MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF

ELECTRICAL-ELECTRONICS ENGINEERING

EE447 PROJECT REPORT

“Park Sensor System”

2020-2021 Fall Term

Student Name: Efe Berkay YİTİM

Student ID: 2305761

Lecturer: Prof. Dr. Gözde BOZDAĞI AKAR

Project Date: 28.12.2020 – 31.01.2021

Table of Contents

List of Figures

List of Tables

1.Introduction

The main objectives of the term project are to gain the ability to encapsulate the task into sub-tasks, maintaining the co-operation of the modules, and learning and applying the serial communication on TM4C123G and utilization of the facility on SPI protocol.

In this project, we, students, were expected to implement a “Park Sensor System” using TM4C123G Microcontroller and peripherals like stepper motor, ultrasonic distance sensor and LCD screen. According to the project definition, our system should have three major functions: Ultrasonic Sensing, Preventative Breaking and User Interface.

For the Ultrasonic Sensing function, the sensors should continuously measure the distance to the obstacle and if this distance is closer than a user specified threshold, the system will take the control and apply preventative break.

For the Preventative Breaking function, as I have already given hint in the above paragraph, the car needs to stop if an obstacle is too close to the car. The breaks must engage after closing to the threshold, and fully stop the car after reaching to the threshold.

The user should be able to see the measurement, the current threshold and the state of the brakes. For that reasons, a User Interface function is required. If the user is adjusting the threshold manually, the user should also be able to see the threshold he/she is setting. Also, a progress bar is also required to inform the user graphically instead of just numbers.

2.Solution

In this part, I am not going to explain some of the specific parts of my project and source codes. Instead, I am planning to explain every source file I have used in detail. If an explanation is required for a specific portion of codes, I have tried to write comments for almost everything in the source files.

For the solution part, there are three states of the microcontroller. The first one is normal mode, the second mode is the threshold setting mode, and the last mode is preventative breaking mode.

The normal mode indicates that, neither the user is setting threshold nor the threshold value is exceeded. The car is in the normal travel state. Therefore, in this mode, first thing is to do is measuring the distance and displaying this distance value, the current threshold value, the state information as normal mode and the progress bar. Later, this distance measurement will be compared with threshold, and if the measurement is below threshold, the system will enter to breaking mode. If the measurement is between threshold and two times the threshold, the speed of the motor will be calculated accordingly, and break will be engaged proportionally. If the measurement is above two times threshold, the system will continue on normal mode.

In the main loop of the program, it will be checked that the threshold setting button (SW1) is pressed or not. If it is pressed, the system will enter to the threshold setting mode. In this mode there are two options left to decide for the user. The first and default option is to adjust the threshold value using a potentiometer. By rotating the shaft of the potentiometer, the user can adjust the threshold value between 0 and 990. If the threshold button SW1 is pressed again, the threshold value will be set according to the user input. Else if, any of the keypad buttons is pressed, the system will enter to keypad mode. In this mode, the user can press the 0-9 buttons of the keypad to adjust the threshold. Buttons 10-15 will be ignored. For every button press, the previous digits will be shifted one digit to the left and the newly entered digit will be the ones digit. User can press any number of keys; the digits will be always shifted to the left. After the intended threshold value is adjusted using keypad, the user has two options. First one is to press the threshold setting button SW1 to set the threshold, or press the 16. button on the keypad to return to potentiometer mode. While setting the threshold mode, no distance measurement will be done, only threshold value and the threshold mode information will be displayed on the screen.

The last mode of the solution is preventative breaking mode. The only way to enter the breaking mode is from normal mode. In the main loop of the program, it will be checked first if the breaking mode signal is set to 1 in the normal mode. If it is set, the mode of the program will be preventative breaking mode. This mode is rather simple. The first thing to do is stop the motor. Then, wait for the reset button SW2 press. If it is pressed, the program will return to the main loop and continue its flow. If not, the program will stay on the preventative breaking mode, even if the obstacle is moved away the car and the distance measurement is again above threshold value. While in preventative breaking mode, the program will also measure the distance and display this measurement, the current threshold value, the preventative breaking mode information and progress bar.

The main flowchart of my solution can be seen on the Figure 1 below.

