Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

**Report week 6: Introduction to ARM MCU programming**

Laboratory of Microcontrollers

Gilberto Ochoa Ruiz


Perla Vanessa Jaime Gaytán ITE

A00344428

Nicol Gomez Tisnado

A00227180

Instituto Tecnológico de Estudios Superiores de Monterrey Campus Guadalajara

Zapopan, Jalisco, México.


April 13, 2020

1. **Include some research about how the LCD screen and keyboard operate in the corresponding section. Provide enough detail to demonstrate that you understand the basics and how the concepts relate**

Liquid crystal display technology works by blocking light. Specifically, an LCD is made of two pieces of polarized glass (also called substrate) that contain a liquid crystal material between them. A backlight creates light that passes through the first substrate. At the same time, electrical currents cause the liquid crystal molecules to align to allow varying levels of light to pass through to the second substrate and create the colors and images that you see. The most basic of the pins are the power pins for the display to be able to function in the first place. There is a VDD pin for 5 volts and a VSS pin for ground. Just like the microcontroller, the LCD has a row of 8 pins to serve as its port. The pins that serve as its port is D0, D1, D2, D3, D4, D5, D6 and D7. These pins are generally used to pass information into the LCD, but it can also be set to pass information back to the microcontroller. Two types of information can be passed through these pins: data to display on the LCD screen, or control information that is used to do things such as clearing the screen, controlling the cursor position, turning the display on or off, etc. These data pins are connected to the pins of the desired port on the microcontroller.

In a 4×4 matrix keypad, there are four rows and four columns connected to 16 push button switches. It may look like one needs 16 pins for the microcontroller to be connected to the matrix keypad, but practically 16 inputs of keypad interface are possible with the 8 pins of a microcontroller port. All 8 lines can be connected to the same port or different ports based on the application requirements. In fact, 8 port pins of a microcontroller are sufficient for a 4×4 keypad interface using row & column

matrix connection technique by saving other 8 bits of the port. Matrix keypad interfacing and key press identification can be explained in a step by step manner which involves a software. First, we input high on row pins. When a key is pressed, the corresponding row and column get shorted. In the second step, a software scans the pins connected to the columns. If it detects a high on any particular column, then it is found that the key press has been made on a key in that column. The third step is to figure out which key is pressed exactly. For this, the software writes logic high on row pins sequentially. The pin of the column on which the pressed key is situated will become high.

Going into the relation between both concepts, we could say that the keyboard/keypad acts as an input of data to the microcontroller while the LCD operates as an output for the same data.

2. **Include research about noise in digital systems and the need for the debouncer. Include any research you have done and how you implemented the solution for the last part of the lab.**

Noise in digital systems, is also known as bouncing which is the tendency of any two metal contacts in an electronic device to generate multiple signals as the contacts close or open. These transitions are due to mechanic and physical issues. Basically, means that when you pressed a button you may read multiple presses in a very short time fooling the program you are using. Debouncing checks twice in a short period of time to make sure the button is pressed.

There are two categories of solutions: Hardware and Software. Here are some examples of each category.

Hardware:

- Most microcontrollers have internal pullup resistors that can be activated for any given GPIO pin. This lets you establish a definite logic state on a button when it's not pressed by pulling the voltage back up from ground.
- The simplest which works most of the time is debounce with a RC filter. If you select your resistor and your capacitor for the product to be large enough, it will work most of the time.
- To get it right all the time is using a logic chip like: 74HC14 or 40106.

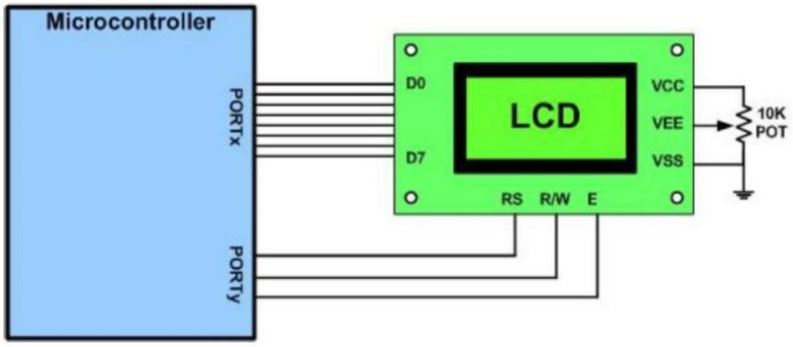
Software:

- Attempts to wait until after the bouncing has stopped before declaring a button press or release. If the switch is still bouncing after a delay time, it delays again until it stables.
- There is the counter-or integrator-based debouncers, which is the equivalent to the RC filter.
- Pattern-based debouncer: takes the overall pattern of the switch's voltage output over a relatively long time period into a account.

The one that we decided to implement was in software which basically checks several times to make sure that the button is pressed to stop reading wrong signals. It is extremely useful when the key is pressed. The code that we implemented is on Part IV.

## Part I.

**Write the code for writing the Hello World of embedded systems when working with the LCD screen. The first part will be done with the 8-bit configuration seen in class. Make use of the internal ready signal instead of delays.**




## Code:

In order to use the registers of the microcontroller we need specificate which library is the one we need and its peripherals. After that we define some constants in order to use them later on. We also declared all the functions that we created in this code.

```
#include "MKL25z4.h"
#include "derivative.h"

#define RS 0x04     // PTA2 mask
#define RW 0x10     // PTA4 mask
#define EN 0x20     // PTA5 mask

void delayMs(int n);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);
```

Then we have the delayMs functions. The first one is commented because is the one that the code has as an example, but in this case, we were required to use the

registers that has the microcontroller and make use of an internal ready signal. This function only wait for the time that it is requiere.

```
/*void delayMs(int n) {
    int i;
    int j;
    for(i = 0 ; i < n; i++){
        for(j = 0 ; j < 7000; j++) { }
    }
}*/

void delayMs(int n)
{
    LPTMR0_CSR = 0; //se limpia todo el registro para poder utilizarlo
    LPTMR0_CMR = n; //Es con el tiempo que se va comparar
    LPTMR0_PSR = 0x00000005; //Prescaler/glitch filter clock 1 selected and Prescaler/glitch filter is bypassed.
    LPTMR0_CSR = 0x00000001; // This one enbable LPTMR to para que no haga un reset interno
    while(!(LPTMR_CSR_TCF_MASK & LPTMR0_CSR)){

    }
}
```

This function writes a character on the LCD.

```
void LCD_data(unsigned char data){
     GPIOA_PSOR = RS;          // RS = 1
     GPIOA_PCOR = RW;          // RW = 0
     GPIOD_PDOR = data;
     GPIOA_PSOR = EN;          //pulse E
     delayMs(0);              //it just a pause without pausing
     GPIOA_PCOR = EN;
     delayMs(1);
 }
```

This function clears the LCD and sets different function on it.

```
void LCD_command(unsigned char command){
     GPIOA_PCOR = RS|RW;          /*RS=0,R/W=0*/
     GPIOD_PDOR = command;
     GPIOA_PSOR = EN;         //pulse E
     delayMs(0);
     GPIOA_PCOR = EN;

     if (command < 4)
     delayMs(4);          /* command 1 and 2 needs up to 1.64ms */
     else
     delayMs(1);          /* all others 40 us */
 }
```

This function is where all the pins are initialized in order to use them. It also enable the clock and prepare the LCD to write.

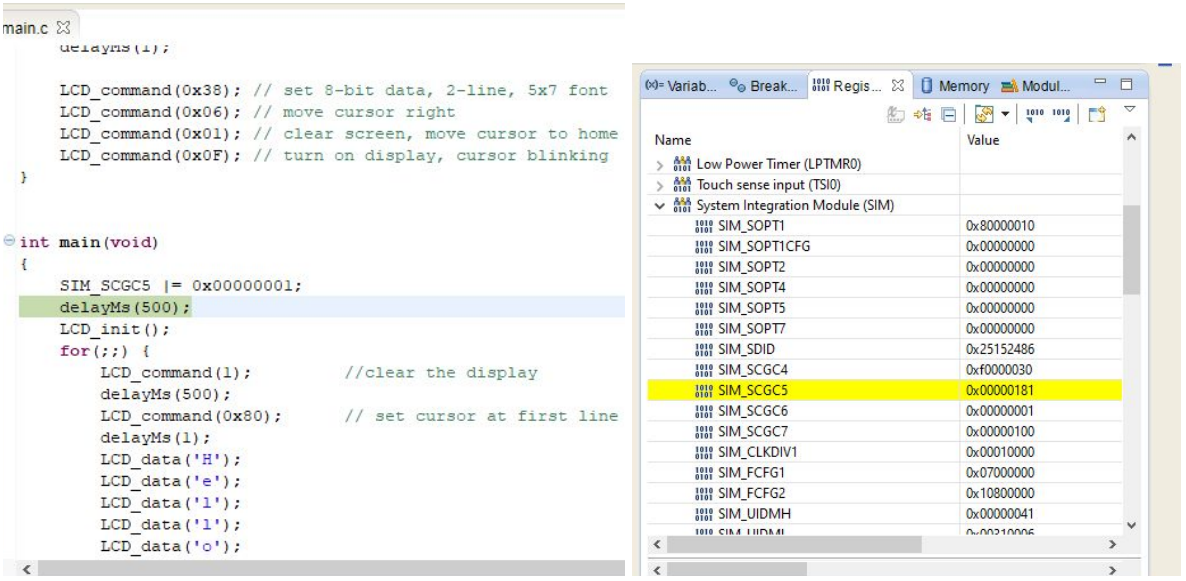
```
void LCD_init(void){
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO
    GPIOD_PDDR = 0xFF;        // make PTD7-0 as output pins

    SIM_SCGC5 |= 0x0200;      // enable clock to Port A
    PORTA_PCR2 = 0x100;       // make PTA2 pin as GPIO
    PORTA_PCR4 = 0x100;       // make PTA4 pin as GPIO
    PORTA_PCR5 = 0x100;       // make PTA5 pin as GPIO
    GPIOA_PDDR |= 0x34;       // make PTA5, 4, 2 as output pins

    delayMs(30);              // initialization sequence
    LCD_command(0x30);
    delayMs(10);
    LCD_command(0x30);
    delayMs(1);
    LCD_command(0x30);
    delayMs(1);

    LCD_command(0x38); // set 8-bit data, 2-line, 5x7 font
    LCD_command(0x06); // move cursor right
    LCD_command(0x01); // clear screen, move cursor to home
    LCD_command(0x0F); // turn on display, cursor blinking
}
```

This one is the function that is going to be executed by the IDE taking in to the microcontroller.

