Water Level Controller

EE447 TERM PROJECT FINAL REPORT

Deniz Karakay 2443307
Hüseyin Totan 2446979

Deniz Karakay 2443307
Hüseyin Totan 2446979

08.01.2022

Table of Contents

Introduction

The purpose of the EE447 laboratory work was to become familiar with the TM4C123G microcontroller and its utility modules. In this final project, we are expected to build upon this knowledge and use it to complete a multi-functional task. The objectives of the project include interpreting the requirements of a complex task and breaking it down into subtasks, ensuring the cooperation of utility modules, understanding the complex hardware, and determining the compatibility of its components, writing multi-task software for a given complex setup, and introducing the use of serial communication on the TM4C123G through the SPI protocol.

Our goal is to create a water level controller system using a water level sensor, water pumps, a Nokia 5110 LCD, a potentiometer, an on-board RGB LED, and external push-buttons. We will use the water level sensor to detect the water level and control the pumps accordingly. The on-board RGB LED will be used to indicate the status of the system, and the LCD will display the limits and status of the sensor. The potentiometer and push-buttons will allow us to modify the upper and lower limits.

Teamwork

Deniz

- ADC configurations for Sensor and Potentiometer
- Configuration for LCD
- Create sub-loops for sensors and potentiometers
- Helping Hüseyin to build the demo
- Preparing report

Hüseyin:

- Driving motors, setting up the circuit
- Configuration for Keypad
- Combining LCD, sensor, potentiometer, keypad etc. in a main loop
- Helping Deniz to build the demo
- Preparing report

Components

1. Water Level Sensor

2. NOKIA 5110 LCD Screen

3. 1 Potentiometer

4. 1 4x4 Keypad

5. RGB LED Placed on the TM4C123G Board

6. 2 Water Pumps

7. 5V Power Supply

8. Jumper cables

9. Resistors, Diodes etc.

10. 4x BC337 Transistors

Pinout of Our System

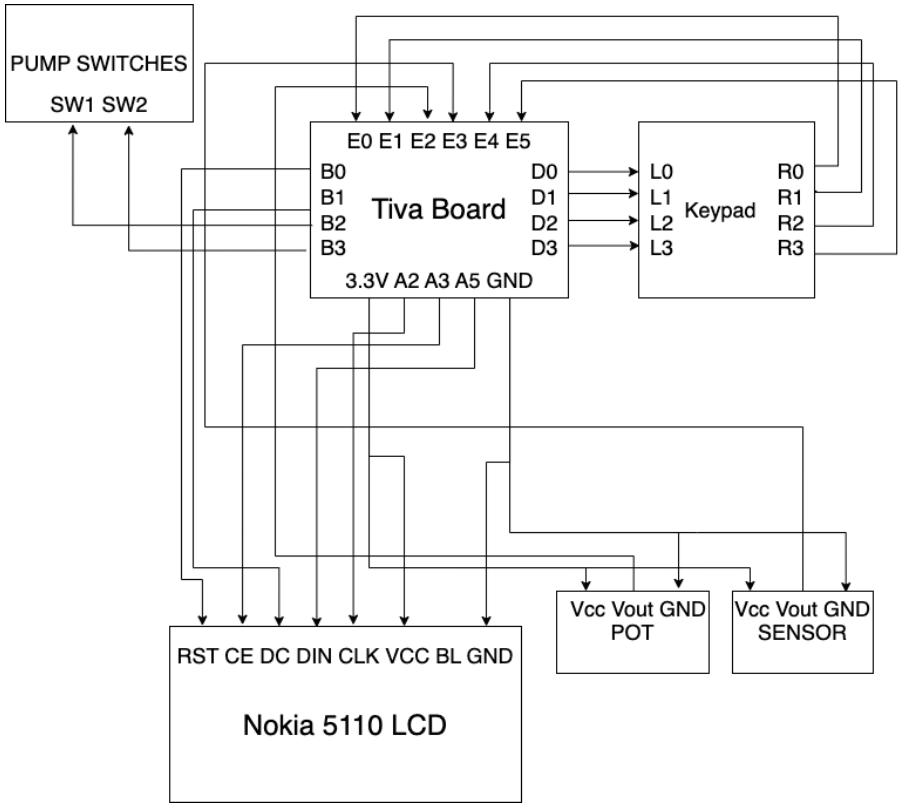

Figure 1. Pinout of our system

Main Flow

In the main program flow we have used a single state variable in order to manage our control flow. Like a mealy machine, our flow chart has an inner state defined by a single state variable and various inputs from potentiometer, sensor, and key matrix from Keypad. Both inner state and inputs are effective on the decision of next state.

