[22-23_SEM1_CE451_A] – Embedded Systems

Papa Kobina Aferi (11622023) | Nana Kwaku Okai-Mensah (14142023)

PROJECT REPORT

P3: BIDIRECTIONAL GUEST COUNTER

December 16, 2022

Table of Contents

Project Overview 3
Hardware Description 3
Integration of Components 5
Software Description 6
Reflection 7
Conclusion ..... **Error! Bookmark not defined.**
Appendix A 10
Appendix B 11

Project Overview

Usually, it becomes a tedious task at events to keep track of the number of guests present. This is because guests come in at varying times during the span of an event. It is very necessary to know the number of guests present at an event as it aids in making immediate decisions, such as knowing the number of gift bags to distribute. Manually getting a headcount could be both time and energy consuming, however, having a system in place that accounts for the number of guests present at any given time during an event could prove very helpful.

The system should be able to allow the host of the event to enter the number of guests expected to be present at the event and give an alert when that limit is exceeded. The system would include two sensors that should detect the entry and exit of guests, respectively. The system should be able to count the number of guests present at an event and display it on a screen. Lastly, the system should be able to be reset upon request by the event host (user).

Hardware Description

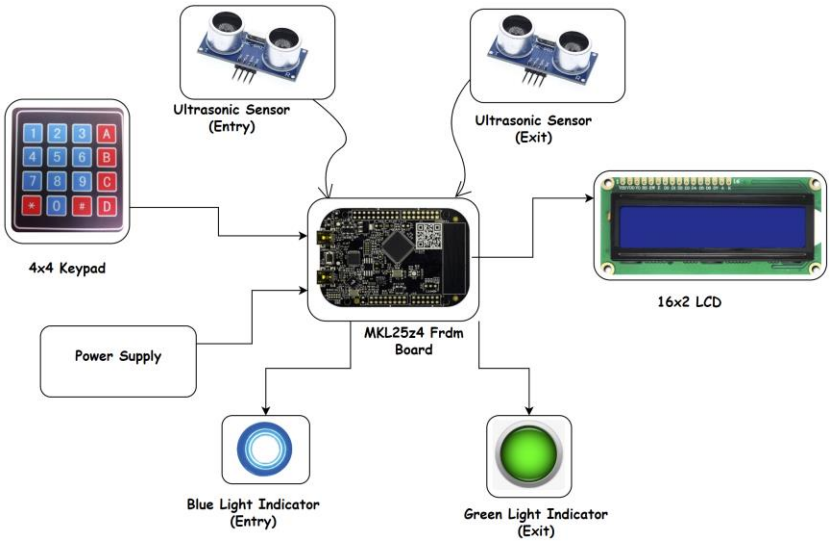

Figure 1: Functional Diagram of the Bidirectional Guest Counter

As seen from figure 1, the components used in this project were two ultrasonic sensors, a keypad, a 16x2 LCD screen, and two indicator lights (blue and green), all connected to the microcontroller (MKL25Z128VLK4) freedom board. Each ultrasonic sensor was designated to the point of entry and exit, respectively. The blue indicator light was used to signal that a guest has entered the room by turning on for about a second and then going off. Analogously, green indicator light was used to signal that a guest is leaving. The LCD screen was used as the user interface to both take in commands from the user and display information.

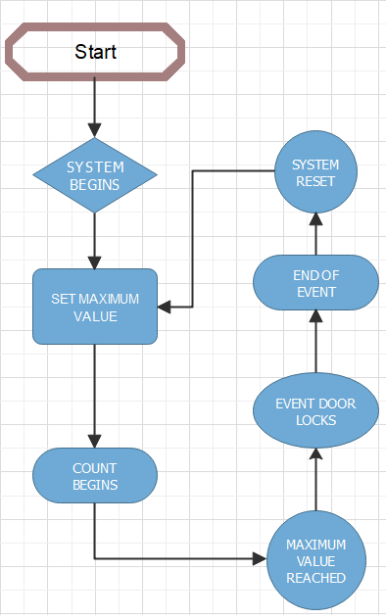

Figure 2: Flowchart of the system

Figure 2 shows the simplified operation of the bidirectional event counter system. The system is booted at the beginning which starts the system. Various writings will be displayed to the user on a 16x2 LCD module, directing the user to set the maximum number of people attending the scheduled event. As the maximum number has been set for the event, the system begins to count the number of people passing through the entry and it is being displayed on the LCD module to the user. At the instance the maximum number is reached, the event door closes to regulate the number of people in the event. This allows for tracking the number of people in the event. At the end of event, the user resets the system which allows for another user, assuming the event center is fully booked for different periods, to set their maximum number of people expecting to be present at the event.

