BERZIET UNIVERSITY

Faculty of Information Technology Computer Systems Engineering Department

COMPUTER DESIGN LAB

ENCS4110

LCD

Report NO.3/ Experiment _9

Prepared by:

Nour Naji– 1190270

Partners:

Rasha Dar Abu Zidan -1190547

Supervised by: Dr. Jamal Seyam

Teacher Assistant: Eng. Raha Zabadi

Section: 2

Date: 17 /11/ 2021

Abstract:

The goal of this experiment is to learn about LCD components, how to write and simulate a program in LCD, and how to use push buttons to control movement direction (shift left, right, clear, or advance to the second row).

Table of Contents

List of Figures

1. Theory

1.1 Alphanumeric LCD display

The most common type of monochrome LCD modules is known as Alphanumeric LCD Displays or a Character LCD Display. Alphanumeric LCD displays are widely used for the following reasons.

- These displays are easier to implement than a graphics unit.
- There are a large number of LCD suppliers that carry drop in equivalent displays.
- They have been a proven technology for many years.
- There are built-in standard configurations.

The model described in figure 1.1.1, is for its low price and great capabilities most frequently used in practice (LM016L LCD). It is based on the HD44780 microcontroller (Hitachi) and can display messages in two lines with 16 characters each

It displays all the character, the character is any letter, capital and non-capital, any number and the punctuation mark, such as period, comma and back slash. The character table built into the micro controller of the LCD can display 255 separate characters. Of course, there are different languages, so you can choose a character table that displays English, French, German and many other languages.

Most alphanumeric LCD displays cannot display Chinese and Japanese without the use of a larger character size. This means that a 16x2 that is built to display Chinese will be larger than a 16x2 that is manufactured to display English characters

Figure 1-1.1: LCD

1.2 : Function Description

1.2.1: Registers

There are two 8-bit registers, an instruction registers (IR), and a data register on the HD44780U. (DR). The IR stores display clear and cursor shift instruction codes as

well as address information for display data RAM (DDRAM) and character generator RAM (CGRAM).

The MPU is the only place where the IR can be written. The DR holds data that will be written into DDRAM or CGRAM and data that will be read from DDRAM or CGRAM momentarily. Internally, data written into the DR from the MPU is automatically written into DDRAM or CGRAM. When reading data from DDRAM or CGRAM, the DR is also used for data storage. When address information is written into the IR, data from DDRAM or CGRAM is read and then saved into the DR via an internal procedure.

1.2.2: Memory

There are three memory slots in the 162 LCD controller HD44780 for storing characters, numerals, and special symbols. The DDRAM (data display RAM) contains ASCII codes, the CGROM (character generating ROM) saves standard character patterns, and the CGRAM (character generating RAM) stores bespoke character patterns, totaling 8 in the 216 module.

1.2.2.1: Display Data RAM (DDRAM)

Display data RAM (DDRAM) is a type of RAM that stores data in the form of 8-bit character codes. It has a capacity of 808 bits, or 80 characters, in expanded mode. The unallocated memory in display data RAM (DDRAM) can be used as general data RAM.

1.2.2.2: Character Generator ROM (CGROM)

The character generator ROM, which is in charge of storing conventional character patterns, generates 58 or 510 dot patterns from 8-bit character codes. It has the ability to generate 208 588 and 32 510 dot character patterns.

1.2.2.3: Character Generator RAM (CGRAM)

The character generating RAM, which stores custom character patterns, only has 8 memory locations accessible for storing user defined characters with addresses 0x00 - 0x07.

1.3 LCD Display

Pins for connecting to the microcontroller run along one side of a compact printed circuit board. There are 14 pins with numbers on them in all (16 if the backlight is fitted in). In the figure below, their purpose is described:

Pin	Symbol	I/O	Description
1	VSS	—	Ground
2	VCC	—	+5V power supply
3	VEE	—	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 4/8-bit data bus
12	DB5	I/O	The 4/8-bit data bus
13	DB6	I/O	The 4/8-bit data bus
14	DB7	I/O	The 4/8-bit data bus

Figure1.3. 1:pin description of LCD

1.3.1 : LCD Screen Modes

The data bus is used to send commands and characters to the LCD, and it runs from D0-D7. A single 8-bit byte or two 4-bit nibbles can be used to transport data to and from the display.

The upper four data lines (D4-D7) are used solely in the second case. When using a microcontroller with a limited number of input/output pins, this 4-bit mode is quite useful.

1.3.2 Displaying Standard Character on LCD

Out of these three memory locations, DDRAM and CGROM are used to generate regular standard characters (ASCII characters). By using these three memory locations, a user can generate different character fonts and symbols on LCD display. A character font describes the shape and style of the character. Each shape of a character is designed by taking the number of pixels in mind. For example, in 16x2 LCD there are 16 segments available per single line. Each segment contains pixels in 5x7 or 5x10 matrix forms. For example, in 16×2 LCD there are 16 segments available per single line. Each segment contains pixels in 5×8 or 5×10 matrix forms.