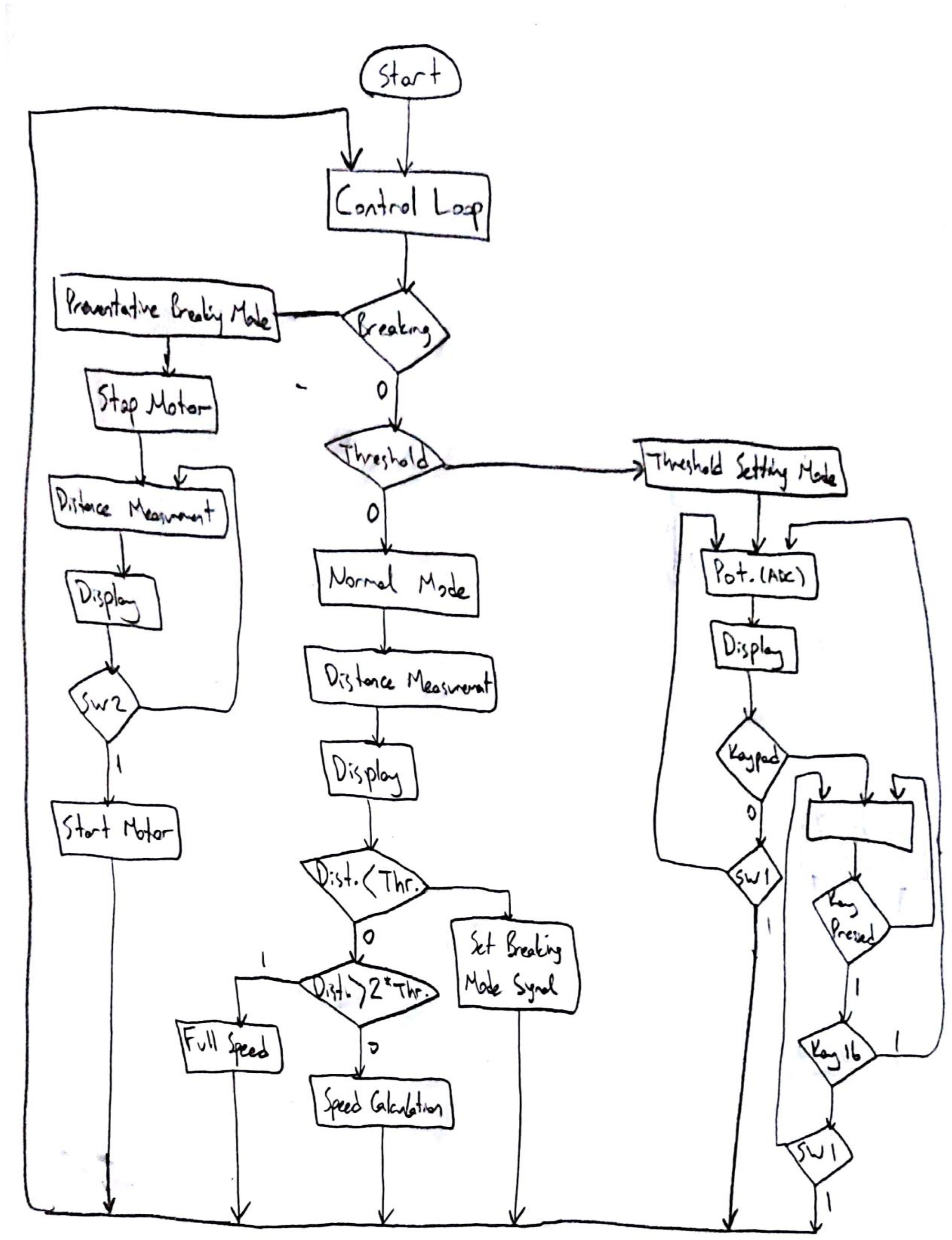

Figure 1 Main Flowchart

2.1. Submodules

For the solution part, I have divided my system design into 5 submodules. These submodules are the Distance Measurement Submodule, the Threshold Setting Submodule, the GUI Submodule, the Movement Submodule, and the User Switch Submodule. I am going to explain every submodule in detail in the following sections.

Before going into the details, I wanted to show the pin assignment of the microcontroller.