This happens at the Count Begins in the main flow chart. An ultrasonic sensor is placed at the entry which adds to the counter and a second ultrasonic sensor is placed at the exit which subtracts from the counter. The counter stores the value from addition and subtraction and displays the value in real time to the user.

Integration of Components

1. Ultrasonic sensor

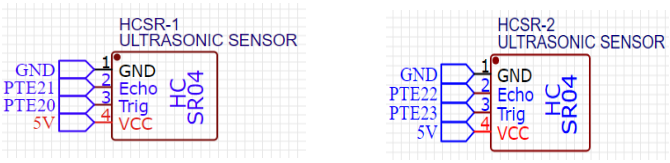

Figure 3: Ultrasonic sensors pinout connections

The HCSR-04 model was used (see Figure 3). Five volts was used to power each of the ultrasonic sensors via the VCC pin. The entry sensor's echo and trigger pins were connected to PORTE 20 and 21 of the microcontroller, while the exit sensor's echo and trigger pins were connected to PORTE 22 and 23. The trigger and pins of the ultrasonic sensor connected were used based on the corresponding PWM configurations in the reference manual [1]. The ultrasonic sensors were made to read and record distances every 30ms. These distances were calibrated and used in setting the range of acceptable distances for which a guest is considered to have passed through a door. The distances were coded to be read in the millimeters. The sensors were placed on the top frame of each door (exit and entry).

2. 16x2 LCD Module

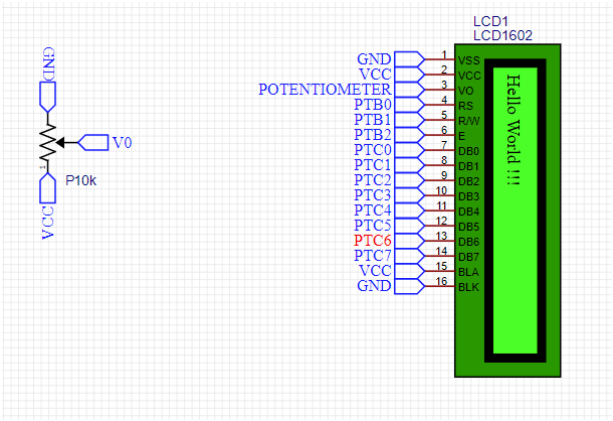

Figure 4: 16x2 LCD module pinout connections

Figure 4 shows the connections of the LCD screen which is being used in 8-bit mode. Data from the LCD module is sent and received by the MCU. The keypad is wired in an active-low configuration to the LCD. To prevent floating pins, pull-up resistors are set up in software to work with the keypad. There are no established protocols or interfaces for the keypad to communicate with the development board. The data pins (D7-D0) of the LCD were connected to microcontroller pins PORTC pins PTC7-PTC0. The RS, RW, and enable pins were connected to PORTB pins, PTB0, PTB1, and PTB2, respectively. The LCD was used to display the maximum number of guests as well as displaying count of the number of guests presently in the room.

3. 4x4 matrix Keypad

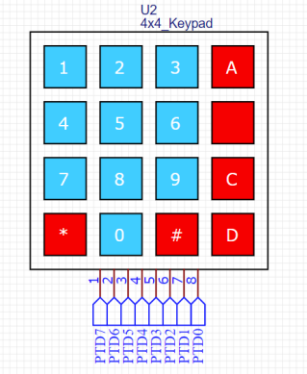

Figure 5: 4x4 matrix keypad

Figure 5 shows the connections of the keypad. The rows of the keypad are configured as active low while the columns of the keypad are set to high using the pull-up resistors in the software. PORTD pins PTD7-PTD4 were connected to the first 4 rows, while PTD3-PTD0 were connected to the remaining 4 columns. The keypad was used to enter the maximum number of guests to be expected in the room.

4. Light emitting diodes (LEDs)

LEDs were used as indicator lights. The blue LED was used to signal entry, and the green LED, exit. The blue and green LEDs were connected to the pins PTB7 and PTB8 of the microcontroller, respectively. They were set up as GPIO output pins.

