Mustafa Algun
2165777

ODTÜ
METU

EE447

LABORATORY PROJECT REPORT

PARK SENSOR SYSTEM

Table of Contents

1. Hardware Pinout & Connections

1.1 TIVA C Series TM4C123G LaunchPad

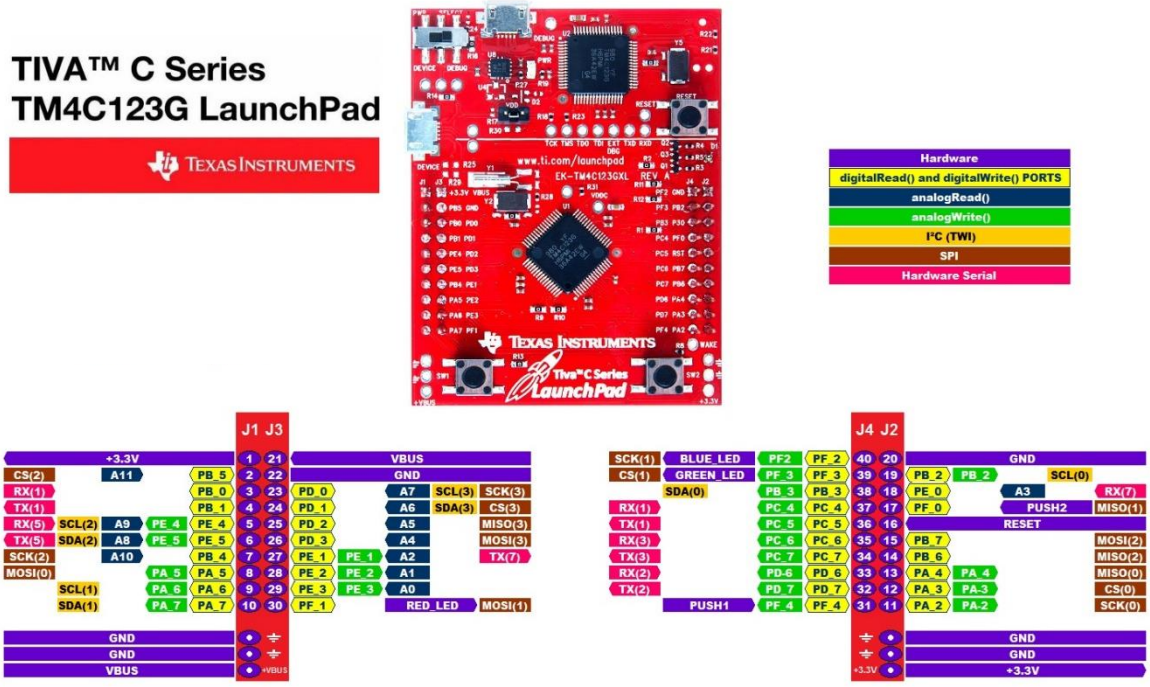
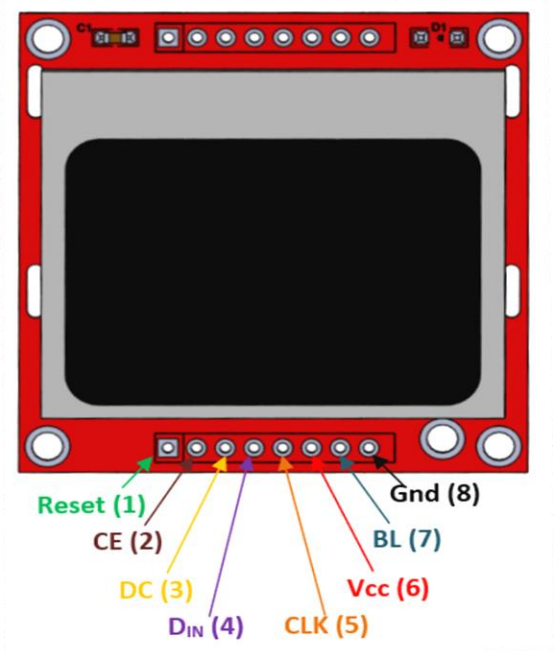


Figure 1: Tiva-C Board pinout

1.2 Nokia 5110 LCD – TIVA-C Board



Nokia 5110	Tiva-C Board
CLK	PA2
CE	PA3
DIN	PA5
DC	PA6
Reset	PA7
BL	V_{BUS}
VCC	3.3V
GND	GND