We also worked on Keypad to accept decimal outputs. Since we had a preliminary work for a lab that asks similar type of work, it was easy for us to configure and implement it. Therefore, we enabled 4x4 Key matrix buttons to not only accept decimal inputs, but also control to change low or high limit based on which button is pressed. Buttons 13, 14, 15 and 16 on the key matrix are used directly modify the inner state of the main program flow. Button 13 associates pot value to lower limit, button 14 associated pot value to upper limit, button 15 directs the main program flow to receive a decimal 2 digits value from key matrix (and set it to lower limit) and button 16 directs the main program flow to receive a decimal 2 digits value from key matrix (and set it to higher limit). The details can be examined in *Figure 2*.

In pot associated modes in order to prevent undesired outcomes limits are checked to ensure limit values does not violate the inequality of *lower_limit < higher_limit*. In addition to these, the main flow program calls necessary functions, for LCD Display, Key Matrix Scan, external pump switches and on board RGB LED control.

Unlike the other functions, ADC data from sensor and Potentiometer are stored in arrays of 256 integers by means of SysTick interrupt *(Figure 3)*. Ones every 256 iterations, the interrupt updates the global pot and sensor data with the average obtained from the previously mentioned arrays.

```
while(1){
    DELAY100();
    key_matrix=keyMatrix((char*) (Port_D+(0x0F<<2)),0x0F,(char*) (Port_E+(0x33<<2)) ,0x33);
    switch(key_matrix){     //uses most sigbificant 4 bits of port B to scan buttons
        case 11:
        case 12:pot_set = 0;break;
        case 13:pot_set = 1;break;  //buttons 10-16 are used to manage control flow
        case 14:pot_set = 2;break;
        case 15:pot_set = 3;break;
        case 16:pot_set = 4;break;
        default:break;
    }
    switch(pot_set){
        case 1:if(upper_l>(pot*99)/4095)lower_l = (pot*99)/4095;    break;
        case 2:if(lower_l<(pot*99)/4095)upper_l = (pot*99)/4095; break;

        case 3:if(0 < key_matrix && key_matrix < 10){pot_set = 5;lower_l = key_matrix*10;}else if(key_matrix == 10){pot_set = 5;lower_l = 0;}break;
        case 4:if(0 < key_matrix && key_matrix < 10){pot_set = 6;upper_l = key_matrix*10;}else if(key_matrix == 10){pot_set = 6;upper_l = 0;}break;

        case 5:if(key_matrix==0)pot_set = 7;break;
        case 6:if(key_matrix==0)pot_set = 8;break;

        case 7:if(0 < key_matrix && key_matrix < 10){pot_set = 0;lower_l += key_matrix;}else if(key_matrix == 10){pot_set = 0;}break;
        case 8:if(0 < key_matrix && key_matrix < 10){pot_set = 0;upper_l += key_matrix;}else if(key_matrix == 10){pot_set = 0;}break;
        default:break;
    }
    update_rgb_led((sensor*2*99)/4095,lower_l,upper_l);
    update_pumps((sensor*2*99)/4095,lower_l,upper_l);
    sprintf(arr,"key: %d pot_s: %d\r\4\n",key_matrix,pot_set);OutStr(arr); //debug
    sprintf(arr,"S:%d P:%d LL:%d UL: %d\r\4\n",(sensor*2*99)/4095,(pot*99)/4095,lower_l,upper_l);
    OutStr(arr);
    //screen_clear();
    lcd_update_screen(lower_l,upper_l,(pot*99)/4095,(sensor*2*99)/4095 );
    }

}
```

Figure 2. Main Loop of our system

```
void SysTick_Handler(){
    int sensor_sum=0,pot_sum=0;
    static int sample_count = 0;
    if(sample_count<256){
        sensor_samples[sample_count] = get_sensor_data();
        pot_samples[sample_count] = get_pot_data();
        sample_count++;
    }else{
        sample_count = 0;
        for(int i=0;i<256;i++){
            sensor_sum += sensor_samples[i];
            pot_sum += pot_samples[i];
        }
        sensor = sensor_sum >> 8;
        pot = pot_sum >> 8;
    }
}
```

Figure 3. SysTick Handler Function

LCD

The Nokia 5110 is a particular type of Liquid Crystal Display (LCD) screen that was first utilized in the Nokia 5110 cell phone. It has a monochrome screen that can show rudimentary pictures and text. It is frequently employed in tasks requiring a compact, affordable display.