Software Description

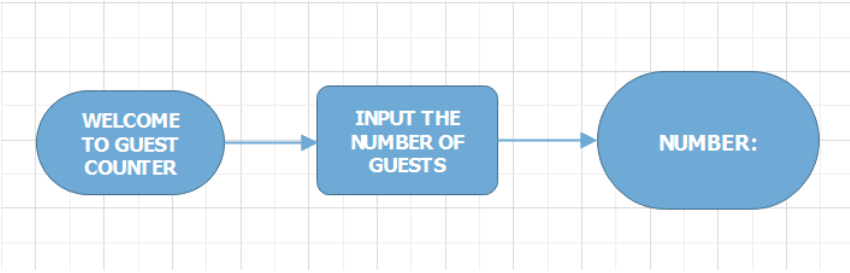

Figure 6: Sequence of the intro function and enterMAX function

Figure 6 shows how the program is introduced to the user. The keypad as well as the LCD module was used in these functions to introduce the user to the system and prompt the user to set the maximum value. The value set by the user is displayed.

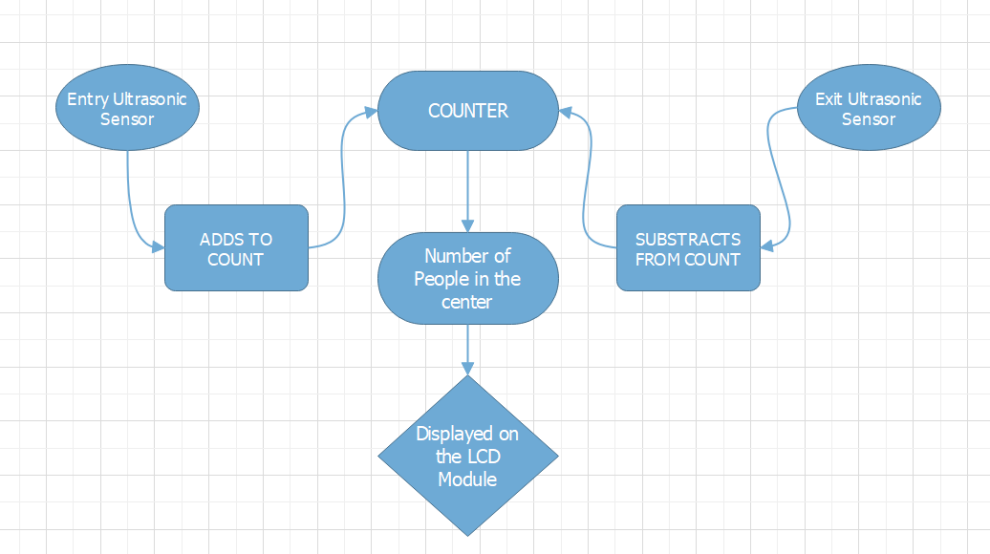

Figure 7: The counting process of the system

There were a chain of initialization functions. There were initialization functions for the keypad, LCD, LEDs, and PWM pins. The input given by the user was a character string, therefore in order to user that value in the code in the counting process, it had to be converted into an integer. So, there was a function meant for that.

However, the huge chunk of the code resided counting computation was done in the TPM interrupt. We wrote two interrupt functions: void TMP1 IRQHandler() and TPM2 IRQHandler. The former was used to detect (by incrementing the global counting variable) the number of guests entering the room, while the latter was used to count detect when a guests leaves (by decrementing the global counting variable). When the counting variable reaches the maximum number of guests expected, the sensors stop recording.

Also, one significant feature was to prevent the sensors from recording multiple times, while it detects someone standing at the door. The sensors were programmed to wait for the guest to either leave or enter before reading.

Reflection and Conclusion

If we were to design this again, we would include an dc motor (actuator) that automatically closes the door without the user having to manually operate the door, when the maximum set value in the system is reached. Also, we would include a Bluetooth module to send commands to manually close the door or open based on the preference and decision of the user. Include a buzzer to sound an alarm when the maximum number is reached instead of the user having to frequently check if the maximum value is reached or exceeded.

One thing that was surprisingly easy was calibrating the ultrasonic sensors. However, integrating both ultrasonic sensors to work concurrently with the timer was surprisingly difficult. It absorbed most of our scheduled time allocated to other parts of the work. Getting both sensors to work concurrently was an integral part of the work, otherwise the other parts of the code would have no better use in the implementation of the project. The ultrasonic sensors gave the other functions usefulness. Also, it was our first time integrating an ultrasonic sensor with the Freedom MKL25z board which was not easy to achieve.

