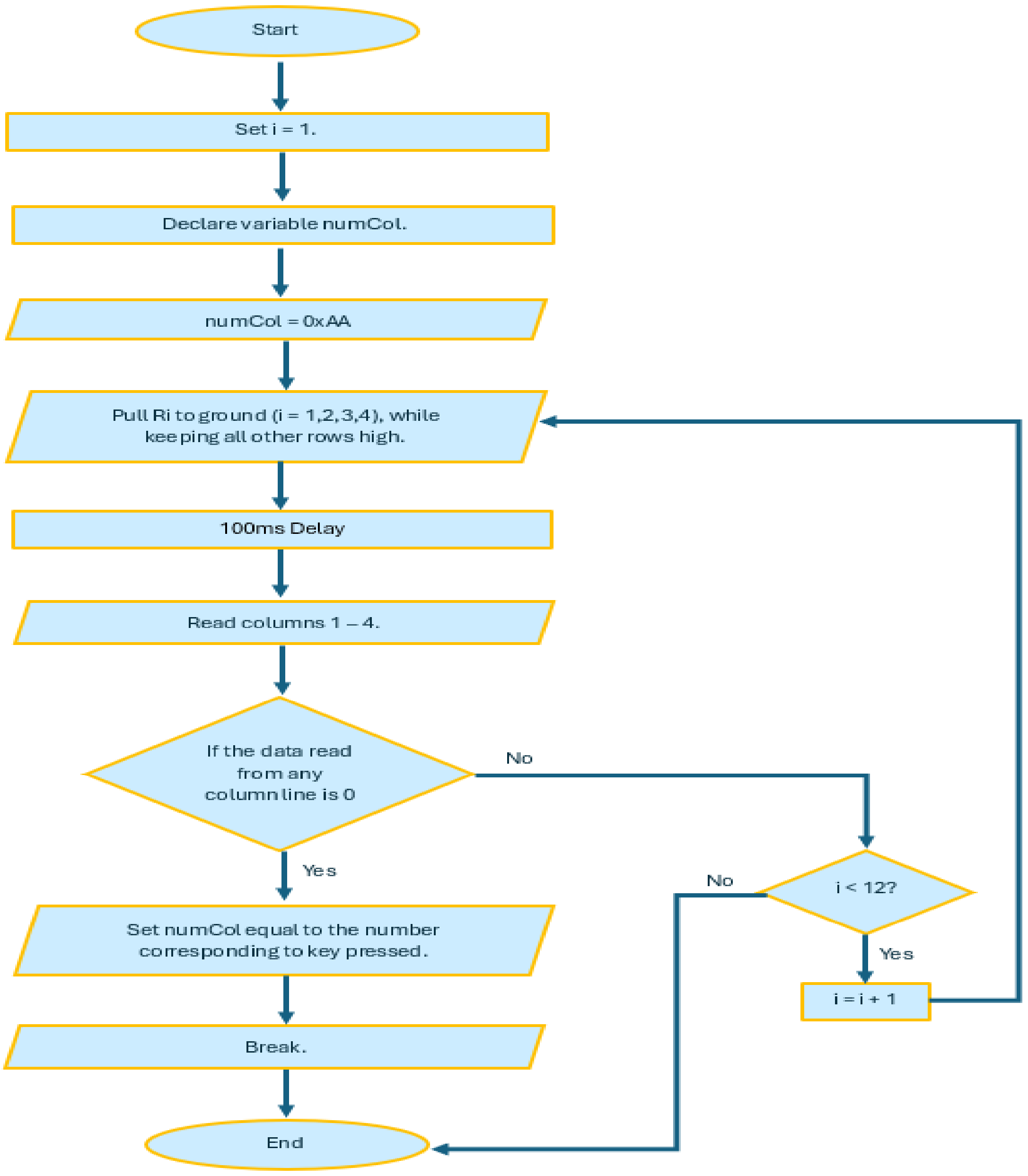

Start
Initialization of the timer, UART, and GPIOs modules.
Pull Ri to ground (i = 1,2,3,4), while keeping all other rows high.
100ms Delay
Read columns 1 – 4.
If the data read from any column line is 0
No
Yes
Transmit the corresponding key.

Figure 7. Program main flow chart.

As for the feature regarding the case when a (* or #) is pressed, the code structure used is described by the flow chart given by figure 8. Again, a 100ms delay is used before reading the data input register of the columns.