Figure 2: Nokia 5110 LCD- Tiva-C Board pin connections

1.3 HC-SR04 – TIVA-C Board

HC-SR04	Tiva-C Board
VCC	V_{BUS}
TRIG	PB6
ECHO	PF2
GND	GND

Figure 3: HC-SR04 – Tiva-C Board pin connections

1.4 ULN2003A PCB (For Stepper Motor) – TIVA-C Board

ULN2003A	Tiva-C Board
IN1	PC4
IN2	PC5
IN3	PC6
IN4	PC7
NEG(-)	GND
POS(+)	V_{BUS}

Figure 4: ULN2003A – Tiva-C Board pin connections

1.5 5k Potentiometer – TIVA-C Board

Figure 5: Potentiometer pinout

2. Algorithm Flowchart

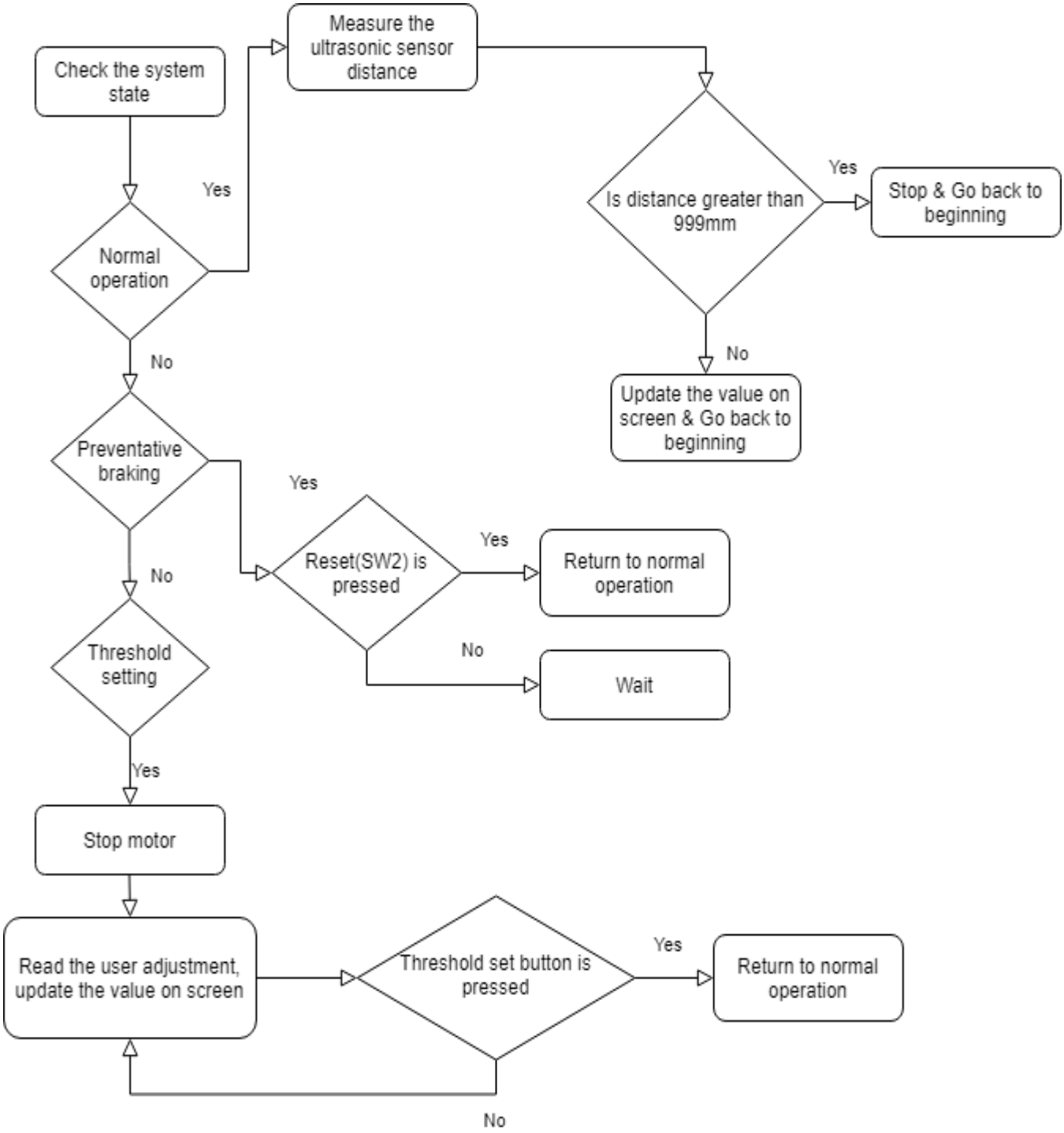


Figure 6: Algorithm Flowchart

3. Project Setup Photos

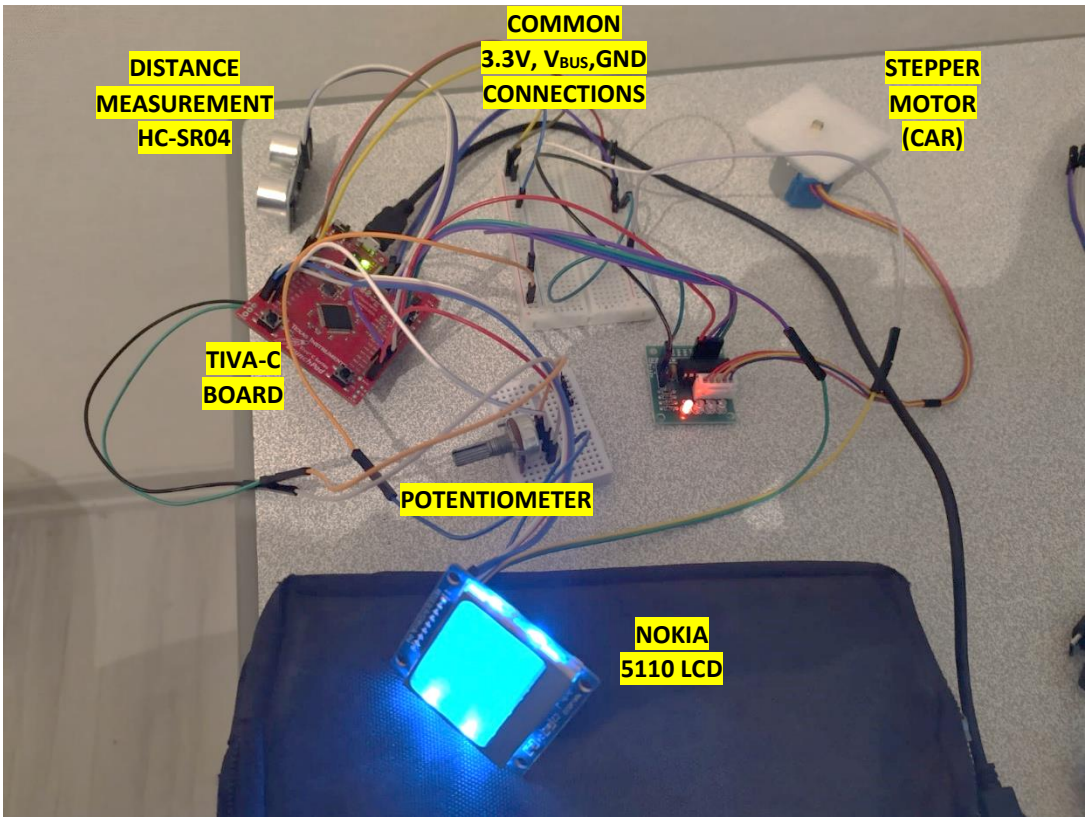


Figure 7: Project setup side view

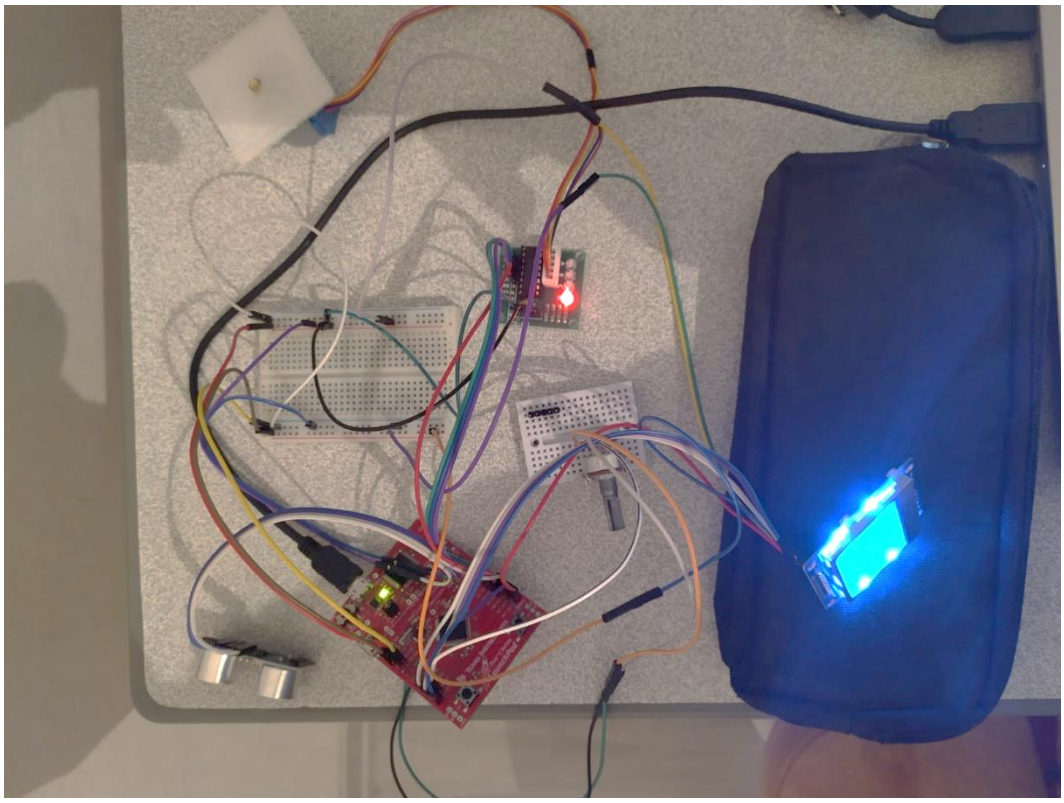


Figure 8: Project setup top view

4. Module Implementations

4.1 Analog to Digital Module

The user makes necessary threshold adjustments through a potentiometer. This potentiometer is read via the ADC module in the Tiva-C board. The ADC module was first initialized with the "INIT_ADC" subroutine to sample analog values and print them on the LCD screen. In this routine, ADC0, SEQ3, and PE3 are configured.

After the configuration, sampling is done in the "**ADC_THRESHOLD**" subroutine. A 0.1V difference is checked between two consecutive data to provide a stable output on the screen in this subroutine. After the ADC data is obtained as a hexadecimal number, the result is converted to a decimal number in a range of [0,999] with equation{1}.

$$Decimal\ Output = (Hexadecimal\ Number) * (90/369) \qquad \text{Equation\{1\}}$$

Finally, the result is updated on the screen by calling the "**THRESHOLD_MODE_OUTPUT**" subroutine in the "**ADC_THRESHOLD**" subroutine.