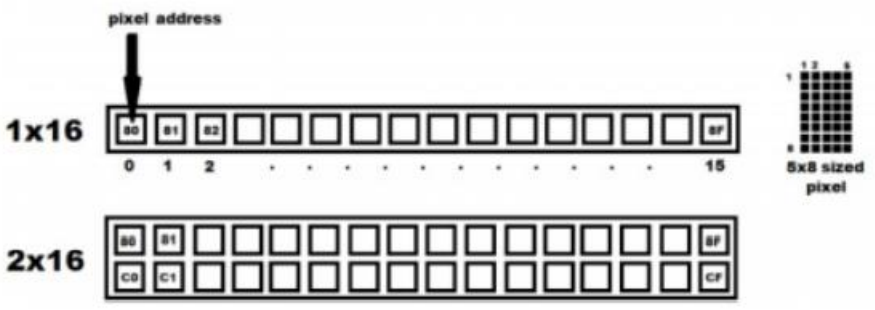

Figure1.3.2. 1: LCD pixels

The Display Data RAM (DDRAM) stores the ASCII code of a character which is sent by the microcontroller. Now the LCD controller (HD44780) maps the corresponding ASCII Code in DDRAM with CGROM address to bring the hexadecimal codes (character pattern) of that particular character. By using those hexadecimal codes the 5x7 matrix segment will light according to that character pattern to display corresponding character on it as shown in figure1.3.2.2.

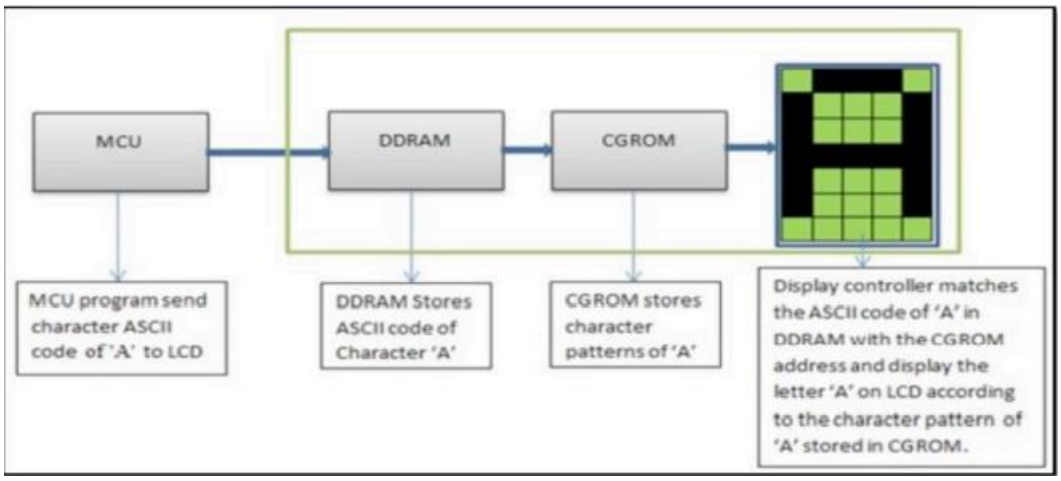

Figure1.3.2. 2: character generation on LCD

1.3.3 Displaying Custom Characters on LCD display

The display controller (HD44780) uses the CGRAM space to store hexadecimal codes (character patterns) specified by the user to create custom characters on the LCD. In addition to the CGRAM area, the DDRAM area is used to store the CGRAM address of a particular character, which is sent in hexadecimal format by the microcontroller.

2. Procedure & Discussion

2.1 Example 1

We use these keywords as shown in figure 2.1.1 when we search for parts in Proteus:

Part	Keyword
Microcontroller	LPC2138
LCD	LM016L

Figure 2.1. 1:keywords

Then, we simulate the program in Proteus by using the parts was searched, the result show in figure 2.1.2.