For the process of continually checking whether a numerical key is pressed within 5 seconds, the code is executed 12 times using a for loop; 4 delays per column checked * 12* delay (=100ms), the resulting total delay is 4.8sec, which can be approximated to equal to 5sec if considering the delay required to execute the other instructions within the loop. Moreover, numCol is initially set to 0xAA such that if numCol value is unchanged, it would indicate that no key was pressed within the 5 seconds period, and the program continues from the beginning of the while (1) loop.



Start
Set i = 1.
Declare variable numCol.
numCol = 0xAA
Pull Ri to ground (i = 1,2,3,4), while keeping all other rows high.
100ms Delay
Read columns 1 – 4.
If the data read from any column line is 0
No
Yes
i < 12?
No
Yes
i = i + 1
Set numCol equal to the number corresponding to key pressed.
Break.
End

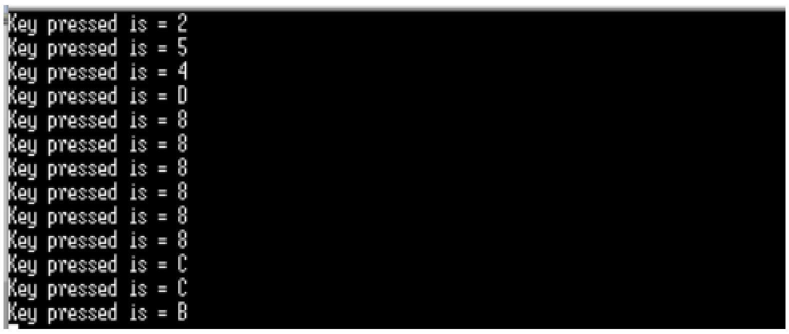
Figure 8. Program sub flow chart to achieve the feature of checking the numerical key input whin 5 second period.

Analysis of Results.

This section discusses the program results of four different cases. Case one describes the Tera Term output of pressing a numerical or alphabetical key. Case two describes the output when a (* and a #) was pressed, followed by pressing a numerical key before the 5 seconds period is elapsed. Finally, case three discusses the Tera Term result when no key was pressed within the 5 seconds period of pressing a (* and #).

Case One

Figure 9 shows the Tera Term result upon pressing different keys (excluding * and #). Per key pressed only a single output is displayed on Tera Term.



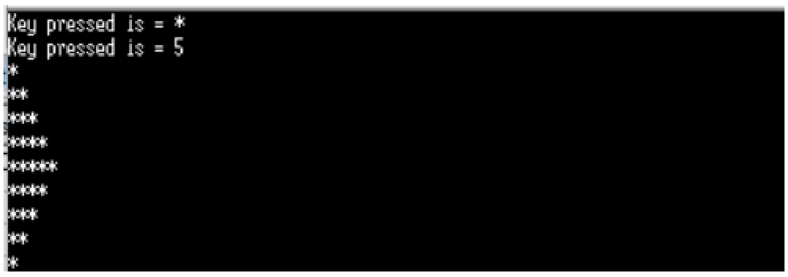
```
Key pressed is = 2
Key pressed is = 5
Key pressed is = 4
Key pressed is = D
Key pressed is = 8
Key pressed is = 8
Key pressed is = 8
Key pressed is = 8
Key pressed is = 8
Key pressed is = 8
Key pressed is = C
Key pressed is = C
Key pressed is = B
```

Figure 9. Tera Term output of Case one.

Case Two

Case Two (a)

Figure 10 shows the Tera Term output when a (*) was pressed followed by the number (5). As shown in the figure, the number of columns of the displayed pattern is five.

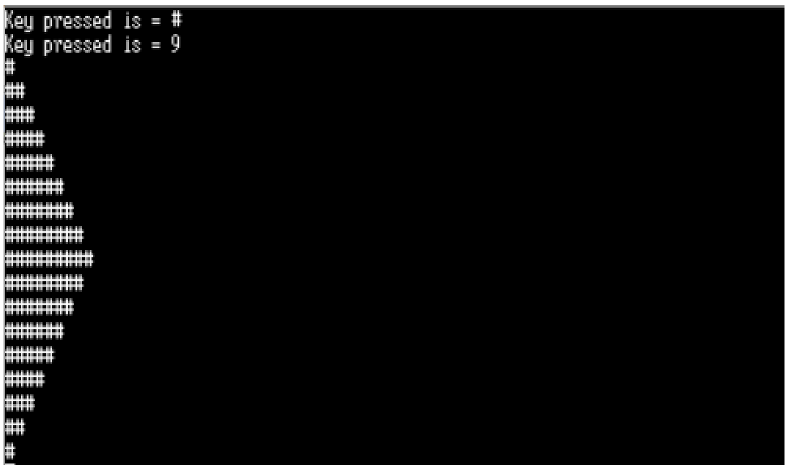


```
Key pressed is = *
Key pressed is = 5
*
**
***
****
*****
****
***
**
*
```

Figure 10. Tera Term output of Case two (a).