4.2 Stepper Motor

As stated in the manual, the stepper motor is driven using GPTM interrupts. For this purpose, **Timer2A** is used to generate an interrupt at every time-out. It is configured as 16-bit, periodic mode in the "**PWM_AND_MOTOR_INIT**" subroutine. At every interrupt, "**MY_MOTOR_Handler**" is called. The interrupt priority is set to 5 from the "**NVIC_PRI**" register for the stepper motor. This handler calls the "**FullStepMode**" subroutine, which makes the motor skip to the next step. Then, it clears the Timer2_RIS register by setting PB0 in the Timer2_ICR register. (This register can be controlled from PB0) Another restriction was to keep the time interval between two consecutive steps greater than five milliseconds. Prescaler is set to #15, resulting in a frequency 16MHz/16 = 1MHz. Then, the timer load value is set to 30.000, allowing us to have 30 milliseconds between two consecutive steps. For GPIO, the motor uses pins PC [4:7]. The configuration of Port C is done in the "**INIT_GPIO**" subroutine. These bits are configured as an output, digital, and no alternate function. The default rotation direction is set as a clockwise direction for the stepper motor.

4.3 HC-SR04 Ultrasonic Sensor

This sensor is fed by 5V (V_{BUS}). To conduct a measurement, we need to send a HIGH pulse signal longer than 10μs via TRIGGER pin to the sensor. Then the sensor understands that it is demanded a measurement and sends the result back to the microcontroller via ECHO pin. In this project, I generated an inverted periodic PWM signal using Timer0A. This PWM signal is HIGH for 25.6ms and LOW for 160ms. This way, I could send a trigger signal longer than 10μs and wait long enough to obtain the result at the ECHO pin. TRIGGER is connected to PB6.

After making the sensor work, all that is left is to measure the coming data through ECHO pin. ECHO pin is connected to PF2. In PF2, I used Timer1A. This timer is configured as edge time mode, both edges, and 16-bit as stated in the project requirements. The configuration Timer1A is done in the "**PWM_AND_MOTOR_INIT**" subroutine. After all initializations, measurement is realized in the "**MEASURE_PWM**" subroutine. In this subroutine, the pulse width is measured as shown in Figure 9.

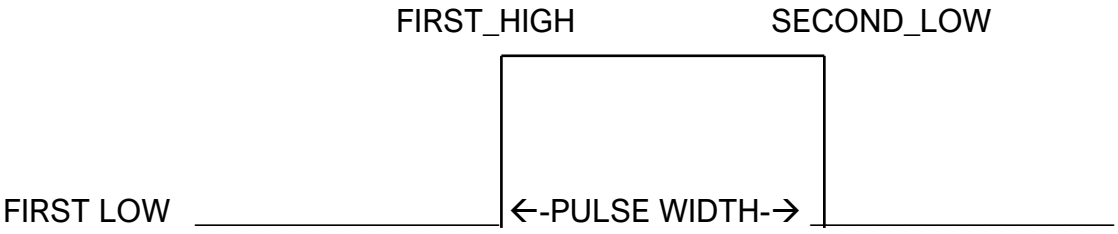


Figure 9: Pulse width measurement

First, I waited for the first low signal. Then I stored the timer_TAR value of the first high in register R2. Then I waited for the second low, and when encountered, I stored the timer_TAR value in register R0. The final value is stored in register R0 by subtracting R0 from R2 as in equation {2}.

$$R0 = R0 - R2 \qquad \text{Equation\{2\}}$$

After calling the "MEASURE_PWM" subroutine in the normal mode, some arithmetical operation is done to convert raw pulse width value into millimeters as in equation {3}.

$$Distance(mm) = Measurement * (0.17)/16 \qquad \text{Equation\{3\}}$$

4.4 Nokia 5110 LCD

SSI protocol is used to interface with this screen. First, I initialized the Nokia display in the "**INIT_NOKIA_5510**" subroutine. In this subroutine, SSI0 and pins PA [2:6] are configured. For GPIO setup,

- PA2, PA3, PA5, and PA6 are set as output and PA4 as input.
- PA2, PA3, PA4, PA5 are set as alternate function enabled and configured to be used in SSI0.
- PA [2:6] are set to digital-enabled.

When GPIO setup is done, I configured SSI0 as below.

- I first started the SSI0 clock.
- I disabled SSI0 during configuration and set it as the master.
- The frequency of the output clock SSInClk is set to 2MHz with the equation {4}

$$\text{SSInClk} = \frac{SysClk}{CPSDVSR*(1+\text{SCR})} \qquad \text{Equation\{4\}}$$

- Then, I reset the LCD memory before the LCD setup by resetting the PA7 (RST) pin for a while and then setting it again.
- For command sending, I set PA6(DC) low first; then, it understands that the coming signal is a command. All data is sent to the screen via SSI in register **R5**.
- I set H=1 for Extended Command Mode, V=0 for Horizontal Addressing.
- I set the contrast V_{OP}.
- I set the temperature control value.
- I set the voltage bias value.
- I set H=0 for basic command mode.
- I configured for normal display mode.
- Then, I cleared the screen for an empty display.