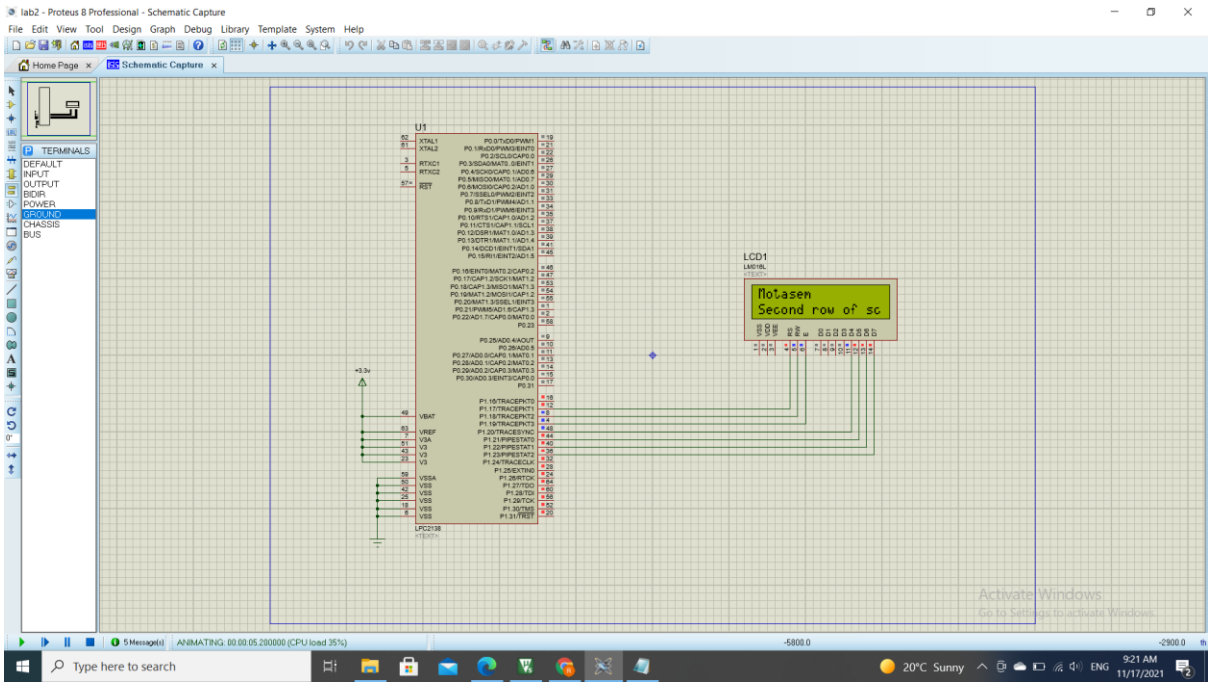

Figure 2.1. 2: Example1

From pin 17 through pin 23, the LCD is connected to the microprocessor on port 1, Then, in Keil MDK, we write the code, which is in the appendix. The code is divided into sections, which are referred to in the main. It first creates a character string and inserts the string "Motasem" after which it sets the LCD pin on port 1 as output, initializes the output, and then executes the function LCD commands1. To begin, clear the port pins and set the enable pin high, as well as the RS=0 for the command register, and then the R/W =0. That is to say, write, then shift 16 bits to write the data on the line using the most 4 bits of the command, then set the enable bit low, empty the port pins, and set the enable bit high, as well as RS=0 for the command register and R/W =0, then take the least 4 bits in the command and shift left 20 bits, and finally set the enable bit low. Returning to the main, I called the method LCD string, which displayed the character on the LCD from the array of names. I then called the function LCD commands, and finally the function LCD string.

2.2 Example 2

In this example, we'll write a software that moves a name across the LCD screen. Using push buttons, the user should be able to control the movement's direction (shift left, right, clear, or go to the second row).

This software begins by searching for parts in Proteus using the following keyword:

Part	Keyword
Microcontroller	LPC2138
LCD	LM016L

Figure 2.2. 1: keywords example_2

Then, we simulate the program in Proteus by using the parts was searched, the result show in figure 2.2.1.

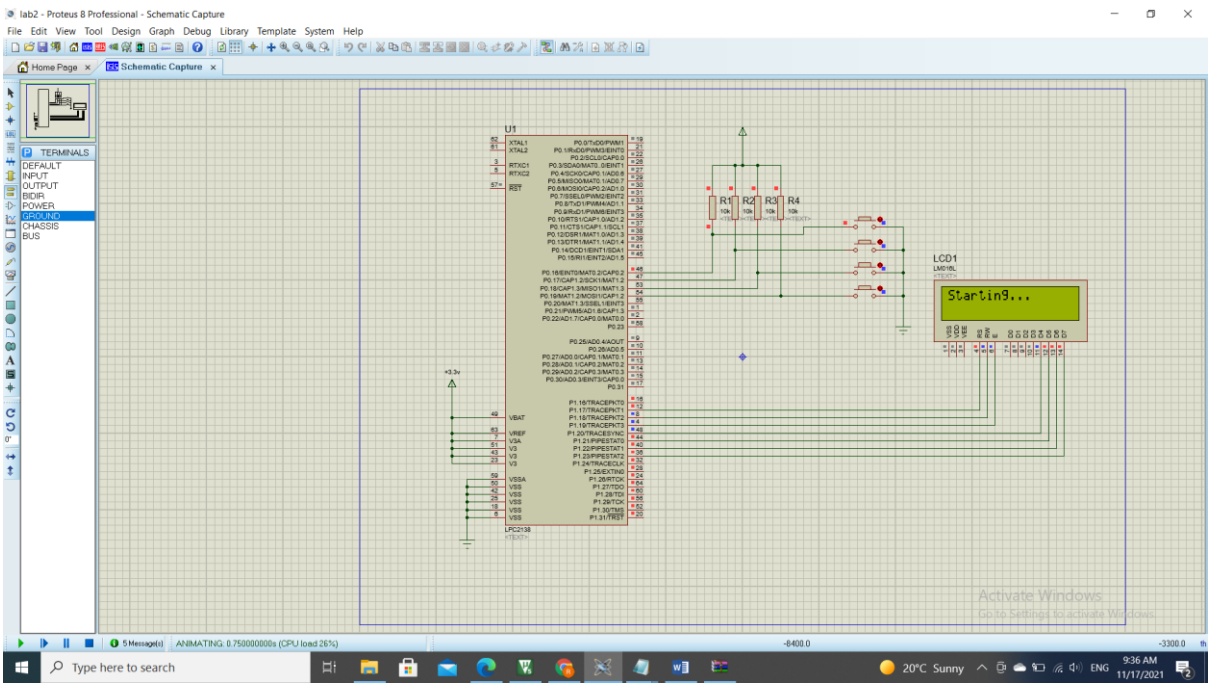

Figure 2.2. 2: example_2 in Proteus_ 1

The LCD is connected to the microcontroller in port 1 from pin 17 to pin 23, and the microcontroller is connected to resisters and a bush button from pin 46 to 54, and the resister connected with power.