The Nokia 5110 LCD screen must be connected to a microcontroller that supports the Serial Peripheral Interface (SPI) protocol to be used with the SPI protocol. The microcontroller is then in charge of transmitting data to the LCD display through the SPI bus. We can do this by feeding the screen several commands and pieces of data, and the screen will understand and display the data as necessary. By using the SPI protocol, we can communicate with the Nokia 5110 LCD screen in a simple and efficient manner.

SPI

Serial transmission is a method of transmitting data one bit at a time over a period of time, using fewer wires compared to parallel communication. However, it is more complex as the data is not transmitted at a uniform rate and is instead sent in packets with pauses between them. In synchronous systems, separate channels are used for data and timing information, with the timing channel transmitting clock pulses to the receiver. The Serial Peripheral Interface (SPI) is a type of synchronous serial communication that is commonly used in embedded systems and requires at least three wires: a clock, serial data out, and serial data in. In this method, the master device sets the clock rate and sends commands to the slave devices through its serial data out port, with the slaves returning output through the serial data in port. Chip select may also be used to specify which slave device should listen to the master.

SPI Configuration for Nokia 5510 LCD

To use a Nokia 5110 display, three steps are necessary: initializing the input/output pins as SPI pins, configuring the SPI module on the TM4C123 device to be compatible with the LCD, and

initializing the display for communication and to receive data. The Nokia 5110 uses SPI signals to receive commands, text, and images, but a separate pin called Data/Command (DC) is used to distinguish between data meant to be displayed and commands meant to control the screen. When DC is low, the incoming SPI bits are interpreted as a command, and when it is high, they are interpreted as data.

We put initialization codes under ***"LCD/init. s"*** file to configure GPIO and SPI interface. We utilized SSI0, which we used Port A pins, namely A2-A5 for SPI configuration. For Reset, and D/C pins, we chose to use Port B, specifically B0 and B1. In order to configure GPIO for use with the Nokia 5110 LCD along with configuring SPI interface, we followed following steps:

1. Enable the clock for the appropriate GPIO (RCGCGPIO) ***[Port A & B]***
2. Enable the clock for the appropriate SSI (RCGCSSI) ***[SSI0]***
3. Set the clock, pins as digital (DEN) ***[Port A & B]***
4. Set the direction of these pins (DIR) ***[Port A & B]***
5. Set the alternate function of these pins (AFSEL) ***[Only for A2-A5]***
6. Set the port control pins to route the SSI interface to the relevant pins (PCTL) ***[Only for A2-A5]***
7. Lock the SPI interface (CR1) ***[SSI0]***
8. Set the clock rate (CPSR, CR0) according to the maximum data rate of the LCD
9. Set the data size to 8-bits and Freescale mode (CR0)
10. Clear SPH and SPO
11. Set the SPI mode (CR0)
12. Unlock the SPI interface (CR1)
13. Check SR of SSI0 before finalization of submodule.

To be able to control the pixels of LCD, we used ***"/LCD/screen.s"*** file. In this file the following steps can be found:

1. Set Reset Pin low ***(B0)***
2. Wait 150MS **(DELAY150.s)**

3. Set Reset Pin high ***(B0)***
4. Command Mode, set low ***(B1)***
5. Set the H=1 & V=0
6. Set the VOP value ***(0xB5)***
7. Set the temperature control value ***(0x05)***
8. Set the voltage bias value to ***(0x13)***
9. Set the H value ***(0x20)***
10. Normal Display Mode ***(0x0C)***
11. Set Cursor X=0 ***(0x80)***
12. Set Cursor Y=0 ***(0x40)***

To write data, we go back to Data Mode. I prepared a simple subroutine called Update Cursor under ***"/LCD/ lcd_update_screen.s"*** to update position of our cursor with respect to X and Y, by holding data in R5 and R6. Details can be seen under *Figure 4.*

```
;                   Update Cursor
;                   Command mode
update_cursor       PROC
                    PUSH    {LR,R11}
                    LDR     R1,=GPIO_PORTB_DATA
                    LDR     R0,[R1]
                    AND     R0,#0xFD
                    STR     R0,[R1]


;                   X=0
                    MOV     R4,#0x80
                    ADD     R4,R5
                    BL      lcd_send_data

;                   Y=0
                    MOV     R4,#0x40
                    ADD     R4,R6
                    BL      lcd_send_data

                    BL      delay

;                   Data Mode
                    LDR     R1,=GPIO_PORTB_DATA
                    LDR     R0,[R1]
                    ORR     R0,#0x2
                    STR     R0,[R1]

                    ;BL     delay
                    POP     {LR,R11}
                    BX      LR

                    ENDP
                    END
```