The easiest parts of this project were writing to the LCD and printing on the screen. These are tasks that have been performed repeatedly in the course so implementing them in this project seemed rather effortless.

References

[1] F. Semiconductor Inc, “Kinetis KL25: 48MHz Cortex-M0+ 32-128KB Flash 32-80 pin,” 2012.

[2] F. Semiconductor Inc, “Kinetis KL25 Sub-Family 48 MHz Cortex-M0+ Based Microcontroller with USB,” 2012. [Online]. Available: http://www.freescale.com

Appendix A

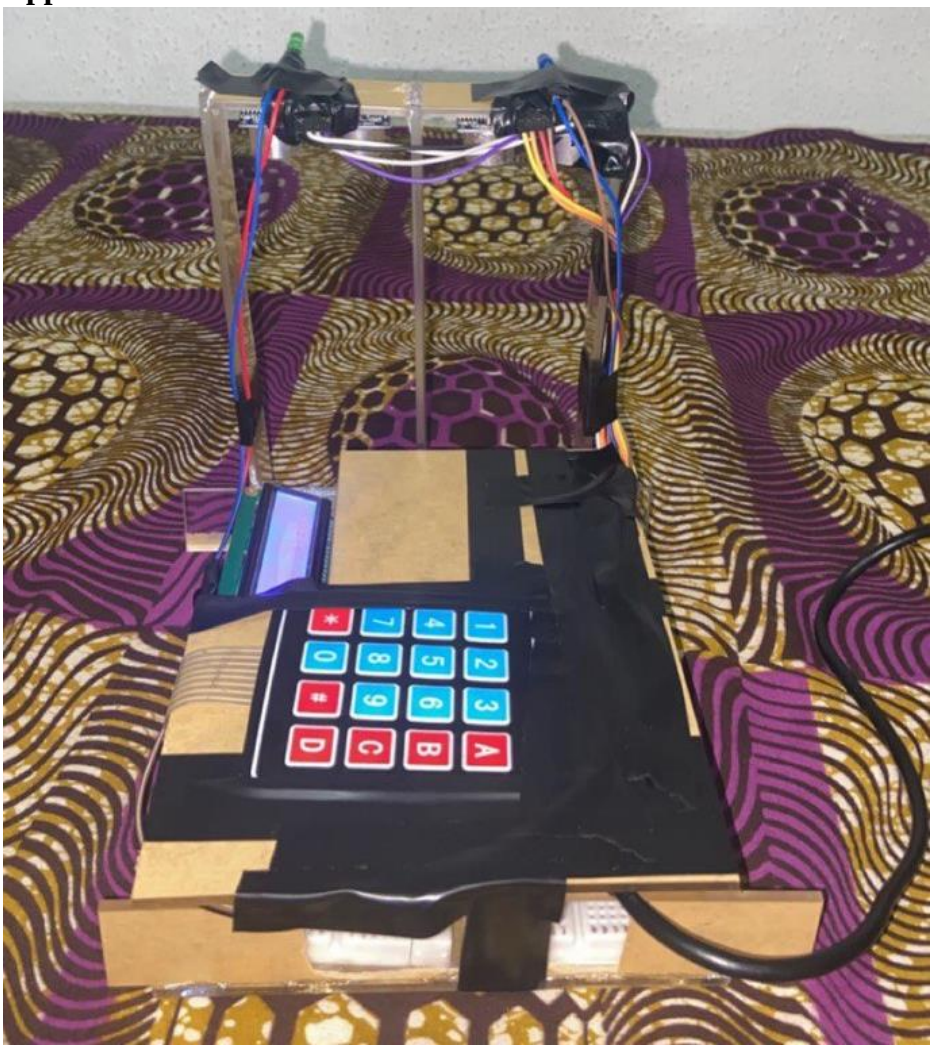

Figure 8

Appendix B

Code

Main.c

```
#include <mkl25z4.h>
#include <stdio.h>
#include <stdint.h>
#include "keypad.h"
#include "lcd.h"
#include "timer.h"
#include "led.h"
#include "stdlib.h"

void init_Timer1();
void init_Timer2();
void init_pin();
void keypad();
void enterMAX();
void intro();
uint16_t countnum();
char* my_itoa(uint16_t number);
void counting();
void numReached();

uint8_t read_distance = 0;
uint8_t read_distance2 = 0;
uint8_t row = 2;
uint8_t column = 1;
uint8_t start =0;

//input capture on rising & falling
uint16_t g_dist=0; //global variable to hold distance.
uint16_t g_dist2=0;
uint8_t counter2 = 0; // increments towards max limit
char max_num[3];
int counter1 = 0; // increments with each key press
uint16_t number;
uint8_t canRead = 0;
uint8_t canRead2 = 0;

int main(){
	LCD_activate_pins();
	init_PWMpins();
	init_Timer1();
	init_Timer2();
	init_LED();
	intro();
	while(1){
	}
}
```

```
uint16_t countnum(){    // convert the number in max_num array into an integer
	number = atoi(max_num);
	return number;
}


void intro(){
	char abc[] = "WELCOME TO";
	Screen_display(abc);
	LCD_write_nextln();
	LCD_write_string("GUEST COUNTER");
	delay_ms(3000);
	enterMAX();
}

void enterMAX(){
	memset(max_num,0,3);
	read_distance=0;
	read_distance2=0;
	counter1=0;
	counter2=0;
	start = 0;
	char abc[] = "INPUT THE NUMBER";
	Screen_display(abc);
	LCD_write_nextln();
	LCD_write_string("OF GUESTS");
	delay_ms(2000);
	char bac[] = "NUMBER: ";
	Screen_display(bac);
	delay_ms(200);
	keypad();
}

void keypad(){

	activate_keypad();
	//set up rows to be output
	PTD->PDDR |=  MASK(R1) ;
	PTD->PDDR |=  MASK(R2) ;
	PTD->PDDR |=  MASK(R3) ;
	PTD->PDDR |=  MASK(R4) ;

	//set up columns to be input
	PTD->PDDR &= ~MASK(C1) ;
	PTD->PDDR &= ~MASK(C2) ;
	PTD->PDDR &= ~MASK(C3) ;
	PTD->PDDR &= ~MASK(C4) ;


	while(1) {

		// Take row 1 to low and write the various keypad configurations
		PTD->PDOR &= ~MASK(R1);
		PTD->PDOR |= MASK(R2);
```

```
		PTD->PDOR |= MASK(R3);
		PTD->PDOR |= MASK(R4);
		delay_us(2); // a 2 pulse (us) timing delay_ms after setting signal to
low

		if((~PTD->PDIR) & MASK(C1)){
			LCD_write_char('1');
			max_num[counter1] = '1';
			counter1++;
			delay_ms(600);
		}

		else if((~PTD->PDIR) & MASK(C2)){
			LCD_write_char('2');
			max_num[counter1] = '2';
			counter1++;
			delay_ms(600);

		}

		else if((~PTD->PDIR) & MASK(C3)){
			LCD_write_char('3');
			max_num[counter1] = '3';
			counter1++;
			delay_ms(600);
		}

		else if((~PTD->PDIR) & MASK(C4)){

		}

		// Take row 2 to low and write the various keypad configurations