Then, in Keil MDK, write the code, which is included in the appendix. The code is divided into sections, each of which is called in turn in the main.

It first creates a character string and inserts the string "DIAB" into it, then sets the LCD pins 16 and 18 in port 1 as input and the other pins as output, then uses the function LCD Intit to initialize the LCD, and finally uses the function LCD command to insert the string "starting" in the first row, as shown in figure 2.2.2, then clear the command and set the raw and column positions to 0 in the function LCD set _cursor pos, then call the function LCD string and put

the string of name in it, then when the push button is pressed, if the pin is 16 and the port 1 is 1, the column is shifted left and the column is changed if the column becomes less than 0 then set the column to 15. if the pin = 18 then it is jump and the row change if the row =0 then go to row 1 and if the row=1 then jump to 0 clear the command and set the new position and the string else if the pin = 18 then it is jump and the row change if the row =0 then go to row 1 and if the row=1 then jump to 0 clear the command and set the new position and the string

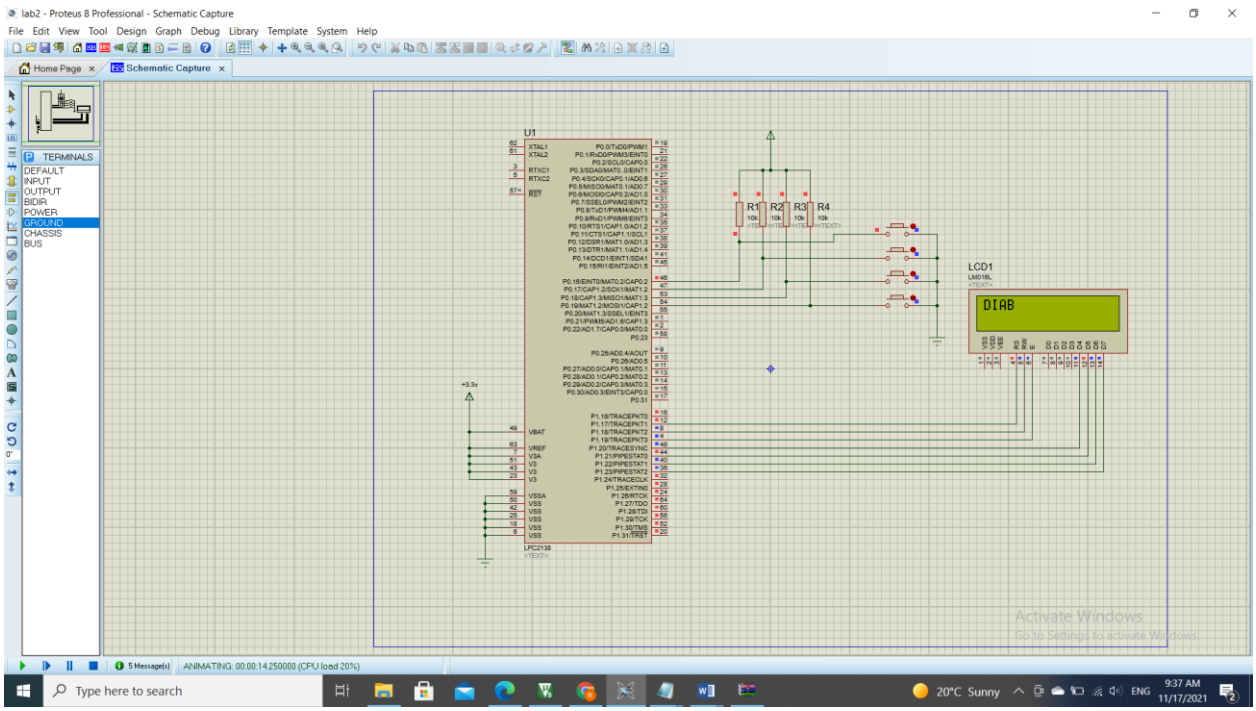

Figure 2.2. 3: example_2 in Proteus_ 2