4.4.1 Printing Data on the Display

I printed every value as a string on the screen. To do that, every value to be printed out is converted to a string first. Since a string is a series of characters, I needed to use the ASCII table in Figure 10 for various characters.

```
DCB     0x00, 0x00, 0x00, 0x00, 0x00 ;// 20
DCB     0x00, 0x00, 0x5f, 0x00, 0x00 ;// 21 !
DCB     0x00, 0x07, 0x00, 0x07, 0x00 ;// 22 "
DCB     0x14, 0x7f, 0x14, 0x7f, 0x14 ;// 23 #
DCB     0x24, 0x2a, 0x7f, 0x2a, 0x12 ;// 24 $
DCB     0x23, 0x13, 0x08, 0x64, 0x62 ;// 25 %
DCB     0x36, 0x49, 0x55, 0x22, 0x50 ;// 26 &
DCB     0x00, 0x05, 0x03, 0x00, 0x00 ;// 27 '
DCB     0x00, 0x1c, 0x22, 0x41, 0x00 ;// 28 (
DCB     0x00, 0x41, 0x22, 0x1c, 0x00 ;// 29 )
DCB     0x14, 0x08, 0x3e, 0x08, 0x14 ;// 2a *
DCB     0x08, 0x08, 0x3e, 0x08, 0x08 ;// 2b +
DCB     0x00, 0x50, 0x30, 0x00, 0x00 ;// 2c ,
DCB     0x08, 0x08, 0x08, 0x08, 0x08 ;// 2d -
DCB     0x00, 0x60, 0x60, 0x00, 0x00 ;// 2e .
DCB     0x20, 0x10, 0x08, 0x04, 0x02 ;// 2f /
DCB     0x3e, 0x51, 0x49, 0x45, 0x3e ;// 30 0
DCB     0x00, 0x42, 0x7f, 0x40, 0x00 ;// 31 1
DCB     0x42, 0x61, 0x51, 0x49, 0x46 ;// 32 2
DCB     0x21, 0x41, 0x45, 0x4b, 0x31 ;// 33 3
DCB     0x18, 0x14, 0x12, 0x7f, 0x10 ;// 34 4
DCB     0x27, 0x45, 0x45, 0x45, 0x39 ;// 35 5
DCB     0x3c, 0x4a, 0x49, 0x49, 0x30 ;// 36 6
DCB     0x01, 0x71, 0x09, 0x05, 0x03 ;// 37 7
DCB     0x36, 0x49, 0x49, 0x49, 0x36 ;// 38 8
DCB     0x06, 0x49, 0x49, 0x29, 0x1e ;// 39 9
DCB     0x00, 0x36, 0x36, 0x00, 0x00 ;// 3a :
DCB     0x00, 0x56, 0x36, 0x00, 0x00 ;// 3b ;
DCB     0x08, 0x14, 0x22, 0x41, 0x00 ;// 3c <
DCB     0x14, 0x14, 0x14, 0x14, 0x14 ;// 3d =
DCB     0x00, 0x41, 0x22, 0x14, 0x08 ;// 3e >
DCB     0x02, 0x01, 0x51, 0x09, 0x06 ;// 3f ?
DCB     0x32, 0x49, 0x79, 0x41, 0x3e ;// 40 @
DCB     0x7e, 0x11, 0x11, 0x11, 0x7e ;// 41 A
DCB     0x7f, 0x49, 0x49, 0x49, 0x36 ;// 42 B
DCB     0x3e, 0x41, 0x41, 0x41, 0x22 ;// 43 C
DCB     0x7f, 0x41, 0x41, 0x22, 0x1c ;// 44 D
DCB     0x7f, 0x49, 0x49, 0x49, 0x41 ;// 45 E
DCB     0x7f, 0x09, 0x09, 0x09, 0x01 ;// 46 F
DCB     0x3e, 0x41, 0x49, 0x49, 0x7a ;// 47 G
DCB     0x7f, 0x08, 0x08, 0x08, 0x7f ;// 48 H
DCB     0x00, 0x41, 0x7f, 0x41, 0x00 ;// 49 I
DCB     0x20, 0x40, 0x41, 0x3f, 0x01 ;// 4a J
DCB     0x7f, 0x08, 0x14, 0x22, 0x41 ;// 4b K
DCB     0x7f, 0x40, 0x40, 0x40, 0x40 ;// 4c L
DCB     0x7f, 0x02, 0x0c, 0x02, 0x7f ;// 4d M
DCB     0x7f, 0x04, 0x08, 0x10, 0x7f ;// 4e N
DCB     0x3e, 0x41, 0x41, 0x41, 0x3e ;// 4f O
DCB     0x7f, 0x09, 0x09, 0x09, 0x06 ;// 50 P
DCB     0x3e, 0x41, 0x51, 0x21, 0x5e ;// 51 Q
DCB     0x7f, 0x09, 0x19, 0x29, 0x46 ;// 52 R
DCB     0x46, 0x49, 0x49, 0x49, 0x31 ;// 53 S
DCB     0x01, 0x01, 0x7f, 0x01, 0x01 ;// 54 T
DCB     0x3f, 0x40, 0x40, 0x40, 0x3f ;// 55 U
DCB     0x1f, 0x20, 0x40, 0x20, 0x1f ;// 56 V
DCB     0x3f, 0x40, 0x38, 0x40, 0x3f ;// 57 W
DCB     0x63, 0x14, 0x08, 0x14, 0x63 ;// 58 X
DCB     0x07, 0x08, 0x70, 0x08, 0x07 ;// 59 Y
DCB     0x61, 0x51, 0x49, 0x45, 0x43 ;// 5a Z
DCB     0x00, 0x7f, 0x41, 0x41, 0x00 ;// 5b [
DCB     0x02, 0x04, 0x08, 0x10, 0x20 ;// 5c '\'
DCB     0x00, 0x41, 0x41, 0x7f, 0x00 ;// 5d ]
DCB     0x04, 0x02, 0x01, 0x02, 0x04 ;// 5e ^
DCB     0x40, 0x40, 0x40, 0x40, 0x40 ;// 5f _
DCB     0x00, 0x01, 0x02, 0x04, 0x00 ;// 60 `
DCB     0x20, 0x54, 0x54, 0x54, 0x78 ;// 61 a
DCB     0x7f, 0x48, 0x44, 0x44, 0x38 ;// 62 b
DCB     0x38, 0x44, 0x44, 0x44, 0x20 ;// 63 c
DCB     0x38, 0x44, 0x44, 0x48, 0x7f ;// 64 d
DCB     0x38, 0x54, 0x54, 0x54, 0x18 ;// 65 e
DCB     0x08, 0x7e, 0x09, 0x01, 0x02 ;// 66 f
DCB     0x0c, 0x52, 0x52, 0x52, 0x3e ;// 67 g
DCB     0x7f, 0x08, 0x04, 0x04, 0x78 ;// 68 h
DCB     0x00, 0x44, 0x7d, 0x40, 0x00 ;// 69 i
DCB     0x20, 0x40, 0x44, 0x3d, 0x00 ;// 6a j
DCB     0x7f, 0x10, 0x28, 0x44, 0x00 ;// 6b k
DCB     0x00, 0x41, 0x7f, 0x40, 0x00 ;// 6c l
DCB     0x7c, 0x04, 0x18, 0x04, 0x78 ;// 6d m
DCB     0x7c, 0x08, 0x04, 0x04, 0x78 ;// 6e n
DCB     0x38, 0x44, 0x44, 0x44, 0x38 ;// 6f o
DCB     0x7c, 0x14, 0x14, 0x14, 0x08 ;// 70 p
DCB     0x08, 0x14, 0x14, 0x18, 0x7c ;// 71 q
DCB     0x7c, 0x08, 0x04, 0x04, 0x08 ;// 72 r
DCB     0x48, 0x54, 0x54, 0x54, 0x20 ;// 73 s
DCB     0x04, 0x3f, 0x44, 0x40, 0x20 ;// 74 t
DCB     0x3c, 0x40, 0x40, 0x20, 0x7c ;// 75 u
DCB     0x1c, 0x20, 0x40, 0x20, 0x1c ;// 76 v
DCB     0x3c, 0x40, 0x30, 0x40, 0x3c ;// 77 w
DCB     0x44, 0x28, 0x10, 0x28, 0x44 ;// 78 x
DCB     0x0c, 0x50, 0x50, 0x50, 0x3c ;// 79 y
DCB     0x44, 0x64, 0x54, 0x4c, 0x44 ;// 7a z
DCB     0x00, 0x08, 0x36, 0x41, 0x00 ;// 7b {
DCB     0x00, 0x00, 0x7f, 0x00, 0x00 ;// 7c |
DCB     0x00, 0x41, 0x36, 0x08, 0x00 ;// 7d }
DCB     0x10, 0x08, 0x08, 0x10, 0x08 ;// 7e ~
```