Peripherals	Pins on Peripheral	TM4C123G
LCD Screen	Pin 1 (RST)	PA7
	Pin 2 (CE)	PA3
	Pin 3 (DO)	PA6
	Pin 4 (DIN)	PA5
	Pin 5 (CLK)	PA2
	Pin 6 (VCC)	3.3V
	Pin 7 (DL)	3.3V
	Pin 8 (GND)	GND
Stepper Motor	IN1	PC4
	IN2	PC5
	IN3	PC6
	IN4	PC7
	VCC	5V
	GND	GND
Ultrasonic Distance Sensor	VCC	5V
	TRIG	PF2
	ECHO	PB4
	GND	GND
Potentiometer	Terminal 1	GND
	Terminal 2	PE3
	Terminal 3	3.3V
Keypad	L1	PB0
	L2	PB1
	L3	PB2
	L4	PB3
	R1	PD0
	R2	PD1
	R3	PD2
	R4	PD3

Table 1 Pin Assignments

2.1.1. Distance Measurement Submodule

This submodule consists of ultrasonic distance sensor module combined with TM4C123G. The main objective of this submodule is to measure the distance to the obstacle. In order to measure the distance, I have created three source files named DISTANCE.s, DIST_EDGE_INIT.s, DIST_PULSE_INIT.s.

In the DIST_PULSE_INIT.s source file, I have first configured the GPIO Pin PF2 I am going to use for pulse generating in order to trigger the distance sensor. I decided to use the PF2 pin of the MCU and configured it as regular port function, digital and output pin. Later on, I have configured Timer0A as 32-bit, periodic, countdown, 1 MHz, timeout interrupt, and I configured the NVIC. I also initialized the TAILR the value to that the pulse should stay as high, and wrote a handler for the interrupt. In this handler, first I drove the PF2 (Trigger) as high for a short time interval (should be higher than 10us, I set it for 255us), and then I drove the PF2 as low for a larger time interval (0x6FFFFF, I have decided it experimentally). By doing so, I have created a pulse train.

In the DIST_EDGE_INIT.s source file, I have configured the GPIO Pin PB4 I am going to use for edge detecting, the echo pin. Therefore, I have configured it as regular port function, digital and input pin. Later, I have configured the Timer1A as 16-bit, capture mode, edge time, count down, detect both edge, and 1 MHz. Therefore, I can detect the edges on the echo pin. By configuring like that, the fourth implementation restriction is satisfied.

The DISTANCE.s source file contains the main distance sensing algorithm. The very first thing to do in this file is that polling the Timer1 RIS until a rising edge is detected. When an edge detected, its TAR value is stored and then Timer1 RIS is polled again to detect the falling edge. When the falling edge is also detected, its TAR value is also stored and these two TAR values are subtracted from each other to calculate the distance. The formula

$$\text{Distance(mm)} = \text{PulseWidth(us)} \times 0.34 / 2$$

is used to determine the distance.

2.1.2. Threshold Setting Submodule

This submodule consists of potentiometer and keypad combined with TM4C123G. The main objective of this submodule is to adjusting the threshold value by getting input from user. In order to adjust the threshold, I have created five source files named POT_ADC.s, POT_ADC_INIT.s, POT_GPIO_INIT.s, KEYPAD_INIT.s, KEYPAD.s.

POT_GPIO_INIT.s source file is used to initialize the GPIO Port Pin that will be used for the potentiometer in that case PE3. The configuration for this pin is input, alternate function selected as ADC, analog.

POT_ADC_INIT.s source file is for initializing the ADC Module of the TM4C123G. The ADC0 is used in this project. ADC0 is configured as sequencer 3, software trigger, AIN0, IE0 and END0, 125ksps. By doing this, the first implementation restriction is satisfied.

POT_ADC.s contains the main routine for the ADC utilization. The routine starts with enabling the sequencer 3 in order to get a measurement, and then poll the RIS in order to indicate the sampling is done. If sampling is done, sample is retrieved from the SSFIFO3 register, and then multiplied and divided by some constants in order to get a value between 0 and 999. Later this value is also multiplied by 10 and then divided by 10 in order to get values between 0 and 990 with increment step as 10.

KEYPAD_INIT.s source file is for initializing the GPIO Pins that will be connected to the 4x4 Keypad. For that purpose, clocks for Port B and Port D are activated. PD0-1-2-3 pins are configured as input, regular port function, and digital. These pins are also pulled up. The interrupt for Port D is also configured as edge sensitive, IEV control, and falling edge. PB0-1-2-3 pins are configured as output, regular port function, and digital.