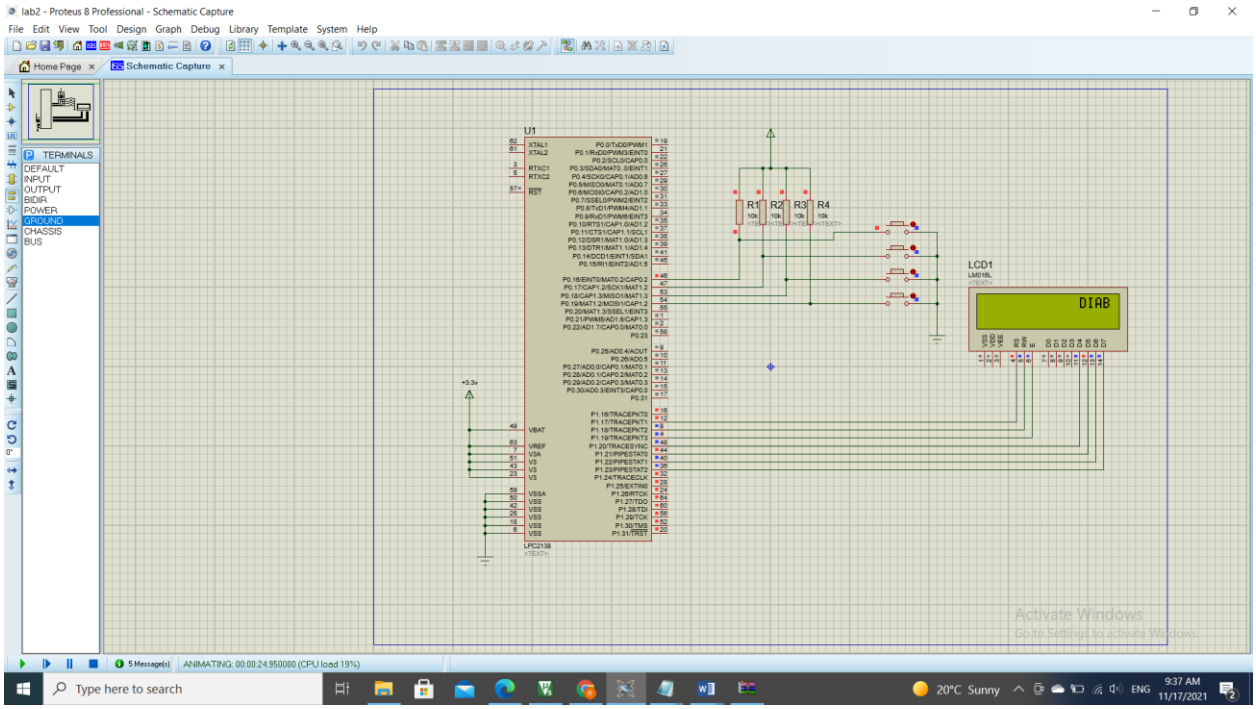

Figure 2.2. 4: shift_1

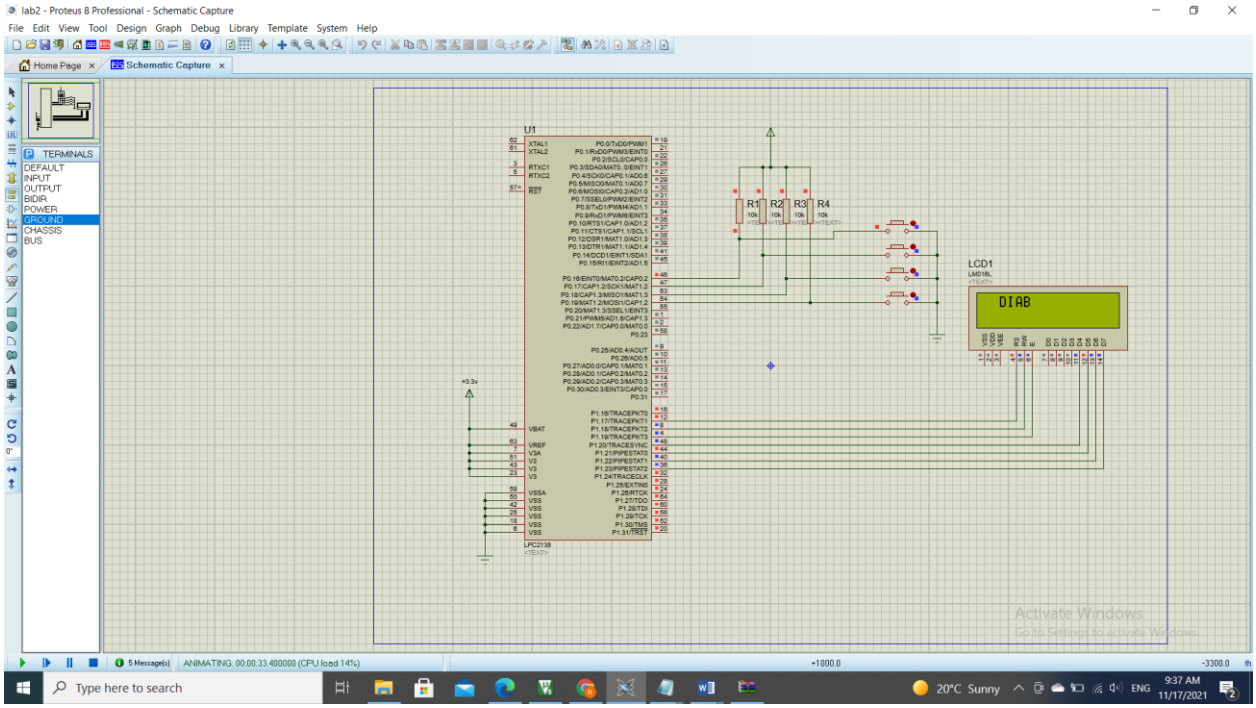

Figure 2.2. 5: shift_2

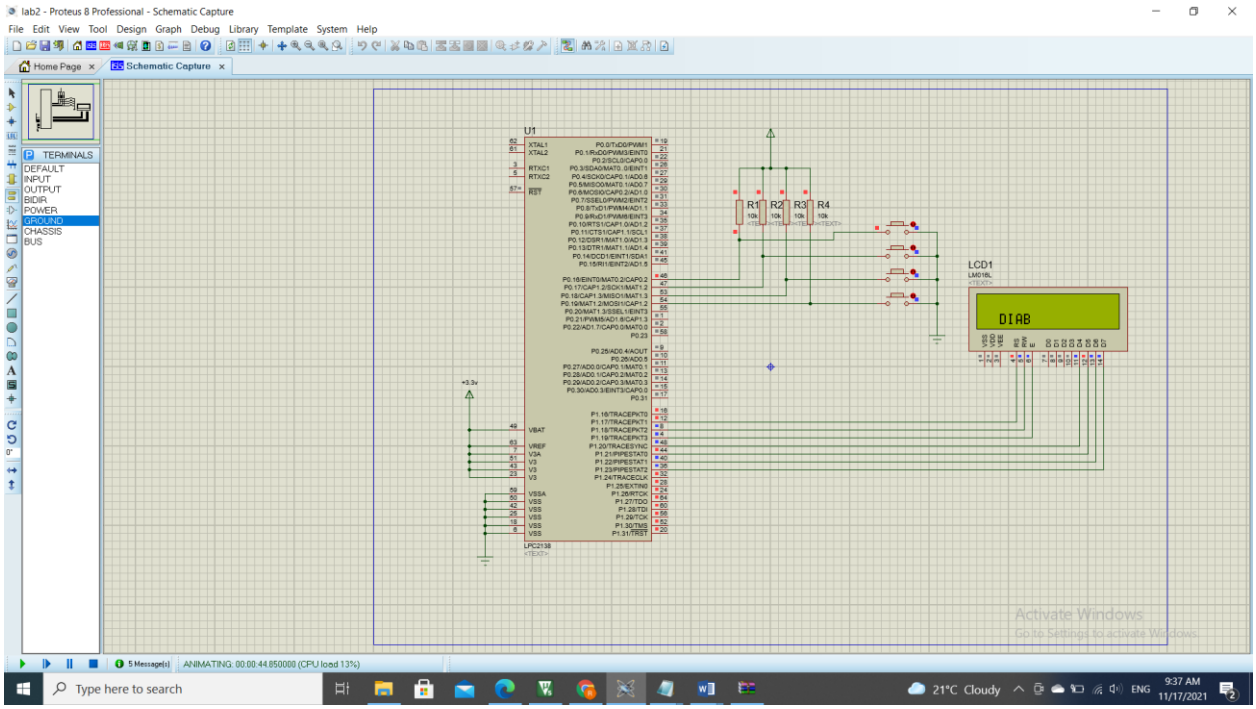

Figure 2.2. 6:jump_1

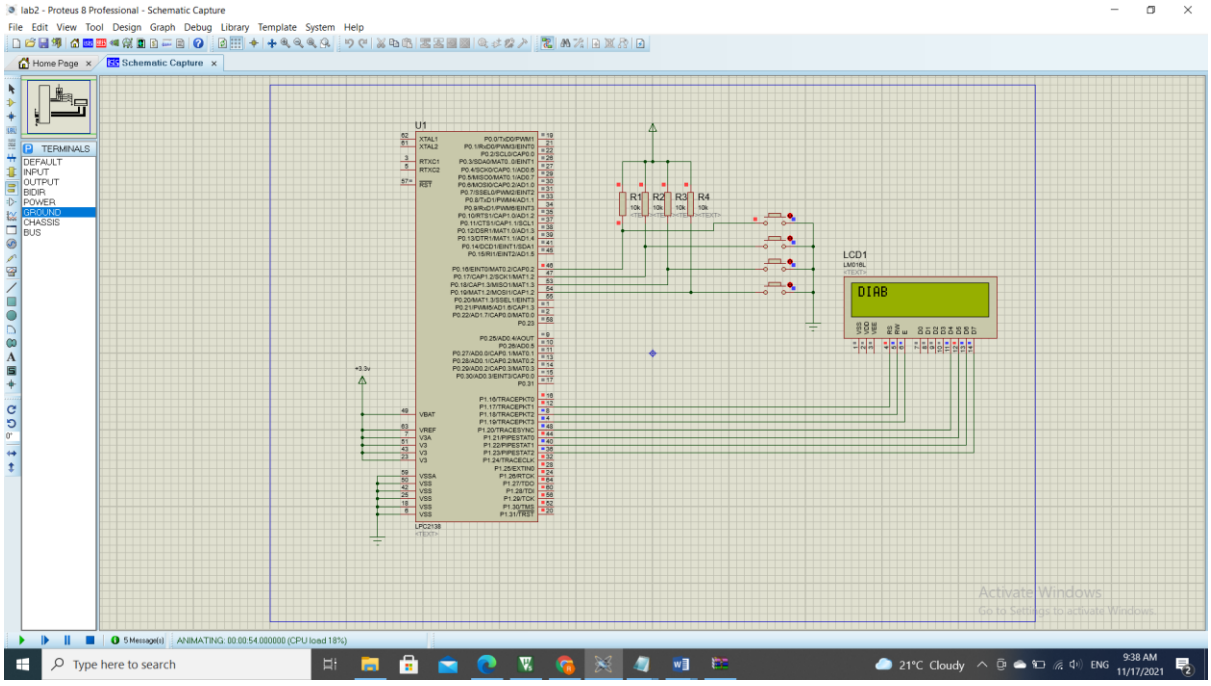

Figure 2.2. 7:jump_2

3. Conclusion

After completing this experiment, we learnt how to use LCD components, write and simulate a program in LCD, and control the movement direction.

4. References

[1] https://focuslcds.com/journals/alphanumeric-lcd-displays/

[2] Lab manual experiment 9
https://ritaj.birzeit.edu/bzumsgs/attach/1967855/ARM_LCD_ADC_Exp2+V3.pdf

5. Appendices

5.1 Example 1

```
// 3-11-11:00
#include <lpc213x.h>
#include "lcd.h"

void Delay(unsigned int times)
{
    int i, j;
    for (j = 0; j < times; j++)
        for (i = 0; i < 300; i++)
            ;
}

void LCD_Command(char command)
{
    int Temp;
    IO1CLR = CLR;
    IO1SET = EN;
    IO1CLR = RS;
    IO1CLR = RW;
    Temp = (command & 0xF0) << 16;
    IO1SET = IO1SET | Temp;
    Delay(2);
    IO1CLR = EN;
}

void LCD_Command1(char command1)
{
    int Temp;
    IO1CLR = CLR; /* Clearing the port pins               */
    IO1SET = EN;  /* Enable pin high                      */
    IO1CLR = RS;  /* RS=0 for command register            */