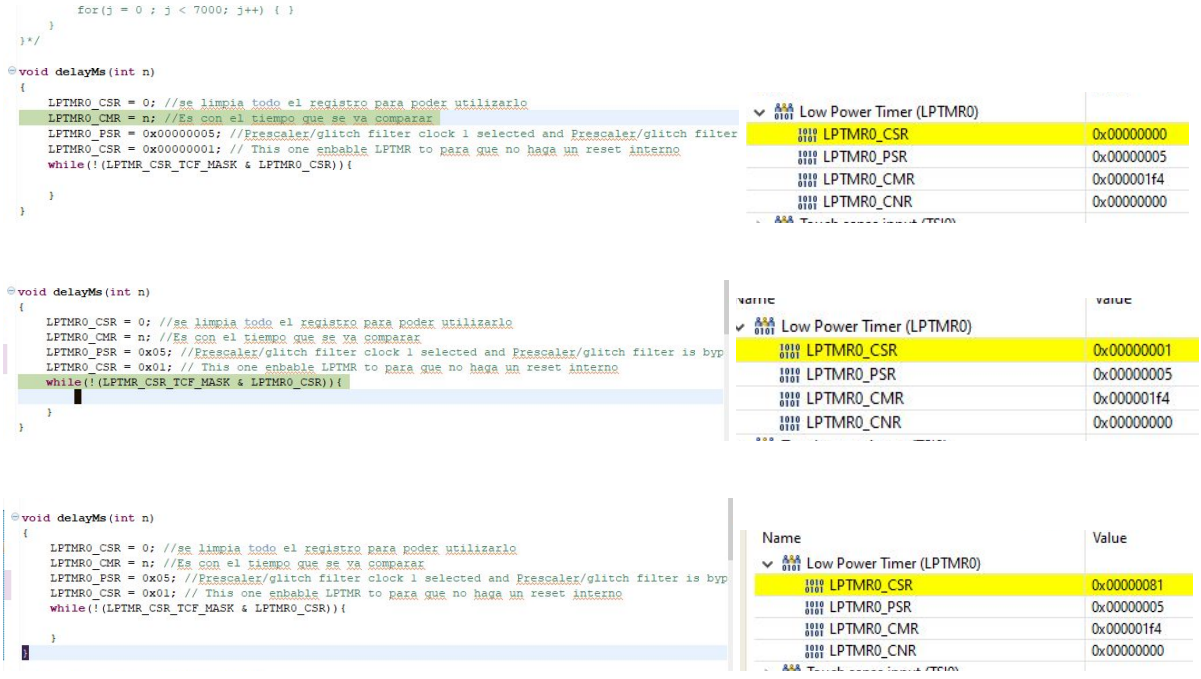
```
int main(void)
{
    SIM_SCGC5 |= 0x00000001;
    delayMs(500);
    LCD_init();
    for(;;) {
        LCD_command(1);          //clear the display
        delayMs(500);
        LCD_command(0x80);       // set cursor at first line
        delayMs(1);
        LCD_data('H');
        LCD_data('e');
        LCD_data('l');
        LCD_data('l');
        LCD_data('o');
        delayMs(500);
    }

    return 0;
}
```

## Registers:

This shows how the register of the clock is assigned.




This part shows how the delay works with the internal flag of the microcontrollers and its registers.



Here it shows how the registers change as initializing it for the LCD.

```
}
void LCD_init(void){
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
```

| | |
|---|---|
| SIM_SCGC4 | 0xf0000030 |
| SIM_SCGC5 | 0x00001181 |
| SIM_SCGC6 | 0x00000001 |

```
void LCD_init(void){
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO
    GPIOD_PDDR = 0xFF;        // make PTD7-0 as output pins
```

| Pin Control and Interrupts (PORTD) | |
|---|---|
| PORTD_PCR0 | 0x00000105 |
| PORTD_PCR1 | 0x00000105 |
| PORTD_PCR2 | 0x00000105 |
| PORTD_PCR3 | 0x00000105 |
| PORTD_PCR4 | 0x00000101 |
| PORTD_PCR5 | 0x00000101 |
| PORTD_PCR6 | 0x00000101 |
| PORTD_PCR7 | 0x00000101 |
| PORTD_PCR8 | 0x00000000 |

```
void LCD_init(void){
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO
    GPIOD_PDDR = 0xFF;        // make PTD7-0 as output pins
```

| General Purpose Input/Output (PTD) | |
|---|---|
| GPIOD_PDOR | 0x00000000 |
| GPIOD_PSOR | non-readable |
| GPIOD_PCOR | non-readable |
| GPIOD_PTOR | non-readable |
| GPIOD_PDIR | 0x00000000 |
| GPIOD_PDDR | 0x000000ff |

General Purpose Input/Output (PTE)

```
SIM_SCGC5 |= 0x0200;      // enable clock to Port A
PORTA_PCR2 = 0x100;       // make PTA2 pin as GPIO
PORTA_PCR4 = 0x100;       // make PTA4 pin as GPIO
PORTA_PCR5 = 0x100;       // make PTA5 pin as GPIO
GPIOA_PDDR |= 0x34;       // make PTA5, 4, 2 as output pins
```

| Pin Control and Interrupts (PORTA) | |
|---|---|
| PORTA_PCR0 | 0x00000706 |
| PORTA_PCR1 | 0x00000005 |
| PORTA_PCR2 | 0x00000105 |
| PORTA_PCR3 | 0x00000703 |
| PORTA_PCR4 | 0x00000105 |
| PORTA_PCR5 | 0x00000105 |

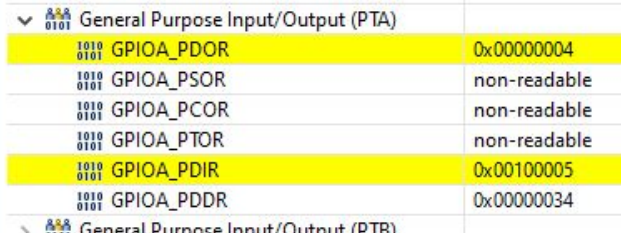
This is how the registers changes for the LCD_command.

```
}
void LCD_command(unsigned char command){
    GPIOA_PCOR = RS|RW;          /*RS=0,R/W=0*/
    GPIOD_PDOR = command;
    GPIOA_PSOR = EN;         //pulse E
    delayMs(0);
    GPIOA_PCOR = EN;

    if (command < 4)
    delayMs(4);          /* command 1 and 2 needs up to 1.64ms
    else
    delayMs(1);          /* all others 40 us */
}
```

| General Purpose Input/Output (PTA) | |
|---|---|
| GPIOA_PDOR | 0x00000020 |
| GPIOA_PSOR | non-readable |
| GPIOA_PCOR | non-readable |
| GPIOA_PTOR | non-readable |
| GPIOA_PDIR | 0x00100021 |
| GPIOA_PDDR | 0x00000034 |

This is the function that writes the data on the LCD and how the registers changed

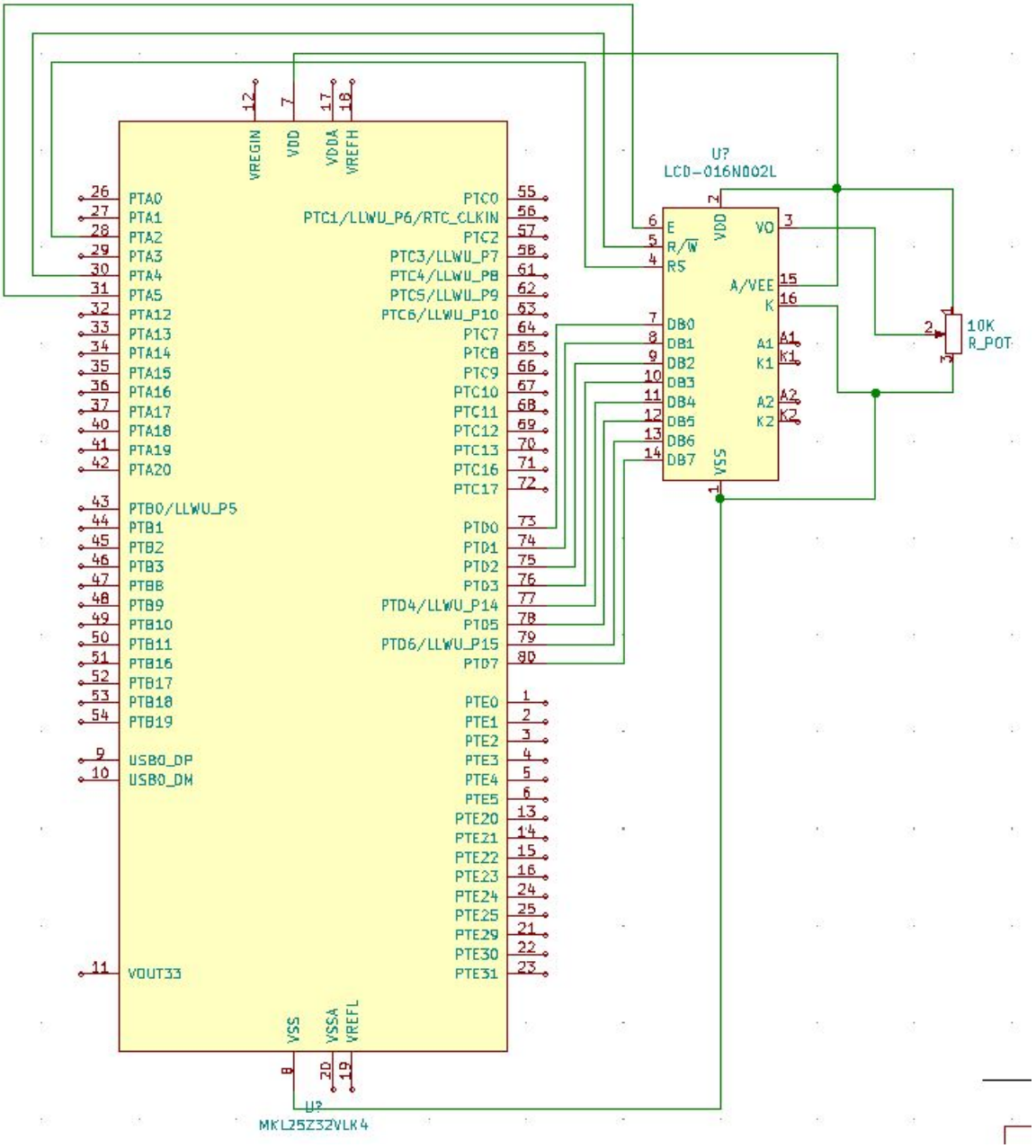
```
void LCD_data(unsigned char data){
    GPIOA_PSOR = RS;          // RS = 1
    GPIOA_PCOR = RW;          // RW = 0
    GPIOD_PDOR = data;
    GPIOA_PSOR = EN;          //pulse E
    delayMs(0);
    GPIOA_PCOR = EN;
    delayMs(1);
}
```