Figure 10: ASCII table for characters in Nokia 5110 LCD

Every character is sent to display in the “**PRINT_CHAR**” subroutine. In this subroutine,

- I pull PA6(DC) high for data transmission.
- Then determine which ASCII character is the value in register R5.
- Finally, send all bytes of the character to the display.

In the “**PRINT_STRING**” subroutine, I send a string of characters using the “PRINT_CHAR” subroutine multiple times until the end (**0x04**) command.

Moreover, to clear the screen, the “**CLEAR_SCREEN**” subroutine is used. This subroutine is loading zeros to all places on the screen to clear it.

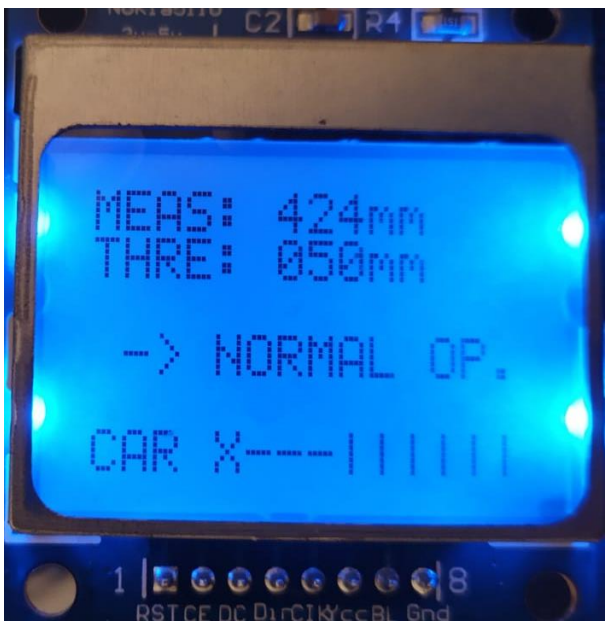
All subroutines that print out the corresponding outputs on the screen for all three modes (normal, preventative, and break) are located in the “**SCREEN_OUTPUTS.s**” file in the project.