    IO1CLR = RW;  /* R/W=0 for write                      */
    /* Taking the first nibble of command   */
    Temp = (command1 & 0xF0) << 16;
    IO1SET = IO1SET | Temp; /* Writing it to data line */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */

    /* same as above for the second nibble */
    IO1CLR = CLR;
    IO1SET = EN;
    IO1CLR = RS;
    IO1CLR = RW;
    Temp = (command1 & 0x0F) << 20;
```

```
    IO1SET = IO1SET | Temp;
    Delay(2);
    IO1CLR = EN;
}

void LCD_Data(char data)
{
    int Temp;
    IO1CLR = CLR;               /* Clearing the port pins               */
    IO1SET = EN;                /* Enable pin high                      */
    IO1SET = RS;                /* RS=1 for data register               */
    IO1CLR = RW;                /* R/W=0 for write                      */
    Temp = (data & 0xF0) << 16; /* Taking the first nibble of command   */
    IO1SET = IO1SET | Temp;     /* Writing it to data line              */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */

    IO1CLR = CLR;               /* Clearing the port pins               */
    IO1SET = EN;                /* Enable pin high                      */
    IO1SET = RS;                /* RS=1 for data register               */
    IO1CLR = RW;                /* R/W=0 for write                      */
    Temp = (data & 0x0F) << 20; /* Taking the second nibble of command  */
    IO1SET = IO1SET | Temp;     /* Writing it to data line              */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */
}

void LCD_String(unsigned char *dat)
{
    /* Check for termination character */
    while (*dat != '\0')
    {
        /* Display the character on LCD */
        LCD_Data(*dat);
        /* Increment the pointer */
        dat++;
    }
}

void LCD_Init(void)
{
    Delay(15);
    LCD_Command(0x30);
    Delay(10);
    LCD_Command(0x30);
    Delay(5);
    LCD_Command(0x30);
    LCD_Command(0x20);
```

```
    LCD_Command1(0x28);
    LCD_Command1(0x01); /* Clear display                    */
    LCD_Command1(0x06); /* Auto increment                   */
    LCD_Command1(0x0C); /* Cursor off                       */
}

int main()
{
    unsigned char name[] = "Motasem";
    IO1DIR = 0x00FE0000; /* LCD pins set as o/p?????????     */
    LCD_Init();          /* Initialise LCD                  */
    LCD_Command1(FIRST_ROW);
    LCD_String(name);
    LCD_Command1(SECOND_ROW);
    LCD_String("Second row of screen");
    Delay(5000);
    LCD_Command1(LCD_CLEAR); /* Clear screen                */
    while (1)
        ;
}
```