		PTD->PDOR |= MASK(R1);
		PTD->PDOR &= ~MASK(R2);
		PTD->PDOR |= MASK(R3);
		PTD->PDOR |= MASK(R4);
		delay_us(2);

		if((~PTD->PDIR) & MASK(C1)){
			LCD_write_char('4');
			max_num[counter1] = '4';
			counter1++;
			delay_ms(600);
		}

		else if((~PTD->PDIR) & MASK(C2)){
			LCD_write_char('5');
			max_num[counter1] = '5';
			counter1++;
			delay_ms(600);
		}

		else if((~PTD->PDIR) & MASK(C3)){
			LCD_write_char('6');
			max_num[counter1] = '6';
```

```
		counter1++;
		delay_ms(300);
	}

	else if((~PTD->PDIR) & MASK(C4)){

	}

	// Take row 3 to low and write the various keypad configurations
	PTD->PDOR |= MASK(R1);
	PTD->PDOR |= MASK(R2);
	PTD->PDOR &= ~MASK(R3);
	PTD->PDOR |= MASK(R4);
	delay_us(2);

	if((~PTD->PDIR) & MASK(C1)){
		LCD_write_char('7');
		max_num[counter1] = '7';
		counter1++;
		delay_ms(600);
	}

	else if((~PTD->PDIR) & MASK(C2)){
		LCD_write_char('8');
		max_num[counter1] = '8';
		counter1++;
		delay_ms(600);
	}

	else if((~PTD->PDIR) & MASK(C3)){
		LCD_write_char('9');
		max_num[counter1] = '9';
		counter1++;
		delay_ms(600);
	}

	else if((~PTD->PDIR) & MASK(C4)){

	}

	// Take row 4 to low and write the various keypad configurations
	PTD->PDOR |= MASK(R1);
	PTD->PDOR |= MASK(R2);
	PTD->PDOR |= MASK(R3);
	PTD->PDOR &= ~MASK(R4);
	delay_us(2);

	if((~PTD->PDIR) & MASK(C1)){
		//reset
		delay_ms(600);
		enterMAX();
		read_distance=0;
		read_distance2=0;
	}
```

```
            else if((~PTD->PDIR) & MASK(C2)){
                  LCD_write_char('0');
                  max_num[counter1] = '0';
                  counter1++;
                  delay_ms(600);
            }

            else if((~PTD->PDIR) & MASK(C3)){
                  delay_ms(600);
                  read_distance = 1;
                  start =1;
                countnum();
                  counting();
                  canRead=1;
            }

            else if((~PTD->PDIR) & MASK(C4)){

            }

      }

}


void init_Timer1(){//use channel0 for signal generation, channel 1 for input capture
      //Clock gate
      SIM->SCGC6 |=SIM_SCGC6_TPM1_MASK; //TPM1 channel 0
      //Select clock source in SIM_SOPT //system clock
      SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1) ;
      //Timeout is is 30ms
      TPM1->MOD= 4920; //use value -1
      //Channel 0 PWM: MSB-A==10 ELSB-A 10, //PWM output, interrupts not needed.
      TPM1->CONTROLS[0].CnSC |= TPM_CnSC_MSB(1) |TPM_CnSC_ELSB(1) ;
      TPM1->CONTROLS[0].CnV |= 2 ; //For trigger of 20us, CnV=3.2
      //input capture: MSB-A==00 ELSB-A 11; for rising & falling edge
      TPM1->CONTROLS[1].CnSC |= TPM_CnSC_ELSA(1) |TPM_CnSC_ELSB(1) ;
      TPM1->CONTROLS[1].CnSC |= TPM_CnSC_CHF_MASK | TPM_CnSC_CHIE_MASK ; //enable
      TPM1->SC |= TPM_SC_TOF_MASK | TPM_SC_PS(7) | TPM_SC_TOIE_MASK ;
      TPM1->SC |= TPM_SC_CMOD(1); //enable internal clock to run

      NVIC_ClearPendingIRQ(TPM1_IRQn);
      NVIC_SetPriority(TPM1_IRQn, 3);
      NVIC_EnableIRQ(TPM1_IRQn);
}

void init_Timer2(){//use channel0 for signal generation, channel 1 for input capture
      //Clock gate