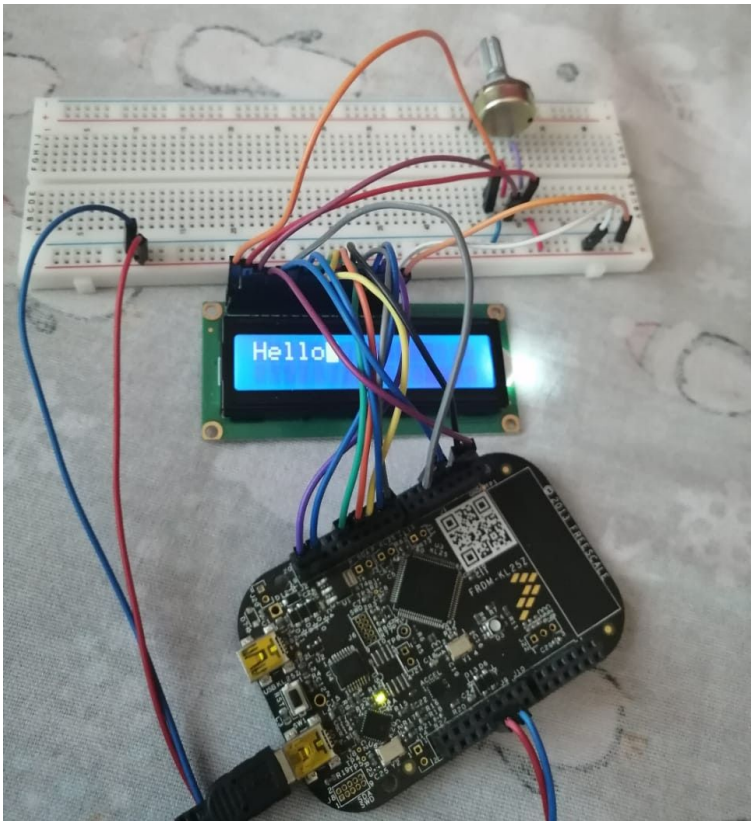

After that we have the rest of the main is similar to the last picture.

```
int main(void)
{
    SIM_SCGC5 |= 0x00000001;
    delayMs(500);
    LCD_init();
    for(;;) {
        LCD_command(1);         //clear the display
        delayMs(500);
        LCD_command(0x80);      // set cursor at first line
        delayMs(1);
        LCD_data('H');
        LCD_data('e');
        LCD_data('l');
        LCD_data('l');
        LCD_data('o');
        delayMs(500);
    }

    return 0;
}
```

**Schematic:**


U?
LCD-016N002L
VDD
VO
E
R/W
RS
A/VEE
K
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A1
K1
A2
K2
VSS
10K
R_POT
VREGIN
VDD
VDDA
VREFH
PTA0
PTA1
PTA2
PTA3
PTA4
PTA5
PTA12
PTA13
PTA14
PTA15
PTA16
PTA17
PTA18
PTA19
PTA20
PTB0/LLWU_P5
PTB1
PTB2
PTB3
PTB8
PTB9
PTB10
PTB11
PTB16
PTB17
PTB18
PTB19
USB0_DP
USB0_DM
VOUT33
PTC0
PTC1/LLWU_P6/RTC_CLKIN
PTC2
PTC3/LLWU_P7
PTC4/LLWU_P8
PTC5/LLWU_P9
PTC6/LLWU_P10
PTC7
PTC8
PTC9
PTC10
PTC11
PTC12
PTC13
PTC16
PTC17
PTD0
PTD1
PTD2
PTD3
PTD4/LLWU_P14
PTD5
PTD6/LLWU_P15
PTD7
PTE0
PTE1
PTE2
PTE3
PTE4
PTE5
PTE20
PTE21
PTE22
PTE23
PTE24
PTE25
PTE29
PTE30
PTE31
VSS
VSSA
VREFL
U?
MKL25Z32VLK4

**Picture:**

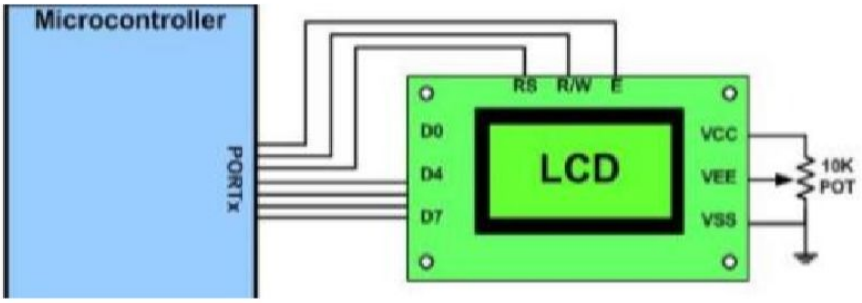



**Video:**

**https://youtu.be/Aw_Foc4pX-0**

## Part II.

**Develop the 4-bit version of the program to save pins in your design.**




## Code:

```
#include "MKL25z4.h"
#include "derivative.h" /* include peripheral declarations */

#define RS 1    /* BIT0 mask */
#define RW 2    /* BIT1 mask */
#define EN 4    /* BIT2 mask */

void delayMs(int n);
void delayUs(int n);
void LCD_nibble_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void LCD_init(void);
```

```
void LCD_init(void)
{
    SIM_SCGC5 |= 0x1000;     // enable clock to Port D
    PORTD_PCR0 = 0x100;      // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;      // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;      // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;      // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;      // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;      // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;      // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;      // make PTD7 pin as GPIO

    GPIOD_PDDR |= 0xF7;          // make PTD7-4, 2, 1, 0 as output pins
    delayMs(30);                 // initialization sequence
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x20, 0);  // use 4-bit data mode
    delayMs(1);
    LCD_command(0x28);           // set 4-bit data, 2-line, 5x7 font
    LCD_command(0x06);           // move cursor right
    LCD_command(0x01);           // clear screen, move cursor to home
    LCD_command(0x0F);           // turn on display, cursor blinking
}
```

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;       // clear lower nibble for control
    control &= 0x0F;    // clear upper nibble for data
    GPIOD_PDOR = data | control;       // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;  // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}


void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0);   // upper nibble first
    LCD_nibble_write(command << 4, 0);     // then lower nibble
    if (command < 4)
        delayMs(4);         // commands 1 and 2 need up to 1.64ms
    else
        delayMs(1);         // all others 40 us
}

void LCD_data(unsigned char data)
{
    LCD_nibble_write(data & 0xF0, RS);    // upper nibble first
    LCD_nibble_write(data << 4, RS);      // then lower nibble
    delayMs(1);
}
```

```
void delayMs(int n) {
    int i;
    int j;
    for(i = 0 ; i < n; i++){
        for(j = 0 ; j < 7000; j++) { }
    }
}

int main(void)
{
    LCD_init();
    for(;;)
    {
     LCD_command(1);         // clear display
     delayMs(500);
     LCD_command(0x85);      // set cursor at the middle
     LCD_data('H');          // write the word
     LCD_data('e');
     LCD_data('l');
     LCD_data('l');
     LCD_data('o');
     delayMs(500);
    }
}
```

## Registers:

Here is where the program starts.

```
int main(void)
{
    LCD_init();
    for(;;)
    {
     LCD_command(1);          // clear display
     delayMs(500);
     LCD_command(0x85);       // set cursor at the middle
     LCD_data('H');           // write the word
     LCD_data('e');
     LCD_data('l');
     LCD_data('l');
     LCD_data('o');
     delayMs(500);
    }
}
```

Here are the registers that changes where the LCD is initialized.

```
void LCD_init(void)
{
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
```

| | |
|---|---|
| SIM_SDID | 0x25152486 |
| SIM_SCGC4 | 0xf0000030 |
| SIM_SCGC5 | 0x00001180 |
| SIM_SCGC6 | 0x00000001 |
| SIM_SCGC7 | 0x00000100 |

```
void LCD_init(void)
{
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO
```

| Pin Control and Interrupts (PORTD) | |
|---|---|
| PORTD_PCR0 | 0x00000105 |
| PORTD_PCR1 | 0x00000105 |
| PORTD_PCR2 | 0x00000105 |
| PORTD_PCR3 | 0x00000105 |
| PORTD_PCR4 | 0x00000101 |
| PORTD_PCR5 | 0x00000101 |
| PORTD_PCR6 | 0x00000101 |
| PORTD_PCR7 | 0x00000101 |
| PORTD_PCR8 | 0x00000000 |

```
void LCD_init(void)
{
    SIM_SCGC5 |= 0x1000;      // enable clock to Port D
    PORTD_PCR0 = 0x100;       // make PTD0 pin as GPIO
    PORTD_PCR1 = 0x100;       // make PTD1 pin as GPIO
    PORTD_PCR2 = 0x100;       // make PTD2 pin as GPIO
    PORTD_PCR3 = 0x100;       // make PTD3 pin as GPIO
    PORTD_PCR4 = 0x100;       // make PTD4 pin as GPIO
    PORTD_PCR5 = 0x100;       // make PTD5 pin as GPIO
    PORTD_PCR6 = 0x100;       // make PTD6 pin as GPIO
    PORTD_PCR7 = 0x100;       // make PTD7 pin as GPIO

    GPIOD_PDDR |= 0xF7;       // make PTD7-4, 2, 1, 0 as output pins
    delayMs(30);              // initialization sequence
    LCD_nibble_write(0x30, 0);
```

| General Purpose Input/Output (PTD) | |
|---|---|
| GPIOD_PDOR | 0x00000000 |
| GPIOD_PSOR | non-readable |
| GPIOD_PCOR | non-readable |
| GPIOD_PTOR | non-readable |
| GPIOD_PDIR | 0x00000008 |
| GPIOD_PDDR | 0x000000f7 |

Here we jump to the nibble_write function.