4.4.2 Display Explanation

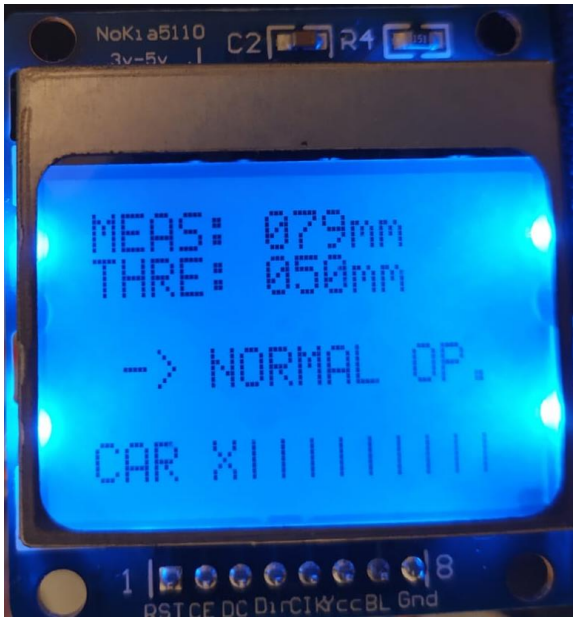
The progress bar is implemented as suggested in the manual.

4.4.2.1 Normal Mode

At the leftmost side of the bar, “**CAR**” represents the HC-SR04 sensor. “**X**” symbol shows the location of the threshold. At the start, the threshold is set to 50mm by default. For example, in Figure 11a, the progress bar tells us that the threshold is set in the [0-99mm] range, and the car is [400mm-499mm] away from the external object. Normal bar update is done in the “**NORMAL_BAR_SELECTION**” subroutine. This subroutine seems long, but its algorithm is quite easy. It first compares the current threshold (register R8) to constants such as “99, 199, 299, 399, 499, 599, 699, 799, and 899” to determine the location of “**X**” symbol. Then, it compares the current distance value (register R4) to the constants up to the threshold to update the normal bar.



(a)

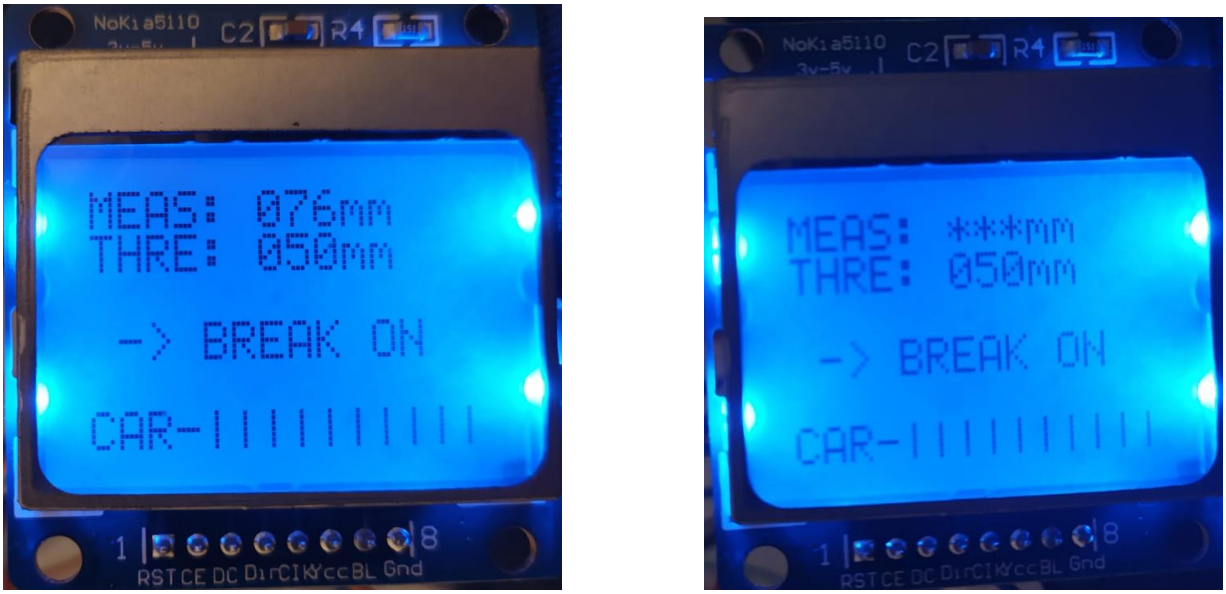


(b)

Figure 11: Nokia 5510 LCD, normal mode

4.4.2.2 Preventative Break Mode

This mode is activated when the object comes closer to the car than the threshold. In this mode, the operation name changes from “-> NORMAL OP.” to “-> BREAK ON” as in Figure 12.



(a) (b)

Figure 12: Nokia 5510 LCD, preventative break mode

Let us assume the system is in preventative break mode as in Figure 12a, and the object is still closer to the car less than the threshold. In this case, if the user presses SW2, since the distance is less than the threshold, the system quickly enters into normal mode and goes back to the preventative break mode. To show this case, “MEAS: ***mm” is shown on the screen as in Figure 12b.

4.4.2.3 Threshold Setting Mode

In this mode, measurement stops. Therefore, we “MEAS: ***” at the measurement part as in Figure 13. The user changes the threshold via the potentiometer, and the updated value is printed out on the screen continuously until the user presses the SW1 button again.