      SIM->SCGC6 |=SIM_SCGC6_TPM2_MASK; //TPM2 channel 0
      //Select clock source in SIM_SOPT //system clock
      SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1) ;
      //Timeout is is 30ms
      TPM2->MOD= 4920;
      //Channel 0 PWM: MSB-A==10 ELSB-A 10, //PWM output, interrupts not needed.
      TPM2->CONTROLS[0].CnSC |= TPM_CnSC_MSB(1) |TPM_CnSC_ELSB(1) ;
```

```
	TPM2->CONTROLS[0].CnV |= 2 ; //For trigger of 20us, CnV=3.2
	//input capture: MSB-A==00 ELSB-A 11; for rising & falling edge
	TPM2->CONTROLS[1].CnSC |= TPM_CnSC_ELSA(1) |TPM_CnSC_ELSB(1) ;
	TPM2->CONTROLS[1].CnSC |= TPM_CnSC_CHF_MASK | TPM_CnSC_CHIE_MASK ; //enable
	TPM2->SC |= TPM_SC_TOF_MASK | TPM_SC_PS(7) | TPM_SC_TOIE_MASK ;
	TPM2->SC |= TPM_SC_CMOD(1); //enable internal clock to run

	NVIC_ClearPendingIRQ(TPM2_IRQn);
	NVIC_SetPriority(TPM2_IRQn, 3);
	NVIC_EnableIRQ(TPM2_IRQn);
}


void TPM1_IRQHandler(){
	static int ctr=0;
	static unsigned int previous=0;
	unsigned int current=0;
	static unsigned int interval=0;
	uint8_t num = countnum();


	if (TPM1->STATUS & TPM_STATUS_CH1F_MASK){ //input capture occured?
		current=TPM1->CONTROLS[1].CnV;
		current |= (ctr <<13);//add no of overflows. Each ctr tick is 2^16
		interval = current-previous;
		previous=current;
		TPM1->CONTROLS[1].CnSC |=TPM_CnSC_CHF_MASK; //clear flag
	}
		if (TPM1->SC & TPM_SC_TOF_MASK){
			if(counter2 < num && start == 1) {
			ctr++; //a timer overflow occurred.
			if (!( ctr %10)){ //check every 10 times
				g_dist=interval;

				if (read_distance == 1){
					if (interval<68 && canRead){
						counter2++;
						PTB->PDOR |= MASK(RED_LED);
						delay_ms(400);
						PTB->PDOR &= ~MASK(RED_LED);
						read_distance2 = 1;
						//printf("discard %d\n", g_dist);
						//uint16_t num = countnum();
						char stor[3]; //create an empty string to
store number
						sprintf(stor, "%d", counter2); //make the
number into string using sprintf function
						LCD_set_cursor(row,column);
						delay_ms(300);
						LCD_write_string(stor);

						//counter2++;
						canRead = 0;
						//}
					}
```

```
                                if(interval >= 100) {
                                        canRead = 1;
                                }

                                else
                                        printf("distance in mm=%d\n", g_dist);
                        }
                }
        }
//      if(counter2 == num && start == 1) {
//              numReached();
//}
        TPM1->SC |= TPM_SC_TOF_MASK ; //clear the interrupt
}

void TPM2_IRQHandler(){
        static int ctr=0;
        static unsigned int previous=0;
        unsigned int current=0;
        static unsigned int interval=0;
        //uint16_t num = countnum();


        if (TPM2->STATUS & TPM_STATUS_CH1F_MASK){ //input capture occured?
                current=TPM2->CONTROLS[1].CnV;
                current |= (ctr <<13);//add no of overflows. Each ctr tick is 2^16
                interval = current-previous;
                previous=current;
                TPM2->CONTROLS[1].CnSC |=TPM_CnSC_CHF_MASK; //clear flag