Figure 4. Update Cursor part under lcd_update_screen

After selecting cursor correctly, it is easy to light up specific pixels. We followed official datasheet to draw what we aim to do. Example code to draw 5 can be examined *in Figure 5*.

```
; FIVE
five
                MOV     R4,#0x8F
                BL      lcd_send_data

                MOV     R4,#0x89
                BL      lcd_send_data

                MOV     R4,#0x89
                BL      lcd_send_data

                MOV     R4,#0x89
                BL      lcd_send_data

                MOV     R4,#0xF9
                BL      lcd_send_data

                MOV     R4,#0x0
                BL      lcd_send_data

                BL      finish
```

Figure 5. Drawing 5 on 5110 LCD

LCD Results

In the end, we followed the requirements, and the final version can be seen in *Figure 6.*

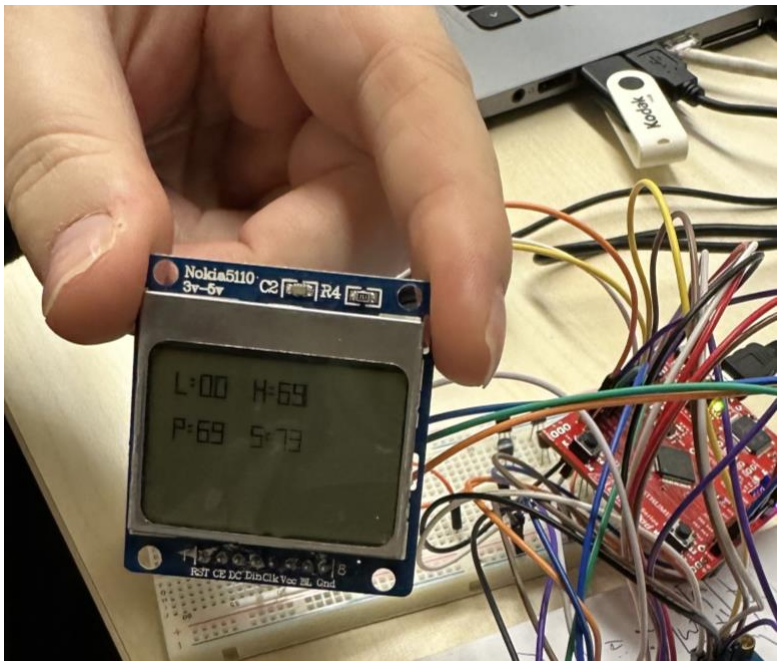

Figure 6. Final LCD View

Sensor & Potentiometer

Water Level Sensor

This water level sensor has five sense and five power traces that are used to detect the level of water. These traces are normally not connected, but when they are submerged in water, they are bridged. The sense and power traces create a variable resistor that functions like a potentiometer, with its resistance changing based on the level of water exposure. The resistance of the sensor is inversely proportional to the water level, meaning that when the sensor is placed in an area with a high-water level, its resistance decreases. The sensor produces an output voltage that is proportional to its resistance, which can be measured to determine the water level.

ADC

For Potentiometer and Water Level Sensor, we utilized ADC0. We used SSCTL3 for Water Level Sensor (E3) and SSCTL2 for Potentiometer. The initialization code can be examined in *Figure 7*.

```
void init_begin(){  //initializes ADC0 and its fifo3 and fifo2, Port_E pins 2-3 for ADC0
    SYSCTL->RCGCGPIO |= 0x30;
    SYSCTL->RCGCADC |=0x01;

    GPIOE->AFSEL |= (0x0C);
    GPIOE->DIR &= ~0x0C;
    GPIOE->DEN &= ~(0x0C);
    GPIOE->AMSEL |= (0x0C);

    ADC0->ACTSS &= ~(1<<3);
    ADC0->EMUX  &= ~(15<<12);
    ADC0->SSCTL3 |=(1<<2);
    ADC0->SSCTL3 |=(1<<1);
    ADC0->PC |= (1<<0);
    ADC0->ACTSS |= (1<<3);



    ADC0->ACTSS  &= ~(1<<2);
    ADC0->EMUX   &= ~(15<<9);
    ADC0->SSMUX2 |= (1<<0);
    ADC0->SSCTL2 |=(1<<2);
    ADC0->SSCTL2 |=(1<<1);
    ADC0->ACTSS |= (1<<2);


}
```