5.2 Example 2

```
// 3-11-11:00
#include <lpc213x.h>
#include "lcd.h"

void Delay(unsigned int times)
{
    int i, j;
    for (j = 0; j < times; j++)
        for (i = 0; i < 300; i++)
            ;
}

void LCD_Command(char command)
{
    int Temp;
    IO1CLR = CLR;                  /* Clearing the port pins               */
    IO1SET = EN;                   /* Enable pin high                      */
    IO1CLR = RS;                   /* RS=0 for command register            */
    IO1CLR = RW;                   /* R/W=0 for write                      */
    Temp = (command & 0xF0) << 16; /* Taking the first nibble of command   */
    IO1SET = IO1SET | Temp;        /* Writing it to data line              */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */
}

void LCD_Command1(char command1)
{
    int Temp;
    IO1CLR = CLR;                   /* Clearing the port pins              */
    IO1SET = EN;                    /* Enable pin high                     */
    IO1CLR = RS;                    /* RS=0 for command register           */
    IO1CLR = RW;                    /* R/W=0 for write                     */
    Temp = (command1 & 0xF0) << 16; /* Taking the first nibble of command  */
    IO1SET = IO1SET | Temp;         /* Writing it to data line             */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */

    IO1CLR = CLR;                   /* Clearing the port pins              */
    IO1SET = EN;                    /* Enable pin high                     */
    IO1CLR = RS;                    /* RS=0 for command register           */
    IO1CLR = RW;                    /* R/W=0 for write                     */
    Temp = (command1 & 0x0F) << 20; /* Taking the second nibble of command */
    IO1SET = IO1SET | Temp;         /* Writing it to data line             */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */
}
```

```
void LCD_Data(char data)
{
    int Temp;
    IO1CLR = CLR;               /* Clearing the port pins               */
    IO1SET = EN;                /* Enable pin high                      */
    IO1SET = RS;                /* RS=1 for data register               */
    IO1CLR = RW;                /* R/W=0 for write                      */
    Temp = (data & 0xF0) << 16; /* Taking the first nibble of command   */
    IO1SET = IO1SET | Temp;     /* Writing it to data line              */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */

    IO1CLR = CLR;               /* Clearing the port pins               */
    IO1SET = EN;                /* Enable pin high                      */
    IO1SET = RS;                /* RS=1 for data register               */
    IO1CLR = RW;                /* R/W=0 for write                      */
    Temp = (data & 0x0F) << 20; /* Taking the second nibble of command  */
    IO1SET = IO1SET | Temp;     /* Writing it to data line              */
    Delay(2);
    IO1CLR = EN; /* Enable pin low to give H-L pulse     */
}

void LCD_String(unsigned char *dat)
{
    while (*dat != '\0') /* Check for termination character      */
    {
        LCD_Data(*dat); /* Display the character on LCD         */
        dat++;          /* Increment the pointer                */
    }
}

void LCD_Init(void)
{
    Delay(15);
    LCD_Command(0x30);
    Delay(10);
    LCD_Command(0x30);
    Delay(5);
    LCD_Command(0x30);
    LCD_Command(0x20);
    LCD_Command1(0x28);
    LCD_Command1(0x01); /* Clear display                        */
    LCD_Command1(0x06); /* Auto increment                       */
    LCD_Command1(0x0C); /* Cursor off                           */
}

void LCD_set_cursor_pos(int row, int col)
```

```
{
    unsigned char cp;
    if (row == 0)
        cp = FIRST_ROW;
    else
        cp = SECOND_ROW;
    cp |= col;
    LCD_Command1(cp);
}

int main()
{
    unsigned char name[] = "DIAB";
    int col = 0, row = 0;

    IO0DIR &= (~(1 << 16)); // P1.16 is input
    IO0DIR &= (~(1 << 17)); // P1.17 is input
    IO0DIR &= (~(1 << 18)); // P1.18 is input
    IO0DIR &= (~(1 << 19)); // P1.19 is input
    IO0SET = 0xFFFFFFFF;

    IO1DIR = 0x00FE0000; /* LCD pins set as o/p?????????         */
    LCD_Init();          /* Initialise LCD                      */

    LCD_Command1(FIRST_ROW); /* 1st row, 1st location                */
    LCD_String("Starting...");
    Delay(1000);
    LCD_Command1(LCD_CLEAR);

    LCD_set_cursor_pos(0, 0);
    LCD_String(name);

    while (1)
    {
        if (!(IO0PIN & (1 << 16)))
        { //left
            col--;
            if (col < 0)
                col = 15;
            LCD_Command1(LCD_CLEAR);
            LCD_set_cursor_pos(row, col);
            LCD_String(name);
            Delay(300);
        }

        if (!(IO0PIN & (1 << 17)))
        { //right
            col++;
```

```
            col %= 16;
            LCD_Command1(LCD_CLEAR);
            LCD_set_cursor_pos(row, col);
            LCD_String(name);
            Delay(300);
        }

        if (!(IO0PIN & (1 << 18)))
        { //jump
            row = (row == 0 ? 1 : 0);
            LCD_Command1(LCD_CLEAR);
            LCD_set_cursor_pos(row, col);
            LCD_String(name);
            Delay(300);
        }

        if (!(IO0PIN & (1 << 19)))
        { //reset
            LCD_Command1(LCD_CLEAR);
            row = col = 0;
            LCD_set_cursor_pos(row, col);
            LCD_String(name);
            Delay(300);
        }
    }
}
```