KEYPAD.S source file contains the algorithm for keypad module. The very first lines of the routine are to detect if SW1 is pressed which indicates return to the main loop of the program. If SW1 is not pressed, then data of input pins are retrieved twice with a 100msec delay in order to debouncing, and then if they are the same value, this value is compared with a value that indicates no keys are pressed. If any key is pressed, then the program continues with determining the which button is pressed part. In order to detect the correct key, output pins are set in order and input is checked. After determining the key, there is a small piece of code that checks if the key is released or not. If it is released, and the pressed key is the 16. key, then the program returns to the potentiometer measurement instead of keypad. If the pressed key is between 10-15, it is ignored. Else if, the pressed key is in between 0-9, the previous second digit stored in the memory location indicated by SECOND_DIGIT alias is shifted to the left, and the previous first digit stored in the memory location indicated by FIRST_DIGIT alias is shifted also to the left. Then, the new digit is stored in the memory location indicated by FIRST_DIGIT alias. After these arrangements, the new threshold value is obtained by multiplying the third digit by 100, multiplying the second digit by 10, and adding the first digit. Then, it is stored in the memory location indicated by THRESHOLD alias. This loop continues until SW1 or 16. key is pressed.

2.1.3. GUI Submodule

This submodule consists of an LCD screen combined with TM4C123G. The main objective of this submodule is to give information related to the system to the user. In order to use the screen, I have created three source files named SCREEN.s, SCREEN_INIT.s, and SCREEN_FUNC.s.

In the SCREEN_INIT.s source file, the first thing to do is initialize the GPIO Port and configure its pins that will be used in this submodule. Therefore, Port A and its pins 2,3,5,6,7 will be configured. First, Port A clock is activated and then pins 2-3-5-6-7 directions are set as output and digital. For pins 2,3,5 SSI function is selected as alternative function. After that, SSI0 clock is activated. PIOSC is used and SSI is configured as 1 MHz by setting CPSDVSR as 8, SCR as 1. SSI is also configured as 8-bit and Freescale mode. After that, LCD settings are done by following the instructions on the project manual. Later, screen is cleared by displaying space character for every possible location on the screen. By doing so, initialization of the screen ends.

In the SCREEN_FUNC.s source file, I have written three different functions called SCR_XY, SCR_CHAR, SCR_BYTE. I want to go over them one by one.

- The SCR_BYTE function is used to send one byte to the screen. For that purpose, first SR is polled until the FIFO is not full, and then the byte is loaded into DR and sent to the screen.
- The SCR_CHAR function is used to send an ASCII character as 5 bytes. For that purpose, I first assure that the screen is in data mode. Then by using the ASCII table that I wrote at the beginning of this source file; I find the

corresponding character by calculating offset. I know that each character is 5 bytes, so the program puts these bytes one by one to the DR and a space character after the ASCII character is outputted. As a final touch, the SR is polled until it is not busy.

- The SCR_XY function is used to set the cursor at the desired location. For that purpose, this function requires Y location in R1, and X location in R0. First, the screen should be taken into the command mode first. Later, the X and Y locations are sent to the screen as one byte through SSI. Before finishing, SSI is taken into the data mode back.

In the SCREEN.s source file, I have written 9 different functions. I am going to go over them one by one. However, some of them are very similar, so that I am going to give details for just one of these similar ones.

- SCREEN_ALWYS function is used to display the things that will be always displayed on the screen no matter what. These things are, 'Meas:' and 'mm' on the first row, 'Thre: ' and 'mm' on the second row, '->' on the fourth row.
- DISPLAY routine is the main routine of the screen submodule actually. When this routine is called, it decides which state the program is in and display accordingly.
- DISP_NORMAL function is the display function of the normal state. It first checks the distance is greater than 999 or not, if it is greater, '>999' will be displayed on the measurement part of the screen. If it is not, then the distance value is displayed. Then the threshold value is displayed, and 'Normal Op.' is displayed on the fourth row of the screen. Later, DISP_BAR function is called.
- DISP_THR is the display function of the threshold setting state and it is very similar to DISP_NORMAL. In the measurement part, it displays '***', and in the threshold part it displays the current threshold value stored in the memory. Later, it displays 'Thr. Adj.' on the fourth row and full of *s in the last row as a progress bar.
- DISP_BRK is the display function of the breaking state. It is almost the same as the DISP_NORMAL. The only difference is on the fourth row 'BRAKE ON' is displayed on this function.
- DISP_BAR is the function that displays the progress bar on the screen. I implemented the same approach on the project manual. It first set the cursor

to the last row and displays CAR as a default. Later, according to the threshold value, and the distance measurement, 10 characters are displayed. Each character represents 100 mm distance. The first thing to check is that, if the hundreds digit of the distance measurement is achieved, every other remaining character will be pipe '|'. If the hundreds digit of the threshold is achieved, 'X' is displayed on that location. Else, '-' character is displayed as a indicator of free space.