Figure 7. Initialization of ADC for sensor & potentiometer

We used SysTick to collect 256 samples and take the average of them. This part is explained in the Main Flow. Furthermore, these two functions utilized to get data from both potentiometer and sensor. We checked RIS of ADC0 and get the sample form SSFIFO3 for sensor, SSFIFO2 for potentiometer *(Figure 8)*

```
volatile int get_sensor_data(){     //returns sensor data from ADC0 fifo3
        volatile int sample = -1;
        if(ADC0->RIS == 0x0C){
            ADC0->PSSI |= (1<<3);
            sample = ADC0->SSFIFO3;
        }
            return sample;

}


volatile int get_pot_data(){        //returns pot data from ADC0 fifo2
        volatile int sample = -1;
        if(ADC0->RIS == 0x0C){
            ADC0->PSSI |= (1<<2);
            sample = ADC0->SSFIFO2;
        }
            return sample;

}
```

Figure 8. Methods to get sensor and pot data

Pumps & RGB LED

For driving pumps, we followed our design from preliminary work. The circuit can be seen in *Figure 9.* We used B2 and B3 for pumps.

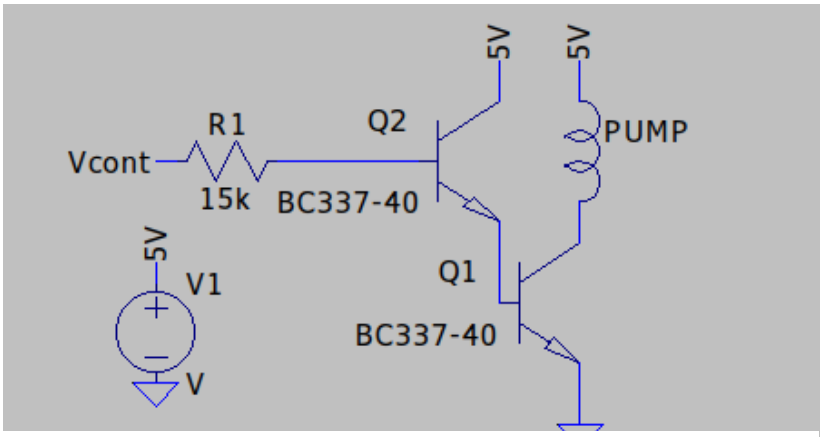

Figure 9. Circuitry to drive pumps

For lighting up on-board RGB LED, we enabled Port F and used F1-F3 to give output in three different colors. Both for driving RGB LED and Pumps, we used similar function, and entire code can be seen in *Figure 10*.

```
void update_rgb_led(int data, int lower_limit, int higher_limit){   //sets RGB led according to limits and sensor data
    if(data<lower_limit){GPIOF -> DATA |= 2;GPIOF -> DATA &= ~0xC;
    }else if(data>higher_limit){GPIOF -> DATA |= 4;GPIOF -> DATA &= ~0xA;
    }else{GPIOF -> DATA |= 8;GPIOF -> DATA &= ~0x6;}
}
void update_pumps(int data, int lower_limit, int higher_limit){     //sets pump switches according to limits and sensor data
    if(data<lower_limit){GPIOB -> DATA |= 4;GPIOB -> DATA &= ~8;}
    else if(data>higher_limit){GPIOB -> DATA |= 8;GPIOB -> DATA &= ~4;} //port B pin 3 export pump, pin 2 import pump
    else GPIOB -> DATA &= ~0xC;
}
```

Figure 10. Functions to drive RGB LED and Pumps

Demo Results

Figure 11. Hüseyin tests the system

Figure 12. Deniz tests the water level sensor

Figure 13. Water flow between our yogurt boxes

References

5110 Datasheet, equivalent, graphic LCD. (no date) *5110 LCD Datasheet pdf - Graphic LCD. Equivalent, Catalog*. Available at: https://datasheetspdf.com/pdf/1418219/Nokia/5110/1 (Accessed: January 8, 2023).

BC337, BC337-25, BC337-40 amplifier transistors - onsemi (no date). Available at: https://www.onsemi.com/pdf/datasheet/bc337-d.pdf (Accessed: January 8, 2023).

Semiconductor company | ti.com - Texas Instruments (no date). Available at: https://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf (Accessed: January 8, 2023).