Case Two (b)

When a (#) was pressed followed by a (9) within 5 seconds, the program showed the pattern seen in figure 11, the number of columns of the pattern is nine ae desired.

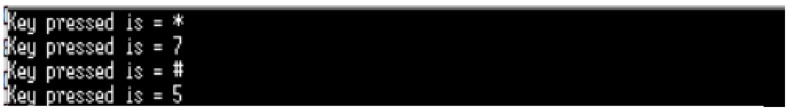


```
Key pressed is = #
Key pressed is = 9
#
##
###
####
#####
######
#######
########
#########
########
#######
######
#####
####
###
##
#
```

Figure 11. Tera Term output of Case two (a).

Case Three

When a (*) is pressed, then 5 seconds is counted, and then the numerical key (7) was pressed, the pattern is not formed, and only the output (Key pressed = 7) is displayed as desired, as shown in figure 12. The same output is observed when preceding (#) followed by pressing the number (5) after 5 seconds is elapsed, also shown in figure 12.



```
Key pressed is = *
Key pressed is = 7
Key pressed is = #
Key pressed is = 5
```

Figure 12. Tera Term output of Case Three.

Discussions and Conclusions.

In conclusion, this project achieved its goal of connecting a 4x4 matrix keypad to the KL 25Z microcontroller. The primary objective of this project was to create an embedded C program capable of accurately detecting and interpreting key presses while showing respective feedback on a terminal emulator. The problem statement revolved around the need to design an algorithm that can interpret electrical connections formed by key presses and translate them into identifiable characters or actions. By understanding how matrix keypads work, we built a system that recognizes when a key is pressed by checking rows and columns. An efficient keypad scanning algorithm was developed, sequentially checking row lines, and efficiently checking column pins to detect key presses. Additionally, the terminal emulator (Tera Term) effectively displayed alpha-numeric values corresponding to the pressed keys, providing real-time feedback to the user.