- SCREEN_KEY is the function that displays the 'KEYPAD' on the fifth row if the user is adjusting the threshold using the keypad.
- SCREEN_POT is the function that displays the 'POT.' on the fifth row if the user is adjusting the threshold using the potentiometer.
- CLEAR4 is the function that clears the fifth row after the threshold mode. Because in normal mode and preventative breaking mode, this row should be empty.

In the Figure 2, Normal Mode cases can be seen. In the leftmost image, the system is in the Normal mode and there is plenty of empty space for the car. In the middle image, the obstacle is very close to the threshold value, and in the rightmost image it can be seen what happens if the measurement is above 999 mm.

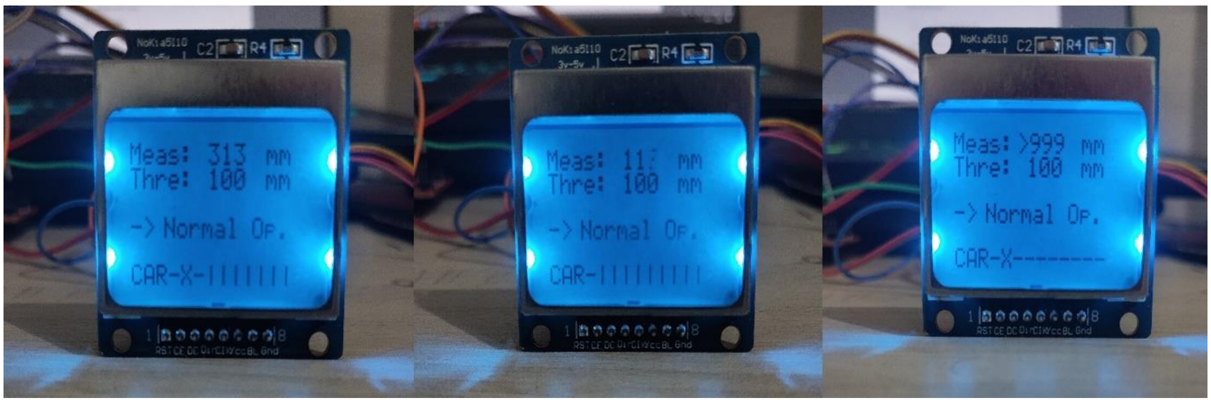

Figure 2 Normal Operation Cases

In the Figure 3, Preventative Breaking Mode cases can be seen. The image on the left shows that, the system is in the Preventative Breaking mode and the distance measurement is below threshold. The image on the left shows that, even if the obstacle moved away from the car, if the user did not press the SW2, it stays on the preventative breaking mode.

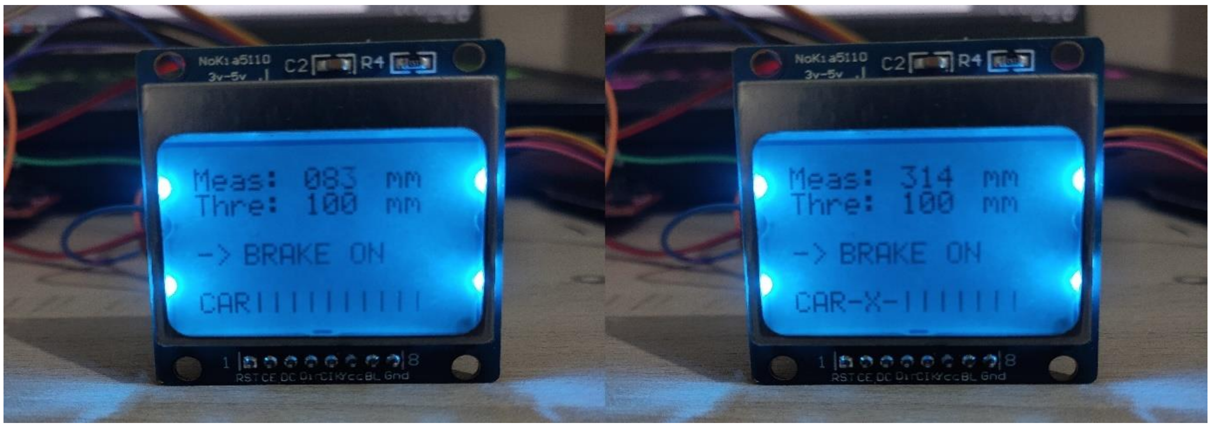

Figure 3 Preventative Breaking Cases

In the Figure 4, Threshold Setting mode cases can be seen. The image on the left shows that, the system is in the Threshold Setting mode and the user is adjusting the threshold value using potentiometer. The image on the left shows that, the user is adjusting the threshold value using the keypad.