        }
        if(counter2 != 0) {
                if (TPM2->SC & TPM_SC_TOF_MASK){
                        ctr++; //a timer overflow occurred.
                        if (!( ctr %10)){ //check every 10 times
                                g_dist2=interval; //needed to multiply bt 1.047. Just
approx
                                /**
                                 * actual calculation:
                                 * Clock freq is 20,971,520
                                 * PS= 128
                                 * => 20.9M/128 -->1s
                                 * time for cnv--> cnv/20.9M*128 <=T
                                 * dist = speed * time
                                 * = 340 * T
                                 * = (double distance and in meters)
                                 * dist in mm = (340*T)/2*1000
                                 * = cnv *1.038 mm
                                 * (or 1.047 if using speed as 343m/s)
                                 */
                                if (read_distance2 == 1){
                                        if (interval<68 && canRead2){
                                                counter2--;
                                                PTB->PDOR |= MASK(GREEN_LED);
                                                delay_ms(400);
```

```
                                        PTB->PDOR &= ~MASK(GREEN_LED);
                                        //printf("discard %d\n", g_dist);
                                        //uint16_t num = countnum();
                                        char stor[3]; //create an empty string to
store number
                                        sprintf(stor, "%d", counter2); //make the
number into string using sprintf function
                                        LCD_set_cursor(row,column);
                                        delay_ms(300);
                                        LCD_write_string(stor);

                                        //counter2++;
                                        canRead2 = 0;
                                        //}
                                }
                                if(interval >= 100) {
                                        canRead2 = 1;
                                }

                                //else
                                //printf("distance2 in mm=%d\n", g_dist2);
                        }

                }
        }
        TPM2->SC |= TPM_SC_TOF_MASK ; //clear the interrupt
}

char* my_itoa(uint16_t numb) {
        char str[3]; //create an empty string to store number
        sprintf(str, "%d", numb); //make the number into string using sprintf function
        printf("str: %s\n",str);
        return str;
}

void counting(){
        delay_ms(1000);
        char stprt[] = "COUNTING...";
        Screen_display(stprt);
        LCD_write_nextln();
        //LCD_write_string("");
}

void numReached(){
        delay_ms(600);
        char endprt[] ="GUESTS ARRIVED!";
        Screen_display(endprt);
        keypad();
}
```