```
    GPIOD_PDDR |= 0xF7;          //
    delayMs(30);                 //
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
```

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;       // clear lower nibble for control
    control &= 0x0F;    // clear upper nibble for data
    GPIOD_PDOR = data | control;       // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;  // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}
```

| General Purpose Input/Output (PTD) | |
|---|---|
| GPIOD_PDOR | 0x00000030 |
| GPIOD_PSOR | non-readable |
| GPIOD_PCOR | non-readable |
| GPIOD_PTOR | non-readable |
| GPIOD_PDIR | 0x00000038 |
| GPIOD_PDDR | 0x000000f7 |

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;       // clear lower nibble for control
    control &= 0x0F;    // clear upper nibble for data
    GPIOD_PDOR = data | control;       // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;  // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}
```

| General Purpose Input/Output (PTD) | |
|---|---|
| GPIOD_PDOR | 0x00000034 |
| GPIOD_PSOR | non-readable |
| GPIOD_PCOR | non-readable |
| GPIOD_PTOR | non-readable |
| GPIOD_PDIR | 0x0000003c |
| GPIOD_PDDR | 0x000000f7 |

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;       // clear lower nibble for control
    control &= 0x0F;    // clear upper nibble for data
    GPIOD_PDOR = data | control;       // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;  // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}
```

| General Purpose Input/Output (PTD) | |
|---|---|
| GPIOD_PDOR | 0x00000030 |
| GPIOD_PSOR | non-readable |
| GPIOD_PCOR | non-readable |
| GPIOD_PTOR | non-readable |
| GPIOD_PDIR | 0x00000038 |
| GPIOD_PDDR | 0x000000f7 |

```
void LCD_nibble_write(unsigned char data, unsigned char control)
{
    data &= 0xF0;       // clear lower nibble for control
    control &= 0x0F;    // clear upper nibble for data
    GPIOD_PDOR = data | control;       // RS = 0, R/W = 0
    GPIOD_PDOR = data | control | EN;  // pulse E
    delayMs(0);
    GPIOD_PDOR = data;
    GPIOD_PDOR = 0;
}
```

| General Purpose Input/Output (PTD) | |
|---|---|
| GPIOD_PDOR | 0x00000000 |
| GPIOD_PSOR | non-readable |
| GPIOD_PCOR | non-readable |
| GPIOD_PTOR | non-readable |
| GPIOD_PDIR | 0x00000008 |
| GPIOD_PDDR | 0x000000f7 |

The next function to be analyzed is LCD_command. The delay in this case it is not used with the flags of the microcontroller.

```
int main(void)
{
    LCD_init();
    for(;;)
    {
     LCD_command(1);          // clear display
     delayMs(500);
     LCD_command(0x85);       // set cursor at the middle
     LCD_data('H');           // write the word
     LCD_data('e');
     LCD_data('l');
     LCD_data('l');
     LCD_data('o');
     delayMs(500);
    }
}
```

This function is a compilation of the function nibble_write that we already see before.

```
void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0);   // upper nibble first
    LCD_nibble_write(command << 4, 0);     // then lower nibble
    if (command < 4)
        delayMs(4);         // commands 1 and 2 need up to 1.64ms
    else
        delayMs(1);         // all others 40 us
}
```

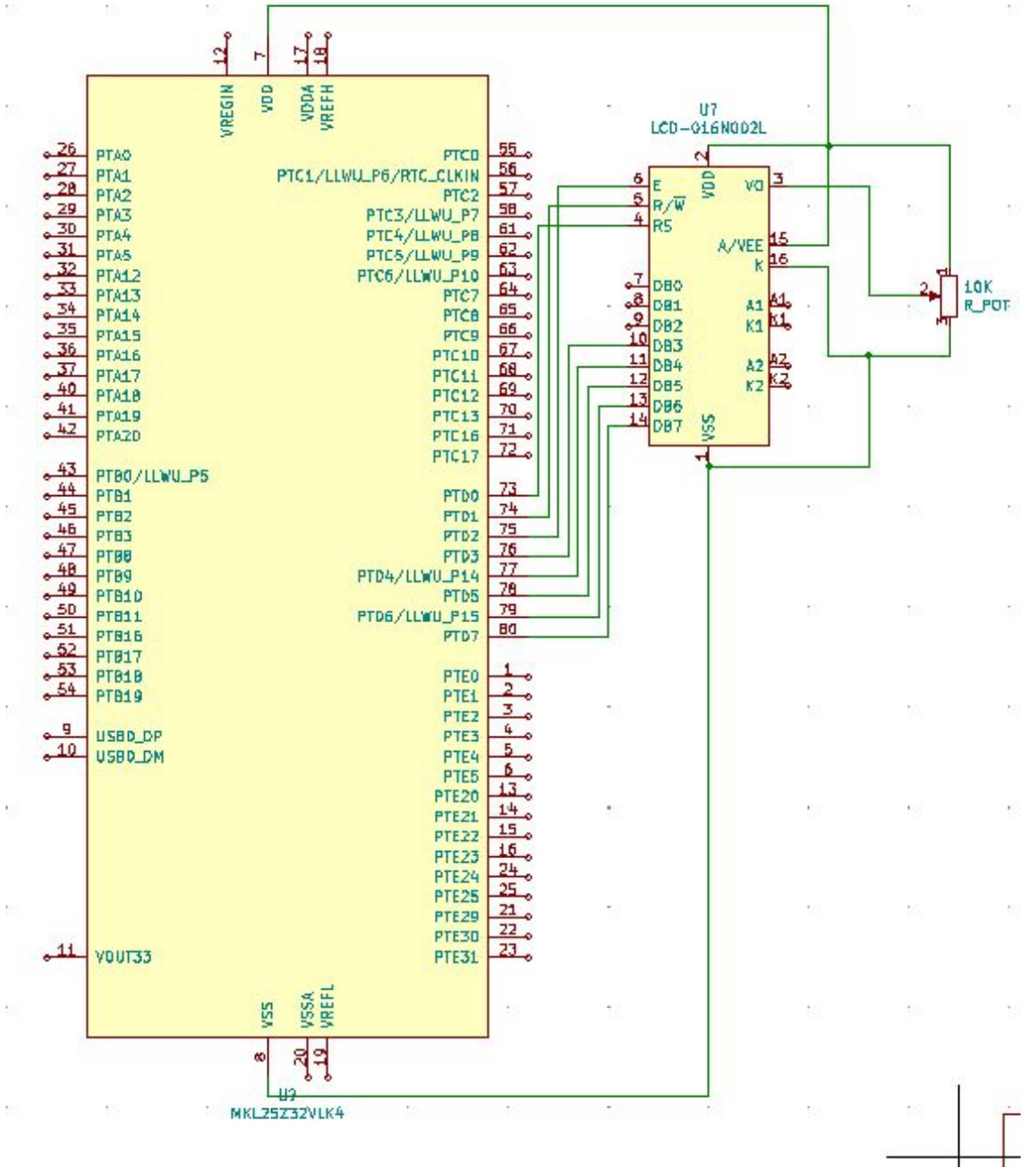
After that it goes to the LCD_data function.

```
int main(void)
{
    LCD_init();
    for(;;)
    {
     LCD_command(1);         // clear display
     delayMs(500);
     LCD_command(0x85);      // set cursor at the middle
     LCD_data('H');          // write the word
     LCD_data('e');
     LCD_data('l');
     LCD_data('l');
     LCD_data('o');
     delayMs(500);
    }
}
```

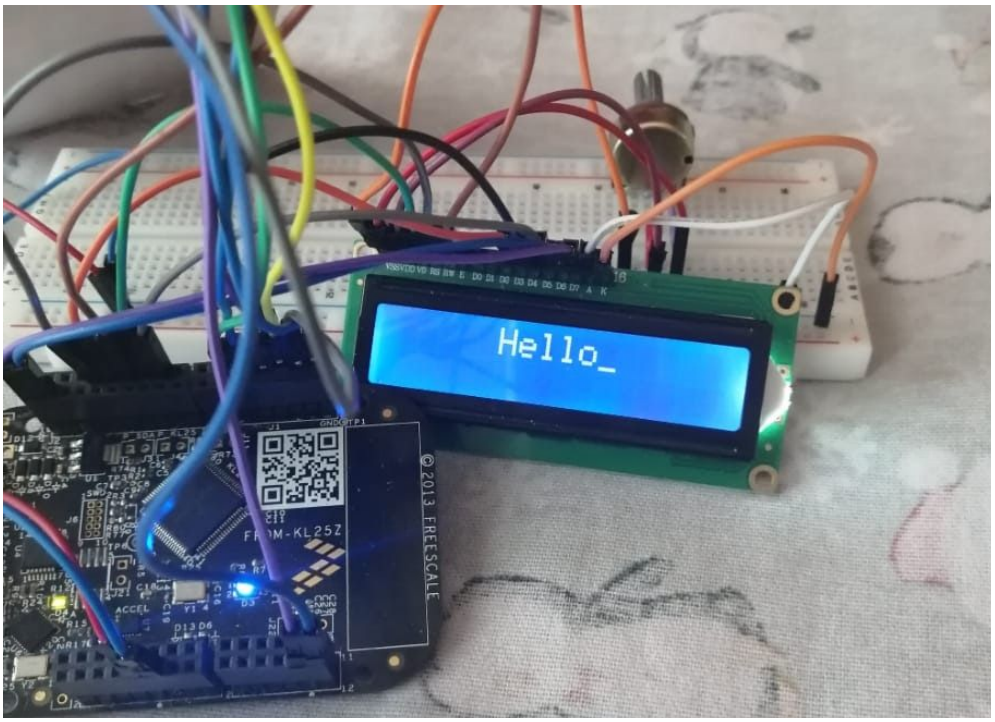
And that function is also the nibble_write that we described before.

```
void LCD_data(unsigned char data)
{
    LCD_nibble_write(data & 0xF0, RS);     // upper nibble first
    LCD_nibble_write(data << 4, RS);       // then lower nibble
    delayMs(1);
}
```

## Schematic:


U7
LCD-G16N002L
E
R/W
RS
VDD
VO
A/VEE
K
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
VSS
A1
K1
A2
K2
10K
R_POT
VREGIN
VDD
VDDA
VREFH
PTA0
PTA1
PTA2
PTA3
PTA4
PTA5
PTA12
PTA13
PTA14
PTA15
PTA16
PTA17
PTA18
PTA19
PTA20
PTB0/LLWU_P5
PTB1
PTB2
PTB3
PTB8
PTB9
PTB10
PTB11
PTB16
PTB17
PTB18
PTB19
USB0_DP
USB0_DM
VOUT33
PTC0
PTC1/LLWU_P6/RTC_CLKIN
PTC2
PTC3/LLWU_P7
PTC4/LLWU_P8
PTC5/LLWU_P9
PTC6/LLWU_P10
PTC7
PTC8
PTC9
PTC10
PTC11
PTC12
PTC13
PTC16
PTC17
PTD0
PTD1
PTD2
PTD3
PTD4/LLWU_P14
PTD5
PTD6/LLWU_P15
PTD7
PTE0
PTE1
PTE2
PTE3
PTE4
PTE5
PTE20
PTE21
PTE22
PTE23
PTE24
PTE25
PTE29
PTE30
PTE31
VSS
VSSA
VREFL
U9
MKL25Z32VLK4