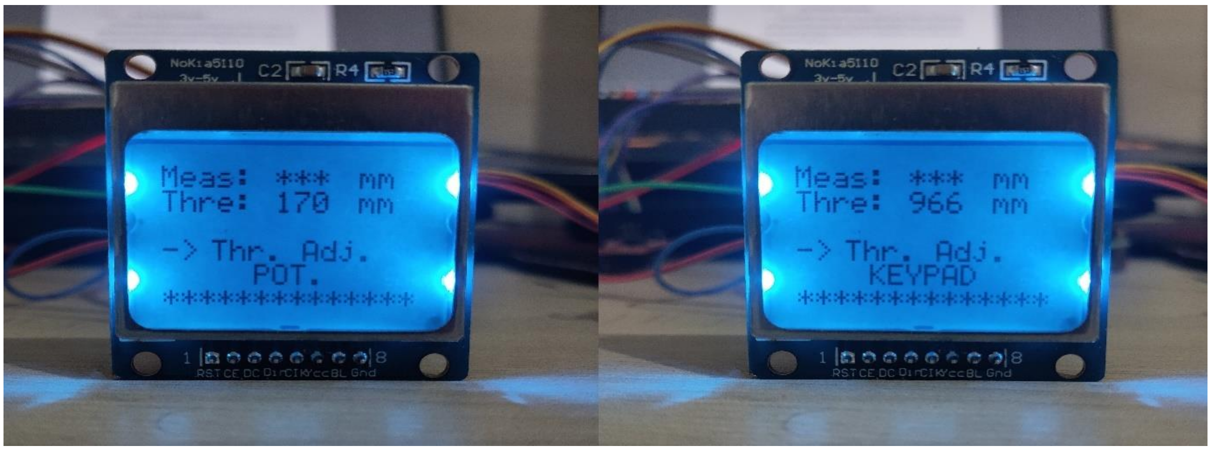

Figure 4 Threshold Adjustment Cases

2.1.4. Movement Submodule

This submodule consists of a step motor combined with TM4C123G. The main objective of this submodule is to drive the step motor. In order to drive it, I have written three different source files named, MOTOR_GPTM_ISR.s, MOTOR_GPIO_INIT.s, MOTOR_GPTM_INIT.s.

In the MOTOR_GPIO_INIT.s source file, I have configured the GPIO pins that I have used for the motor driving. For that purpose, I have used the PC4-5-6-7 pins of the MCU. I configured these pins as output, digital and regular function. Also, at the end of the initialization, I have set the PC5 high to start the motor driving on the second step.

In the MOTOR_GPTM_INIT.s source file, I have configured the timer that I have used for the motor driving. In that case, I have used the Timer5A. I have configured this timer module as 16-bit, periodic, count-down, and 1 MHz. Later, I have written 0x1388 (5000d) into the TAILR register because of the third implementation restriction that states the motor should not advance steps sooner than 5 milliseconds. By doing so, the third implementation restriction is satisfied. After that, NVIC for the GPTM is configured.

In the MOTOR_GPTM_ISR.s source file, the ISR is written. First, RIS is reset by writing 0x01 to ICR and then motor is driven by circulating the high input pin number.

2.1.5. User Switch Submodule

This submodule consists just two user switches on the TM4C123G board. The main objectives of this submodule are determining the RESET signal coming from SW2 and Threshold Mode Signal coming from SW1. For that purpose, I have written a source file named SWITCH_INIT.s. In this file, first, the locked PF0 pin is unlocked. Then PF0 and PF4 are configured as input, regular port functionality, and digital. Also, these two pins are pulled up. After that, interrupt is configured for rising edge. I have decided to not to use NVIC, therefore I am going to poll the RIS in order to detect the interrupt.