Appendix A- Program Code

```
# include <MKL25Z4.h>
# include <string.h>

int main (void)
{
void DELAY_hundred_milis(void);
int i;
int col_int,p,k,m;
int numCol;
char message [] = "Key pressed is = ";

SIM-> SCGC5|= SIM_SCGC5_PORTB(1);          // Enable port B clock gate
SIM-> SCGC5|= SIM_SCGC5_PORTE(1);          // Enable port E clock gate
for (i=0; i<4; i++) {          // Configure the control register of pins 0-3 of ports B and E as GPIO
PORTB->PCR[i] = PORT_PCR_MUX(1);
PORTE->PCR[i] = PORT_PCR_MUX(1);
 }
GPIOB-> PDDR |= 0xF;                       // Configure pins 0 -3 of port B an output
GPIOE-> PDDR &= ~(0x0000000F);             // Configure pins 0 -3 of port B an input

MCG_C1 |= MCG_C1_CLKS(0);                  //
MCG_C1 |= MCG_C1_IREFS(1);
MCG_C4 |= MCG_C4_DRST_DRS(1);
MCG_C4 |= MCG_C4_DMX32(1);

SIM-> SCGC4 |= SIM_SCGC4_UART0(1);         //Enable the clock gate of the UART0 module
SIM-> SCGC5 |= SIM_SCGC5_PORTA(1);         // Enable the clock gate of port A
SIM-> SOPT2 |= SIM_SOPT2_UART0SRC(1);      // Select MCGFLLCLK as UART0 clock
UART0 -> C2 = 0x0;                    // Disable the transmitter and receiver while
configuration
UART0 -> C4 = 0xF;                    // Set the over sampling ratio to 15
UART0 -> C1 = 0x0;               // Configure serial port for 8-bit data, one stop bit, and no parity
UART0-> BDL = 0x38;              // Set the baud rate as 9600 bauds
UART0-> BDH = 0x01;

UART0->C2 |= UART_C2_TE(1);           // Enable the UART0 transmitter
PORTA->PCR[2] = PORT_PCR_MUX(2); // Configure the control register of port A pin 2 for S.C.

UART0->C2|= UART_C2_RE(1);            // Enable the UART0 receiver
PORTA->PCR[1] = PORT_PCR_MUX(2); // Configure the control register of port A pin 1 for S.C.

while (1) {
GPIOB-> PDOR = 0xE;
```

```
DELAY_hundred_milis();
if (! (GPIOE -> PDIR & 0x8))
{
 for (m = 0; m <strlen(message) ; m++)

 {
 while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 'A';
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;
}
else if (!( GPIOE -> PDIR & 0x4))

{
        for (m = 0; m<strlen(message) ; m++)
         {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x33;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;
}

else if (!( GPIOE -> PDIR & 0x2))

{
        for (m = 0; m<strlen(message) ; m++)
        {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x32;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))){}
        UART0 -> D = 0x0A;
}
```

```
else if   (!( GPIOE -> PDIR & 0x1))
{
          for (m = 0; m<strlen(message) ; m++)
          {
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = message[m];
          }
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x31;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0D;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0A;
}

GPIOB-> PDOR = 0xD;

DELAY_hundred_milis();

if (!( GPIOE -> PDIR & 0x8))
 {
          for (m = 0; m<strlen(message) ; m++)
          {
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = message[m];
          }
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 'B';

          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0D;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0A;
}
else if (!( GPIOE -> PDIR & 0x4))
 {
          for (m = 0; m<strlen(message) ; m++)
           {
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = message[m];
          }
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x36;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0D;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0A;
}


else if (!( GPIOE -> PDIR & 0x2))
{
          for (m = 0; m<strlen(message) ; m++)
          {
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = message[m];
          }
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x35;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0D;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0A;
}
else if (!( GPIOE -> PDIR & 0x1))
 {
          for (m = 0; m<strlen(message) ; m++)
           {
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = message[m];
          }
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x34;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0D;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0A;
}
GPIOB-> PDOR = 0xB;
 DELAY_hundred_milis();

if (!( GPIOE -> PDIR & 0x8)) {
          for (m = 0; m < strlen(message) ; m++)
           {
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = message[m];
          }
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 'C';
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0D;
          while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
          UART0 -> D = 0x0A;
}
```

```
else if (!( GPIOE -> PDIR & 0x4))
{
        for (m = 0; m<strlen(message) ; m++)
        {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x39;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;
}
else if (!( GPIOE -> PDIR & 0x2))
{
        for (m = 0; m<strlen(message) ; m++)
         {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x38;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;
}
else if (!( GPIOE -> PDIR & 0x1))
{
        for (m = 0; m<strlen(message) ; m++)
        {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x37;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;
}

GPIOB-> PDOR = 0x7;
DELAY_hundred_milis();
```

```
if (!( GPIOE -> PDIR & 0x8))
{
        for (m = 0; m<strlen(message) ; m++)
         {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 'D';
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;
}
else if (!( GPIOE -> PDIR & 0x4))
{
        for (m = 0; m<strlen(message) ; m++)
         {
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = message[m];
        }
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = '#';
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0D;
        while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = 0x0A;

        numCol = 0xAA;
        for (i=0; i<12; i++)
        {
        GPIOB->PDOR = 0xE;
        DELAY_hundred_milis();
          if (!( GPIOE -> PDIR & 0x4))
          {
          numCol = 0x33;
          break;
          }
          else if (!( GPIOE -> PDIR & 0x2))
          {
          numCol = 0x32;
          break;
          }
          else if (!( GPIOE -> PDIR & 0x1))
          {
            numCol = 0x31;
            break;    }
          GPIOB->PDOR = 0xD;
          DELAY_hundred_milis();