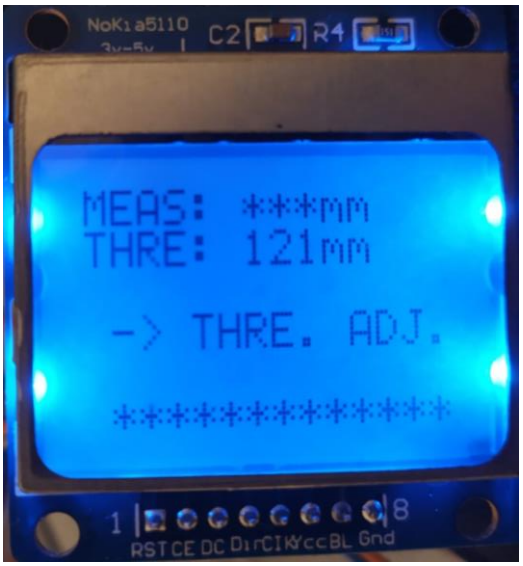


Figure 13: Nokia 5510 LCD, threshold setting mode

4.5 SW1 and SW2 Switches

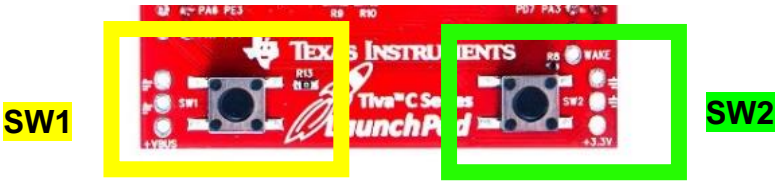


Figure 14: SW1 and SW2 switches

SW1 is connected to PF4, and SW2 is connected to PF0. To detect if these buttons are pressed, GPIO interrupts are used in this project. To use these switches, first, I configured port F in the "**INIT_GPIO**" subroutine. PF0 is locked by default. Therefore I unlocked this pin in this configuration. I connected a pull-up resistor to each pin, making these pins HIGH unless SW1 & SW2 buttons are pressed. Then, I configured their interrupts as edge sensitive, not both edges, and falling edge. So, whenever any switch is pressed, PF0 and PF4 data values become LOW, which triggers the GPIO interrupt for port F. The interrupt priority is set to 2 from the **NVIC_PRI7** register so that switches have a higher priority than the stepper motor.

4.5.1 SW1 Switch

The user uses SW1 switch to start setting the threshold value. To use this switch, first, I decided from which pin the interrupt is triggered because both switches use the GPIO port F interrupt. This is done in the "**My_GPIO_PORTF_Handler**" subroutine. In this subroutine, I first check the "**GPIO_PORTF_MIS**" register to determine which button is pressed. After understanding that it was SW1, the motor is stopped by disabling its timer "Timer2A," and the mode is set to threshold mode by making register R9 = 2. Then I clear RIS and MIS registers by writing to the ICR register for the next interrupt. To understand whether SW1 is pressed twice, I count the value in register R11. If it is zero, this means SW1 pressed only once. If it is one, this means that SW1 is pressed twice. If SW1 is pressed twice, I set the threshold as the last ADC output and change the mode to normal mode.

4.5.2 SW2 Switch

This switch is used to ignore and disable the preventative break mode. When the object comes closer to the car less than the threshold, preventative break mode is activated, and the car stops. When the object moves away, the preventative break mode is still on. To reset this mode and go back to the normal mode, the user is expected to press SW2. In the "**My_GPIO_PORTF_Handler**" subroutine, I first check the "**GPIO_PORTF_MIS**" register to determine which button is pressed. After understanding that it was SW2, I clear RIS and MIS registers by writing to the ICR register for the next interrupt. Then, I enable the stepper motor and distance measurement again and set the mode to normal mode.

5. Assembly Files & Corresponding Subroutines List

This table is prepared to provide a better understanding to the reader.

Assembly File (.s)	Subroutines
INIT_GPIO.s	INIT_GPIO
INIT_ADC.s	INIT_ADC
My_GPIO_PORTF_Handler.s	My_GPIO_PORTF_Handler
FullStepMode.s	FullStepMode
MEASURE_PULSE_WIDTH.s	MEASURE_PWM
MY_MOTOR_Handler.s	MY_MOTOR_Handler
PWM_AND_MOTOR_INIT.s	PWM_AND_MOTOR_INIT
CONVERT_PRINT.s	CONVERT_PRINT
BAR_SELECTION.s	NORMAL_BAR_SELECTION
NOKIA_5110_LCD	INIT_NOKIA_5510 SSI_SEND PRINT_CHAR PRINT_STRING CLEAR_SCREEN
ADC_THRESHOLD.s	ADC_THRESHOLD
SCREEN_OUTPUTS.s	NORMAL_MODE_OUTPUT PREVENTATIVE_BREAK_MODE_OUTPUT THRESHOLD_MODE_OUTPUT

Table 1: Assembly file vs. subroutines

6. General-Purpose Register Contents

R0 → Distance measurement data

R1→ General purpose

R2→ General purpose

R3→ General purpose

R4 → Distance measurement value & value to be converted into ASCII character

R5 → To send data through SSI

R6→ General purpose

R7→ General purpose

R8 → Threshold value

R9 → Mode selection

R10→ General purpose

R11 → #Times SW2 pressed

R12→ General purpose

7. Pre-Demo Video Link & Explanations

In the pre-demo video, I showed but forgot to mention that I am polling to check the operation mode in the main. For motor, I am using GPTM interrupts with an interrupt priority “5”. For switches, I am using GPIO interrupts with an interrupt priority “2”. Since I recorded this video with my left hand and since the camera does not have an optic stabilizer, the video is too shaky. I am sorry for the inconvenience.

Pre-demo video Youtube link: https://youtu.be/nXpKGj_k1U0