**Picture:**




**Video:**

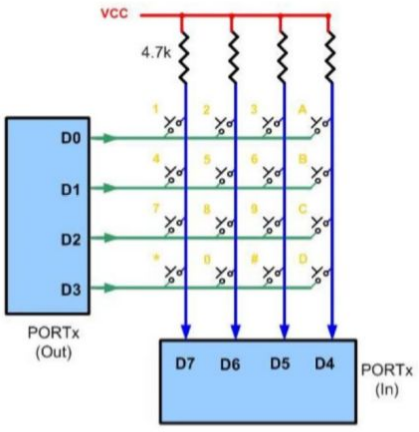
**https://youtu.be/bDn88pRJ4QI**

## Part III.

**To reduce the microcontroller I/O pin usage, keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 × 8 matrix of 64 keys can be connected to a microprocessor. When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns**

**The figure shows a 4 × 4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port. All the input pins have pull-up resistor connected. If no key has been pressed, reading the input port will yield 1s for all columns.**

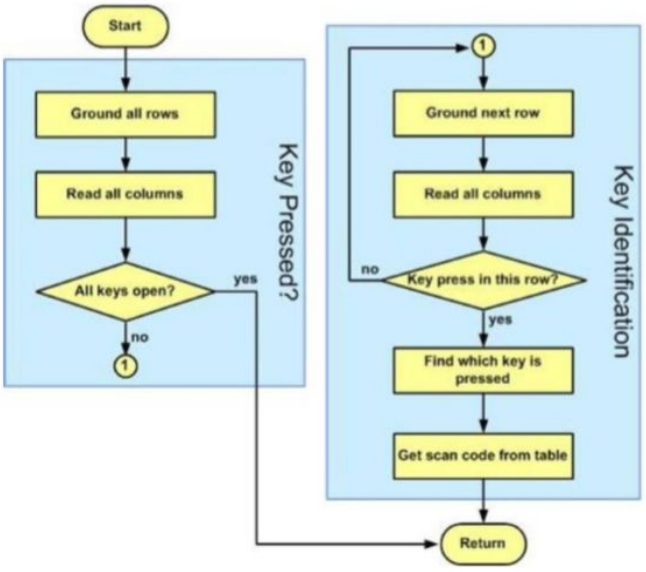
**If all the rows are driven low and a key is pressed, the column will read back 0. It is the function of the microprocessor to scan the keyboard continuously to detect and identify the key pressed.**




**To detect the key pressed, the microprocessor drives all rows low then it reads the columns. If the data read from the columns is D7–D4 = 1111, no key has been pressed and the process continues until a key press is detected However, if one of the column bits has a zero, this means that a key was pressed. For**

**example, if D7–D4= 1101, this means that a key in the D5 column has been pressed**

**After a key press is detected, the microprocessor will go through the process of identifying the key. Starting from the top row, the microprocessor drives one row low at a time; then it reads the columns. If the data read is all 1s, no key in that row is pressed and the process is moved to the next row. It drives the next row low, reads the columns, and checks for any zero. This process continues until a row is identified with a zero in one of the columns. The next task is to find out which column the pressed key belongs to. This should be easy since each column is connected to a separate input pin.**




**In this part, you have to implement a mechanism by which the microprocessor scans and identifies the key turning on the LED of different color depending on the column.**

1st column : RED

2nd column: GREEN

3rd column: Purple.

## Code:

Some internal declarations and including the libraries that are required.

```
#include "MKL25z4.h"
#include "derivative.h" /* include peripheral declarations */

void delayMs(int n);
void delayUs(int n);

void keypad_init(void);
char keypad_getkey(void);

void led_init(void);
void led_set(int value);
```

Here is where all the ports of the microcontrollers are internal activated and configured in order to use them.

```
void keypad_init(void){
    SIM_SCGC5 |= 0x0800;     //enable clk to port c

    PORTC_PCR0 = 0x103;     //PTC0 as GPIO and enable pullup
    PORTC_PCR1 = 0x103;     //PTC1 as GPIO and enable pullup
    PORTC_PCR2 = 0x103;     //PTC2 as GPIO and enable pullup
    PORTC_PCR3 = 0x103;     //PTC3 as GPIO and enable pullup
    PORTC_PCR4 = 0x103;     //PTC4 as GPIO and enable pullup
    PORTC_PCR5 = 0x103;     //PTC5 as GPIO and enable pullup
    PORTC_PCR6 = 0x103;     //PTC6 as GPIO and enable pullup
    PORTC_PCR7 = 0x103;     //PTC7 as GPIO and enable pullup
    GPIOC_PDDR = 0x0F;      //make PTC7-0 as input pins
}
```

This function return the exact position from 1 to 16 from the key that was pressed.

```
char keypad_getkey(void){

    int col, row;
    const char row_select[] = {0x01, 0x02, 0x04, 0x08};

    GPIOC_PDDR = 0x0F;  //enable al rows
    GPIOC_PCOR = 0x0F;
    delayUs(2);         //wait for signal

    col = 0xF00 & GPIOC_PDIR;   //read all columns

    GPIOC_PDDR = 0;             //disable all rows

    if (col == 0xF0)            //no key pressed;
    return 0;

    for (row = 0; row<4; row++) //finds out which key was pressed
    {
        GPIOC_PDDR = 0;                 //disable all rows
        GPIOC_PDDR |= row_select[row];  //enable one row
        GPIOC_PCOR = row_select[row];   //drive the active row low
        delayUs(2);                     //wait for signal to settle
        col = GPIOC_PDIR & 0xF0;        //read all columns
        if (col != 0xF0) break;         //if one of the inputs is low some key is pressed
    }

    GPIOC_PDDR = 0; //disable all rows
```

```
    if (row == 4)   //no key was pressed
    return 0;

    if (col == 0xE0) return row * 4 + 1; //key in column 1
    if (col == 0xD0) return row * 4 + 2; //key in column 2
    if (col == 0xB0) return row * 4 + 3; //key in column 3
    if (col == 0x70) return row * 4 + 4; //key in column 4
    return 0;                            //other information received
}
```

Then, here it is the declaration of the internal ports to use the color of the RGB LED and the internal enable of the clocks to this ports.

```
void led_init(void){

    SIM_SCGC5 |= 0x400;     //enable clk to port B
    SIM_SCGC5 |= 0x1000;    //enable clk to port D

    //RED LED
    PORTB_PCR18 = 0x100;    //make PTB18 as GPIO
    GPIOB_PDDR |= 0x40000;  //make PTB18 as output pin
    GPIOB_PSOR |= 0x40000;  //turn off red LED

    //GREEN LED
    PORTB_PCR19 = 0x100;    //make PTB19 as GPIO
    GPIOB_PDDR |= 0x80000;  //make PTB19 as output pin
    GPIOB_PSOR |= 0x80000;  //turn off green LED

    //BLUE LED
    PORTD_PCR1 = 0x100;     //make PD1 as GPIO
    GPIOD_PDDR |= 0x02;     //make PD1 as output pin
    GPIOD_PSOR |= 0x02;     //turn off blue LED
}
```

Here is where depending of the columns the LED turns on of a color.

```
void led_set(int value)
{
    //RED LED
    if (value == 1 || value == 5|| value == 9|| value == 13|| value == 3|| value == 7|| value == 11|| value == 15)
    GPIOB_PCOR |= 0x40000;
    else
    GPIOB_PSOR |= 0x40000;

    //GREEN LED
    if (value == 2|| value == 6|| value == 10|| value == 14)
    GPIOB_PCOR |= 0x80000;
    else
    GPIOB_PSOR |= 0x80000;

    //JUST FOR FUN ADDING COLOR PURPLE ON VALUES 3 7 11 15, THATS WHY RED AND BLUE TURN ON AT THAT POSITION

    //BLUE LED
    if (value == 4|| value == 8|| value == 12|| value == 16 || value == 3|| value == 7|| value == 11|| value == 15)
    GPIOD_PCOR |= 0x02;
    else
    GPIOD_PSOR |= 0x02;
}
```

Here we have some basic delay functions.

```
void delayUs(int n){
    int i,j;
    for(i = 0 ; i < n; i++) {
        for(j = 0; j < 5; j++) ;
    }
}

void delayMs(int n) {
    int i;
    int j;
    for(i = 0 ; i < n; i++){
        for(j = 0 ; j < 7000; j++) { }
    }
}
```

Here we have the main which is the one executed.

```
int main(void)
{
    unsigned char key;
    keypad_init();
    led_init();


    while(1)
    {
        key = keypad_getkey();  //identify the key pressed
        led_set(key);           //turning on the LEDs according to the key code
    }
}
```

## Registers:

It starts with the main.

```
int main(void)
{
    unsigned char key;
    keypad_init();
    led_init();


    while(1)
    {
        key = keypad_getkey();  //identify the key pressed
        led_set(key);           //turning on the LEDs according to the key code
    }
}
```

Here is where the clock is set to the Port C

```
void keypad_init(void){
    SIM_SCGC5 |= 0x0800;    //enable clk to port c

    PORTC_PCR0 = 0x103;    //PTC0 as GPIO and enable pullup
    PORTC_PCR1 = 0x103;    //PTC1 as GPIO and enable pullup
```

| | |
|---|---|
| SIM_SDID | 0x25152486 |
| SIM_SCGC4 | 0xf0000030 |
| SIM_SCGC5 | 0x00000980 |
| SIM_SCGC6 | 0x00000001 |