            if (!( GPIOE -> PDIR & 0x4))
            {
            numCol = 0x36;
            break;
            }
            else if (!( GPIOE -> PDIR & 0x2))
            {
            numCol = 0x35;
            break;
            }
            else if (!( GPIOE -> PDIR & 0x1))
            {
            numCol = 0x34;
            break;    }
          GPIOB->PDOR = 0xB;
          DELAY_hundred_milis();
            if (!( GPIOE -> PDIR & 0x4))
            {
            numCol = 0x39;
            break;
            }
            else if (!( GPIOE -> PDIR & 0x2))
            {
            numCol = 0x38;
            break;
            }
            else if (!( GPIOE -> PDIR & 0x1))
            {
            numCol = 0x34;
            break;    }
          GPIOB->PDOR = 0x7;
          DELAY_hundred_milis();

            if (!( GPIOE -> PDIR & 0x1))
            {
            numCol = 0x30;
            break;    }
          }

if (numCol == 0xAA)
{
continue;
}
 for (m = 0; m<strlen(message) ; m++)
 {
  while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
      UART0 -> D = message[m];
       }
       while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
        UART0 -> D = numCol;
        col_int = numCol - '0';
         while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
         UART0 -> D = 0x0D;
         while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
         UART0 -> D = 0x0A;
          for (p = 1; p<= col_int;p++)
           {
            for ( k =1; k<=p ; k++)
             {
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = '#';
              }
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = 0x0A;
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
               UART0_D = 0x0D;
                    }
            for (p = 1; p< col_int;p++)
            {
             for ( k =p; k<col_int ; k++)
             {
             while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = '#';
              }
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = 0x0A;
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = 0x0D;
             }
            }
else if (!( GPIOE -> PDIR & 0x2)){
           for (m = 0; m<strlen(message) ; m++)
           {
           while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
           UART0 -> D = message[m];
           }
           while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
           UART0 -> D = 0x30;
           while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
           UART0 -> D = 0x0D;
           while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
           UART0 -> D = 0x0A;
```

```
}

else if (!( GPIOE -> PDIR & 0x1))
{
for (m = 0; m<strlen(message) ; m++)
{
 while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
 UART0 -> D = message[m];
}
while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
UART0 -> D = '*';
while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
UART0 -> D = 0x0D;
while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
UART0 -> D = 0x0A;
numCol = 0xAA;
            for (i=0; i<12; i++)
            {
            GPIOB->PDOR = 0xE;
            DELAY_hundred_milis();
               if (!( GPIOE -> PDIR & 0x4))
               {
               numCol = 0x33;
               break;
               }
               else if (!( GPIOE -> PDIR & 0x2))
               {
               numCol = 0x32;
               break;
               }
               else if (!( GPIOE -> PDIR & 0x1))
               {
               numCol = 0x31;
               break;    }
            GPIOB->PDOR = 0xD;
            DELAY_hundred_milis();
               if (!( GPIOE -> PDIR & 0x4))
               {
               numCol = 0x36;
               break;
               }
               else if (!( GPIOE -> PDIR & 0x2))
               {
               numCol = 0x35;
               break;
               }
               else if (!( GPIOE -> PDIR & 0x1))
               {
```

```
              numCol = 0x34;
              break;    }
            GPIOB->PDOR = 0xB;
            DELAY_hundred_milis();
              if (!( GPIOE -> PDIR & 0x4))
              {
              numCol = 0x39;
              break;
              }
              else if (!( GPIOE -> PDIR & 0x2))
              {
              numCol = 0x38;
              break;
              }
              else if (!( GPIOE -> PDIR & 0x1))
              {
              numCol = 0x34;
              break;    }
            GPIOB->PDOR = 0x7;
            DELAY_hundred_milis();

              if (!( GPIOE -> PDIR & 0x1))
              {
              numCol = 0x30;
              break;    }
            }

if (numCol == 0xAA)
{
continue;
}
 for (m = 0; m<strlen(message) ; m++)
 {
  while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
     UART0 -> D = message[m];
     }
     while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
      UART0 -> D = numCol;
      col_int = numCol - '0';
       while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
       UART0 -> D = 0x0D;
       while (!(UART0 -> S1 & UART0_S1_TDRE(1))) {}
       UART0 -> D = 0x0A;

        for (p = 1; p<= col_int;p++)
         {
          for ( k =1; k<=p ; k++)
           {
            while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
```

```
                UART0_D = '*';
                }
                while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
                UART0_D = 0x0A;
                while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
                 UART0_D = 0x0D;
                        }
            for (p = 1; p< col_int;p++)
            {
             for ( k =p; k<col_int ; k++)
             {
             while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = '*;
              }
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = 0x0A;
              while((UART0_S1 & UART0_S1_TDRE(1))==0) {}
              UART0_D = 0x0D;
             }
            }
}
}

void DELAY_hundred_milis(void)
 {
SIM -> SCGC6 |= SIM_SCGC6_TPM0(1);
SIM -> SOPT2 |= SIM_SOPT2_TPMSRC(1);
TPM0 -> SC = 0x0;
TPM0 -> MOD = 0xFFFF;
TPM0 -> SC |= TPM_SC_PS(6); // Set the pre-scaling factor to 2^6 =  64
TPM0 -> SC |= 0x80; // Clear the timer overflow flag
TPM0 -> SC |= 0x08; // enable the timer free-running mode

            while (!(TPM0->SC &0x80)) {} // wait until timer overflow flag is set
           TPM0->SC |= TPM_SC_TOF(1); // clear TOF
}
```