3.Conclusion

In this project, I have implemented the methods that I have learned in EE447 course such as utilizing ADC, GPIO and GPIO Interrupts, GPTM and GPTM Interrupts, serial data transfer using SSI, polling and NVIC usage. Using these techniques, I have successfully completed the project and bonus parts. I have improved myself on how to determine the necessities of a given project and how to divide these necessities into sub-tasks, how to co-operate multiple modules together meaningfully. I have practiced and improved my overall skills and coding skills.

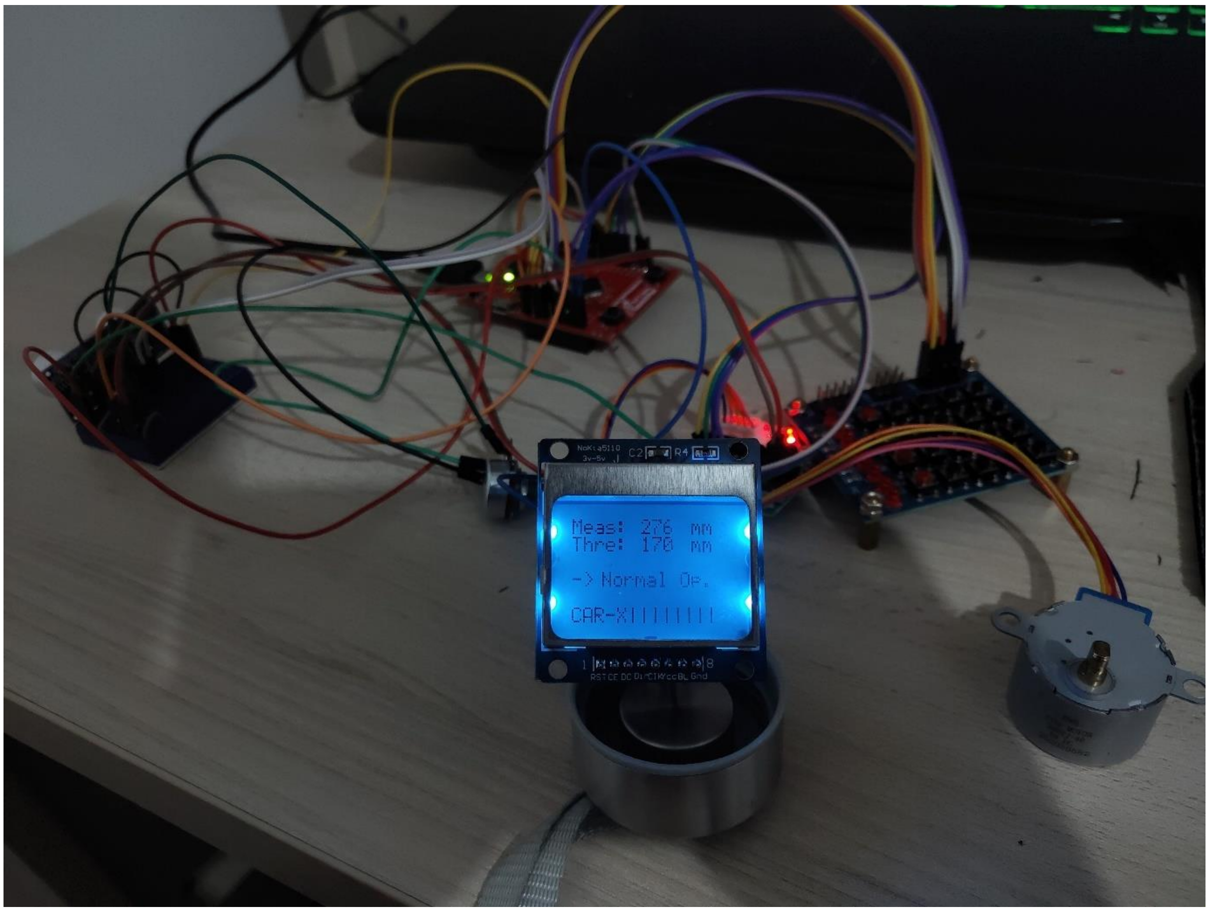

Figure 5 Project Setup

4.References

[1] TI, “Tiva™ c series tm4c123g launchpad evaluation board user's guide.” http://www.ti.com/lit/ug/spmu296/spmu296.pdf.

[2] TI, “Tiva™ tm4c123gh6pm microcontroller data sheet." http://www.ti.com/lit/ds/spms376e/spms376e.pdf.

[3] Project Manual

5.Appendix

Pre-Demo Video