Here we have from PORTC_PCR0 to PORTC_PCR7 as GPIO.

```
void keypad_init(void){
    SIM_SCGC5 |= 0x0800;    //enable clk to port c

    PORTC_PCR0 = 0x103;    //PTC0 as GPIO and enable pullup
    PORTC_PCR1 = 0x103;    //PTC1 as GPIO and enable pullup
    PORTC_PCR2 = 0x103;    //PTC2 as GPIO and enable pullup
    PORTC_PCR3 = 0x103;    //PTC3 as GPIO and enable pullup
    PORTC_PCR4 = 0x103;    //PTC4 as GPIO and enable pullup
    PORTC_PCR5 = 0x103;    //PTC5 as GPIO and enable pullup
    PORTC_PCR6 = 0x103;    //PTC6 as GPIO and enable pullup
    PORTC_PCR7 = 0x103;    //PTC7 as GPIO and enable pullup
    GPIOC_PDDR = 0x0F;     //make PTC7-0 as input pins
}
```

| Pin Control and Interrupts (PORTC) | |
|---|---|
| PORTC_PCR0 | 0x00000107 |
| PORTC_PCR1 | 0x00000107 |
| PORTC_PCR2 | 0x00000107 |
| PORTC_PCR3 | 0x00000103 |
| PORTC_PCR4 | 0x00000103 |
| PORTC_PCR5 | 0x00000103 |
| PORTC_PCR6 | 0x00000103 |
| PORTC_PCR7 | 0x00000103 |
| PORTC_PCR8 | 0x00000005 |
| PORTC_PCR9 | 0x00000005 |

```
void keypad_init(void){
    SIM_SCGC5 |= 0x0800;    //enable clk to port c

    PORTC_PCR0 = 0x103;    //PTC0 as GPIO and enable pullup
    PORTC_PCR1 = 0x103;    //PTC1 as GPIO and enable pullup
    PORTC_PCR2 = 0x103;    //PTC2 as GPIO and enable pullup
    PORTC_PCR3 = 0x103;    //PTC3 as GPIO and enable pullup
    PORTC_PCR4 = 0x103;    //PTC4 as GPIO and enable pullup
    PORTC_PCR5 = 0x103;    //PTC5 as GPIO and enable pullup
    PORTC_PCR6 = 0x103;    //PTC6 as GPIO and enable pullup
    PORTC_PCR7 = 0x103;    //PTC7 as GPIO and enable pullup
    GPIOC_PDDR = 0x0F;     //make PTC7-0 as input pins
}
```

| General Purpose Input/Output (PTC) | |
|---|---|
| GPIOC_PDOR | 0x00000000 |
| GPIOC_PSOR | non-readable |
| GPIOC_PCOR | non-readable |
| GPIOC_PTOR | non-readable |
| GPIOC_PDIR | 0x000000f0 |
| GPIOC_PDDR | 0x0000000f |

Then the next function is led_init.

```
int main(void)
{
    unsigned char key;
    keypad_init();
    led_init();


    while(1)
    {
        key = keypad_getkey();  //identify the key pressed
        led_set(key);           //turning on the LEDs according to the key code
    }
}
```

Here we have the enable of the clok to PORT B and D in order to used the LED.

```
void led_init(void){

    SIM_SCGC5 |= 0x400;     //enable clk to port B
    SIM_SCGC5 |= 0x1000;    //enable clk to port D

    //RED LED
    PORTB_PCR18 = 0x100;    //make PTB18 as GPIO
    GPIOB_PDDR |= 0x40000;  //make PTB18 as output pin
    GPIOB_PSOR |= 0x40000;  //turn off red LED

    //GREEN LED
    PORTB_PCR19 = 0x100;    //make PTB19 as GPIO
    GPIOB_PDDR |= 0x80000;  //make PTB19 as output pin
```

| | |
|---|---|
| SIM_SCGC4 | 0xf0000030 |
| SIM_SCGC5 | 0x00000d80 |
| SIM_SCGC6 | 0x00000001 |
| SIM_SCGC7 | 0x00000100 |

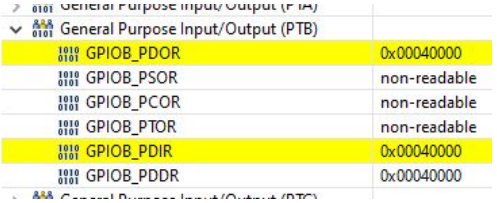
```
void led_init(void){

    SIM_SCGC5 |= 0x400;     //enable clk to port B
    SIM_SCGC5 |= 0x1000;    //enable clk to port D

    //RED LED
```

| | |
|---|---|
| SIM_SDID | 0x25152486 |
| SIM_SCGC4 | 0xf0000030 |
| SIM_SCGC5 | 0x00001d80 |
| SIM_SCGC6 | 0x00000001 |
| SIM_SCGC7 | 0x00000100 |

Here it is the initialization of the 3 colors of the LED as a GPIO and output pin.

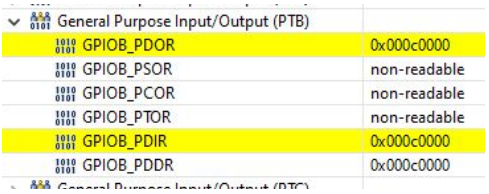
```
void led_init(void){

    SIM_SCGC5 |= 0x400;     //enable clk to port B
    SIM_SCGC5 |= 0x1000;    //enable clk to port D

    //RED LED
    PORTB_PCR18 = 0x100;    //make PTB18 as GPIO
    GPIOB_PDDR |= 0x40000;  //make PTB18 as output pin
    GPIOB_PSOR |= 0x40000;  //turn off red LED

    //GREEN LED
    PORTB_PCR19 = 0x100;    //make PTB19 as GPIO
    GPIOB_PDDR |= 0x80000;  //make PTB19 as output pin
    GPIOB_PSOR |= 0x80000;  //turn off green LED

    //BLUE LED
```


```
void led_init(void){

    SIM_SCGC5 |= 0x400;     //enable clk to port B
    SIM_SCGC5 |= 0x1000;    //enable clk to port D

    //RED LED
    PORTB_PCR18 = 0x100;    //make PTB18 as GPIO
    GPIOB_PDDR |= 0x40000;  //make PTB18 as output pin
    GPIOB_PSOR |= 0x40000;  //turn off red LED
```




```
    //GREEN LED
    PORTB_PCR19 = 0x100;    //make PTB19 as GPIO
    GPIOB_PDDR |= 0x80000;  //make PTB19 as output pin
    GPIOB_PSOR |= 0x80000;  //turn off green LED
```


```
//GREEN LED
PORTB_PCR19 = 0x100;    //make PTB19 as GPIO
GPIOB_PDDR |= 0x80000;  //make PTB19 as output pin
GPIOB_PSOR |= 0x80000;  //turn off green LED
```




Then we move on to the keypad_getkey function.

```
int main(void)
{
    unsigned char key;
    keypad_init();
    led_init();


    while(1)
    {
        key = keypad_getkey();  //identify the key pressed
        led_set(key);           //turning on the LEDs according to the key code
    }
}
```

```
char keypad_getkey(void){

    int col, row;
    const char row_select[] = {0x01, 0x02, 0x04, 0x08};

    GPIOC_PDDR = 0x0F;  //enable al rows
    GPIOC_PCOR = 0x0F;
    delayUs(2);         //wait for signal

    col = 0xF00 & GPIOC_PDIR;   //read all columns

    GPIOC_PDDR = 0;             //disable all rows

    if (col == 0xF0)            //no key pressed;
    return 0;
```

| General Purpose Input/Output (PTC) | |
|---|---|
| GPIOC_PDOR | 0x00000000 |
| GPIOC_PSOR | non-readable |
| GPIOC_PCOR | non-readable |
| GPIOC_PTOR | non-readable |
| GPIOC_PDIR | 0x000000e0 |
| GPIOC_PDDR | 0x0000000f |

```
char keypad_getkey(void){

    int col, row;
    const char row_select[] = {0x01, 0x02, 0x04, 0x08};

    GPIOC_PDDR = 0x0F;  //enable al rows
    GPIOC_PCOR = 0x0F;
    delayUs(2);         //wait for signal

    col = 0xF00 & GPIOC_PDIR;   //read all columns

    GPIOC_PDDR = 0;             //disable all rows

    if (col == 0xF0)            //no key pressed;
    return 0;
```

| General Purpose Input/Output (PTC) | |
|---|---|
| GPIOC_PDOR | 0x00000000 |
| GPIOC_PSOR | non-readable |
| GPIOC_PCOR | non-readable |
| GPIOC_PTOR | non-readable |
| GPIOC_PDIR | 0x000000ff |
| GPIOC_PDDR | 0x00000000 |

```
for (row = 0; row<4; row++) //finds out which key was pressed
{
    GPIOC_PDDR = 0;                 //disable all rows
    GPIOC_PDDR |= row_select[row];  //enable one row
    GPIOC_PCOR = row_select[row];   //drive the active row low
    delayUs(2);                     //wait for signal to settle
    col = GPIOC_PDIR & 0xF0;        //read all columns
    if (col != 0xF0) break;         //if one of the inputs is low some key is pressed
}

GPIOC_PDDR = 0; //disable all rows
```

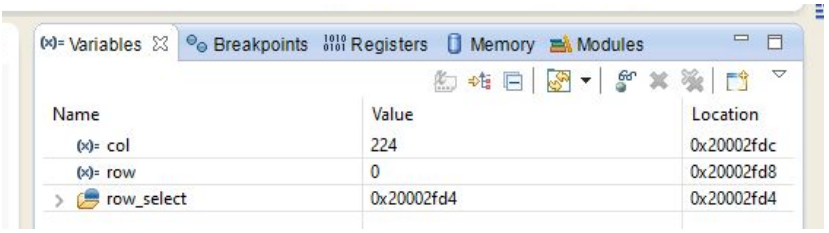
| General Purpose Input/Output (PTC) | |
|---|---|
| GPIOC_PDOR | 0x00000000 |
| GPIOC_PSOR | non-readable |
| GPIOC_PCOR | non-readable |
| GPIOC_PTOR | non-readable |
| GPIOC_PDIR | 0x000000ee |
| GPIOC_PDDR | 0x00000001 |

```
for (row = 0; row<4; row++) //finds out which key was pressed
{
    GPIOC_PDDR = 0;                 //disable all rows
    GPIOC_PDDR |= row_select[row];  //enable one row
    GPIOC_PCOR = row_select[row];   //drive the active row low
    delayUs(2);                     //wait for signal to settle
    col = GPIOC_PDIR & 0xF0;        //read all columns
    if (col != 0xF0) break;         //if one of the inputs is low
}

GPIOC_PDDR = 0; //disable all rows

if (row == 4)   //no key was pressed
return 0;
```

| General Purpose Input/Output (PTC) | |
|---|---|
| GPIOC_PDOR | 0x00000000 |
| GPIOC_PSOR | non-readable |
| GPIOC_PCOR | non-readable |
| GPIOC_PTOR | non-readable |
| GPIOC_PDIR | 0x000000ff |
| GPIOC_PDDR | 0x00000000 |

Here, since the key that is had been pressed was from the first column, it match with the first if.

```
    if (col == 0xE0) return row * 4 + 1; //key in column 1
    if (col == 0xD0) return row * 4 + 2; //key in column 2
    if (col == 0xB0) return row * 4 + 3; //key in column 3
    if (col == 0x70) return row * 4 + 4; //key in column 4
    return 0;                            //other information received
}
```




Then, since is the first  key, it match with the first if wich turn on the LED of the color red. And the last 2 else just maintain the other colors off.

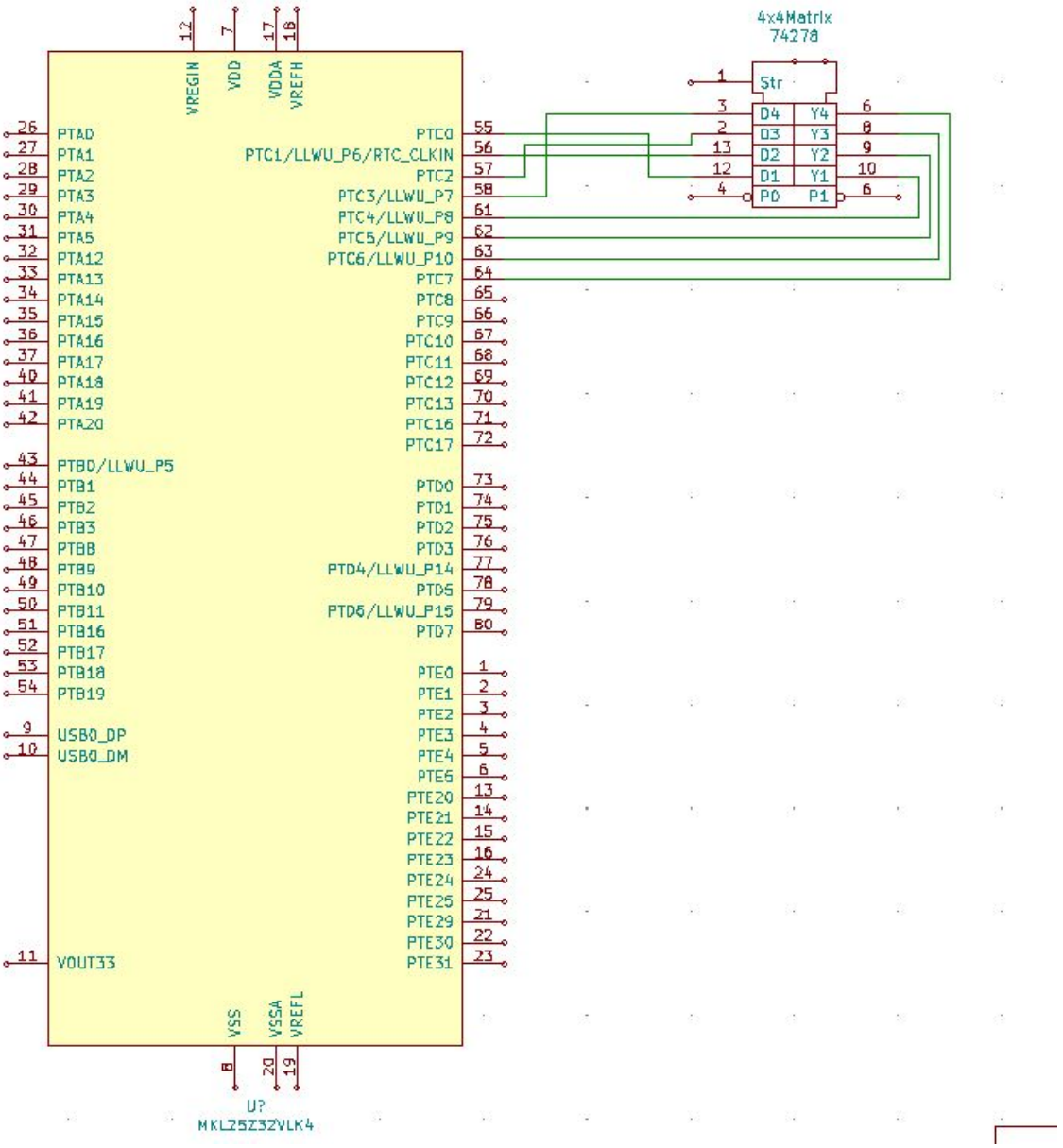
```
void led_set(int value)
{
    //RED LED
    if (value == 1 || value == 5|
    GPIOB_PCOR |= 0x40000;
    else
    GPIOB_PSOR |= 0x40000;
```

General Purpose Input/Output (PTB)

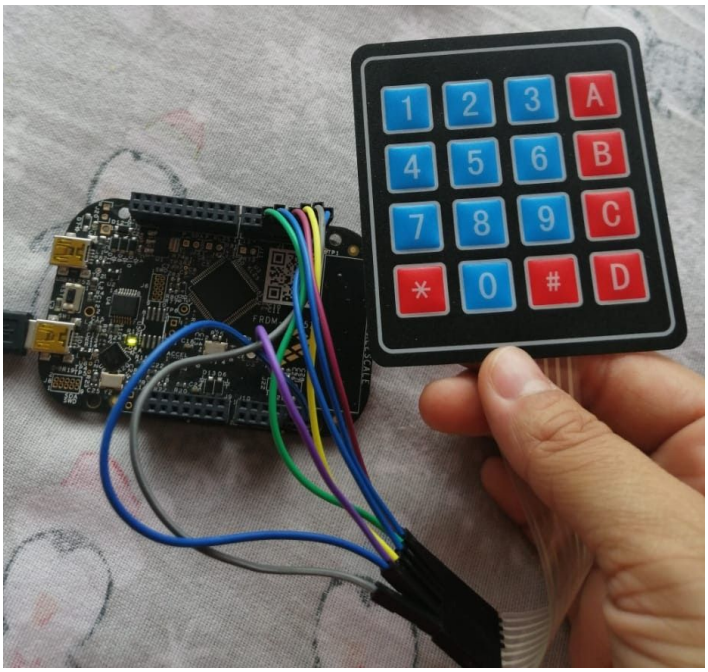
| | |
|---|---|
| GPIOB_PDOR | 0x00080000 |
| GPIOB_PSOR | non-readable |
| GPIOB_PCOR | non-readable |
| GPIOB_PTOR | non-readable |
| GPIOB_PDIR | 0x00080000 |
| GPIOB_PDDR | 0x000c0000 |

```
    //GREEN LED
    if (value == 2|| value == 6|| value == 10|| value == 14)
    GPIOB_PCOR |= 0x80000;
    else
    GPIOB_PSOR |= 0x80000;

    //JUST FOR FUN ADDING COLOR PURPLE ON VALUES 3 7 11 15, THATS

    //BLUE LED
    if (value == 4|| value == 8|| value == 12|| value == 16 || va
    GPIOD_PCOR |= 0x02;
    else
    GPIOD_PSOR |= 0x02;
}
```

## Schematic:


4x4Matrix
74278
Str
D4
D3
D2
D1
P0
Y4
Y3
Y2
Y1
P1
VREGIN
VDD
VDDA
VREFH
PTA0
PTA1
PTA2
PTA3
PTA4
PTA5
PTA12
PTA13
PTA14
PTA15
PTA16
PTA17
PTA18
PTA19
PTA20
PTB0/LLWU_P5
PTB1
PTB2
PTB3
PTB8
PTB9
PTB10
PTB11
PTB16
PTB17
PTB18
PTB19
USB0_DP
USB0_DM
VOUT33
PTC0
PTC1/LLWU_P6/RTC_CLKIN
PTC2
PTC3/LLWU_P7
PTC4/LLWU_P8
PTC5/LLWU_P9
PTC6/LLWU_P10
PTC7
PTC8
PTC9
PTC10
PTC11
PTC12
PTC13
PTC16
PTC17
PTD0
PTD1
PTD2
PTD3
PTD4/LLWU_P14
PTD5
PTD6/LLWU_P15
PTD7
PTE0
PTE1
PTE2
PTE3
PTE4
PTE5
PTE20
PTE21
PTE22
PTE23
PTE24
PTE25
PTE29
PTE30
PTE31
VSS
VSSA
VREFL
U?
MKL25Z32VLK4

**Picture:**




## Video:

https://youtu.be/87LKcFv5-jE

## Part IV.

**Contact debouncer. When a mechanical switch is closed or opened, the contacts do not make a clean transition instantaneously, rather the contacts open and close several times before they settle. This event is called contact bounce**


**So, it is possible when the program first detects a switch in the keypad is pressed but when interrogating which key is pressed, it would find no key**

**pressed. This is the reason we have a return 0 after checking all the rows.**

**Another problem manifested by contact bounce is that one key press may be recognized as multiple key presses by the program. Contact bounce also occurs when the switch is released. Because the switch contacts open and close several times before they settle, the program may detect a key press when the key is released.**

**For many applications, it is important that each key press is only recognized as one action. When you press a numeral key of a calculator, you expect to get only one digit. A contact bounce results in multiple digits entered with a single key press. A simple software solution is that when a transition of the contact state change is detected such as a key pressed or a key released, the software does a delay for about 10 – 20 ms to wait out the contact bounce. After the delay, the contacts should be settled and stable.**

**In this part, you have to develop a contact debouncer for the code in the previous part.**

## Code:

All the functions are the same as in the Part III. However, we add an extra function called: read_key in order to do de debouncing of the keypad. This functions only verifies that certain key is pressed.

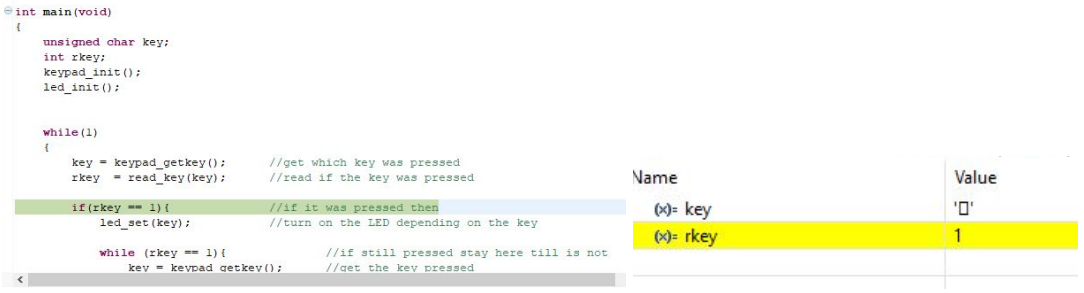
```
int read_key(int x){
    int key_press;

    if(x!=0)                //if a key was pressed return 1
    return key_press = 1;
    else                    //if there is no key pressed return 0
    return key_press = 0;


}
```

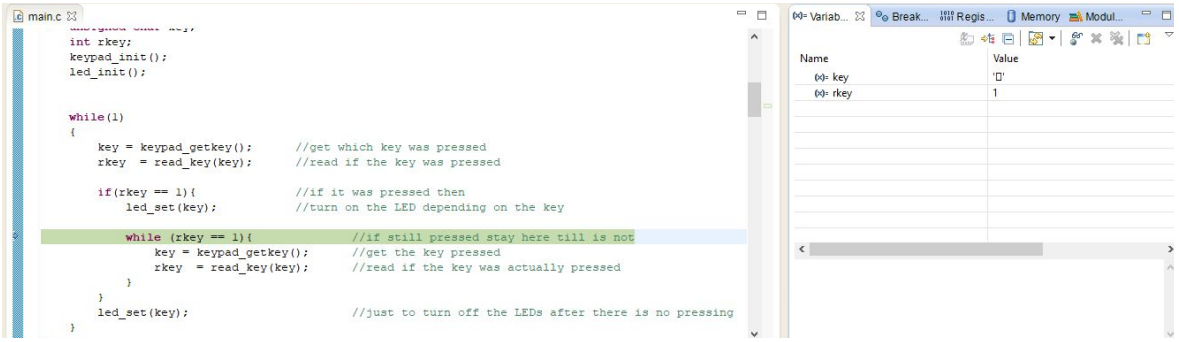
In the main we also add a loop in order to keep it there if the button is pressed and reduce the noise of miss signals.

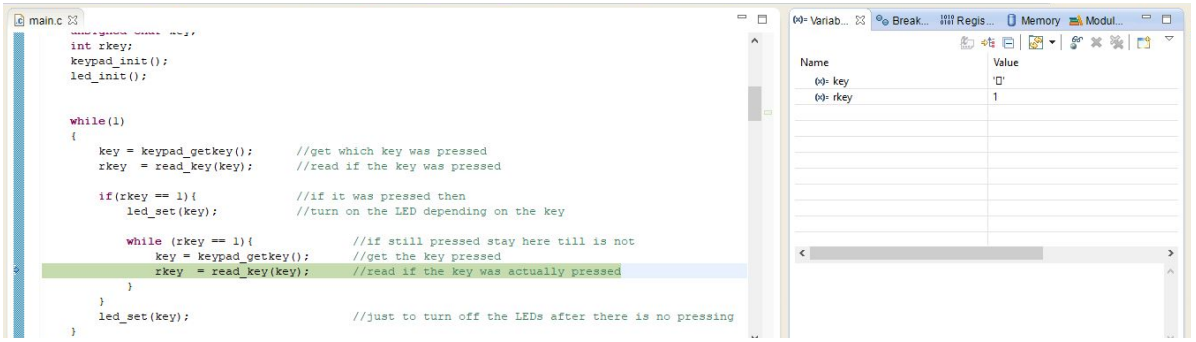
```
int main(void)
{
    unsigned char key;
    int rkey;
    keypad_init();
    led_init();


    while(1)
    {
        key = keypad_getkey();      //get which key was pressed
        rkey  = read_key(key);      //read if the key was pressed

        if(rkey == 1){              //if it was pressed then
            led_set(key);           //turn on the LED depending on the key

            while (rkey == 1){              //if still pressed stay here till is not
                key = keypad_getkey();      //get the key pressed
                rkey  = read_key(key);      //read if the key was actually pressed
            }
        }
        led_set(key);                       //just to turn off the LEDs after there is no pressing
    }
}
```

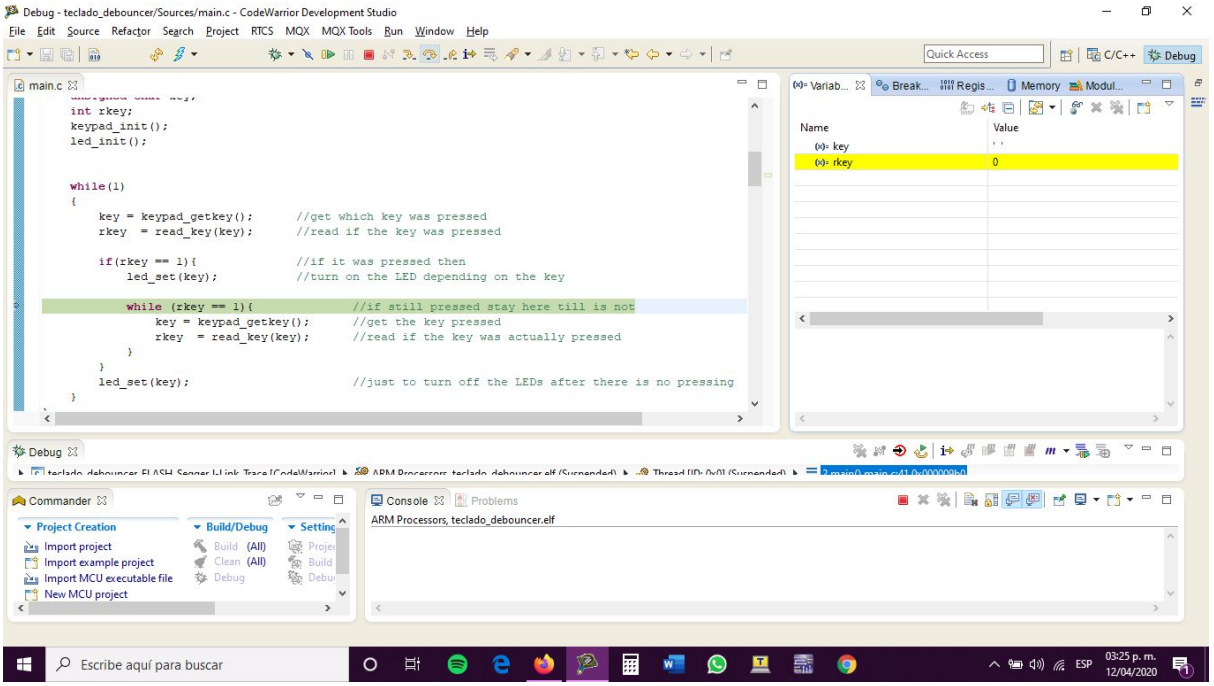
## Registers:

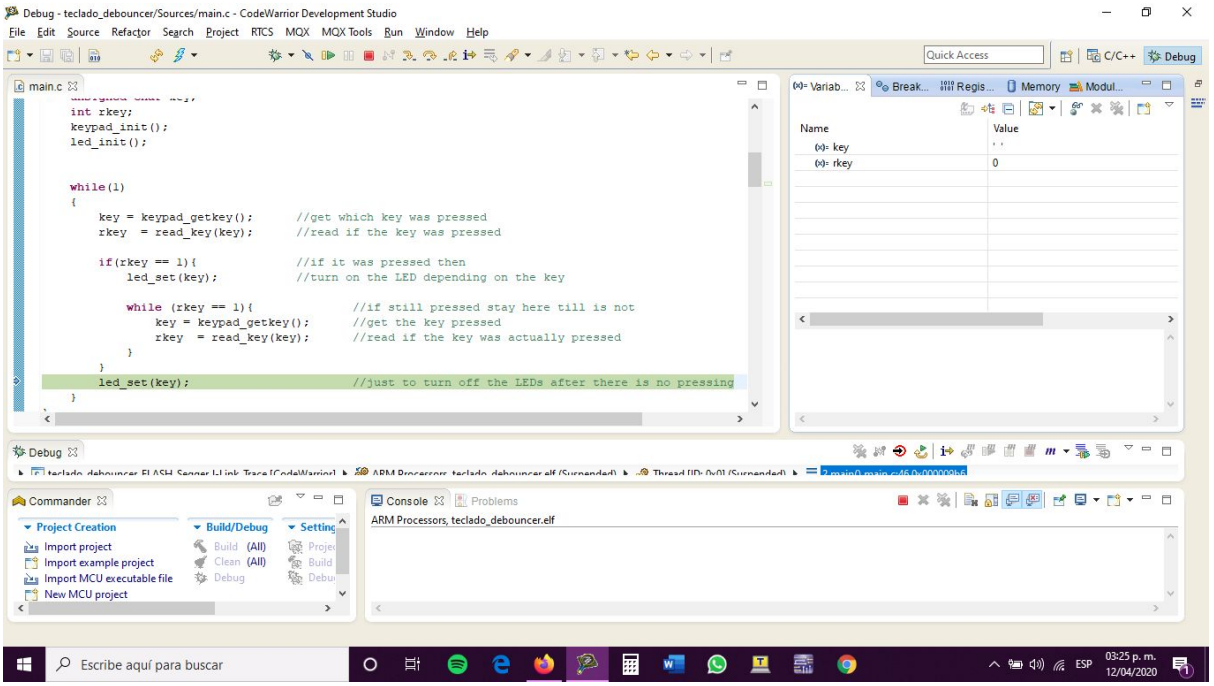
All the registers act the same as Part III. The only difference is on the main and how it keeps double checking that a key is pressed.




main.c
int rkey;
keypad_init();
led_init();
while(1)
{
key = keypad_getkey(); //get which key was pressed
rkey = read_key(key); //read if the key was pressed
if(rkey == 1){ //if it was pressed then
led_set(key); //turn on the LED depending on the key
while (rkey == 1){ //if still pressed stay here till is not
key = keypad_getkey(); //get the key pressed
rkey = read_key(key); //read if the key was actually pressed
}
}
led_set(key); //just to turn off the LEDs after there is no pressing
}
Variab...
Break...
Regis...
Memory
Modul...
Name
Value
key
'□'
rkey
1


Debug - teclado_debouncer/Sources/main.c - CodeWarrior Development Studio
File Edit Source Refactor Search Project RTCS MQX MQX Tools Run Window Help
Quick Access
C/C++
Debug
main.c
int rkey;
keypad_init();
led_init();
while(1)
{
key = keypad_getkey(); //get which key was pressed
rkey = read_key(key); //read if the key was pressed
if(rkey == 1){ //if it was pressed then
led_set(key); //turn on the LED depending on the key
while (rkey == 1){ //if still pressed stay here till is not
key = keypad_getkey(); //get the key pressed
rkey = read_key(key); //read if the key was actually pressed
}
}
led_set(key); //just to turn off the LEDs after there is no pressing
}
Variab...
Break...
Regis...
Memory
Modul...
Name
Value
key
' '
rkey
0
Debug
Commander
Project Creation
Import project
Import example project
Import MCU executable file
New MCU project
Build/Debug
Build (All)
Clean (All)
Debug
Console
Problems
ARM Processors, teclado_debouncer.elf
Escribe aquí para buscar
ESP
03:25 p. m.
12/04/2020




# Schematic, Picture and Video:

Same as Part III

## References:


***Debounce.*** **(2015). Arduino CC. Available at:**

**https://www.arduino.cc/en/Tutorial/Debounce**

**Rouse, M. (2005).** ***debouncing.*** **Available at:**

**https://whatis.techtarget.com/definition/debouncing**

**Williams, E. (2015).** ***EMBED WITH ELLIOT: DEBOUNCE YOUR NOISY BUTTONS, PART I.*** **Available at:**

**https://hackaday.com/2015/12/09/embed-with-elliot-debounce-your-noisy-buttons-part-i/**

**Zain, S. (May, 2017). LCD Interfacing with Microcontrollers. April, 2020, de The Engenineering Projects Available at:**

**https://www.theengineeringprojects.com/2017/05/lcd-interfacing-with-microcontrollers.html**