Keypad.c

```
#include "MKL25Z4.h"
#include "keypad.h"

void activate_keypad(){

	SIM->SCGC5|= MASK(12);


	//Set columns to high
	PORTD->PCR[C1] &= ~0x700;
	PORTD->PCR[C1] |= MASK(8);
	PORTD->PCR[C1] |=  0x003;

	PORTD->PCR[C2] &= ~0x700;
	PORTD->PCR[C2] |= MASK(8);

	PORTD->PCR[C3] &= ~0x700;
	PORTD->PCR[C3] |= MASK(8);
	PORTD->PCR[C3] |=  0x003;

	PORTD->PCR[C4] &= ~0x700;
	PORTD->PCR[C4] |= MASK(8);
	PORTD->PCR[C4] |=  0x003;

	//set up rows as GPIO
	PORTD->PCR[R1] &= ~0x700;
	PORTD->PCR[R1] |= MASK(8);

	PORTD->PCR[R2] &= ~0x700;
	PORTD->PCR[R2] |= MASK(8);

	PORTD->PCR[R3] &= ~0x700;
	PORTD->PCR[R3] |= MASK(8);

	PORTD->PCR[R4] &= ~0x700;
	PORTD->PCR[R4] |= MASK(8);
}
```

keypad.h

```
#include "MKL25Z4.h"

#ifndef KEYPAD_H_
#define KEYPAD_H_

#define R1 (7)
#define R2 (6)
#define R3 (5)
#define R4 (4)

#define C1 (3)
#define C2 (2)
#define C3 (1)
#define C4 (0)
```

```
#define MASK(x) (1UL << x)

//void intro();
//void enterMAX();
void activate_keypad();
//void keypad();
//int countnum();

#endif
```

Lcd.c

```
#include "MKL25Z4.h"
#include <stdint.h>
#include <math.h>
#include <string.h>
#include "keypad.h"
#include "lcd.h"

#define RS (0)
#define RW (1)
#define E  (2)
#define ROWS 2
#define COLUMNS 16
#define ROW_1 0x80
#define ROW_2 0xC0

void delay_us(uint16_t delay){
	for( int i = 0; i<delay;i++ ){
	__asm volatile ("nop");
	__asm volatile ("nop");
	__asm volatile ("nop");
	__asm volatile ("nop");
	__asm volatile ("nop");
	__asm volatile ("nop");
	__asm volatile ("nop");
	__asm volatile ("nop");
	}
}


void delay_ms(uint16_t Delay){
	for( int j = 0; j< 984*Delay; j++ ){
		__asm volatile ("nop");
		__asm volatile ("nop");
	}
}


void LCD_activate_pins(){
	SIM->SCGC5|= MASK(10);
	SIM->SCGC5|= MASK(11);
```

```
	PORTC->PCR[0] &= ~0x700;
	PORTC->PCR[0] |= MASK(8); //D0

	PORTC->PCR[1] &= ~0x700;
	PORTC->PCR[1] |= MASK(8); //D1

	PORTC->PCR[2] &= ~0x700;
	PORTC->PCR[2] |= MASK(8); //D2

	PORTC->PCR[3] &= ~0x700;
	PORTC->PCR[3] |= MASK(8); //D3

	PORTC->PCR[4] &= ~0x700;
	PORTC->PCR[4] |= MASK(8); //D4

	PORTC->PCR[5] &= ~0x700;
	PORTC->PCR[5] |= MASK(8); //D5

	PORTC->PCR[6] &= ~0x700;
	PORTC->PCR[6] |= MASK(8); //D6

	PORTC->PCR[7] &= ~0x700;
	PORTC->PCR[7] |= MASK(8); //D7

	PORTB->PCR[RS] &= ~0x700;
	PORTB->PCR[RS] |= MASK(8); //RS

	PORTB->PCR[RW] &= ~0x700;
	PORTB->PCR[RW] |= MASK(8); //RW

	PORTB->PCR[E] &= ~0x700;
	PORTB->PCR[E] |= MASK(8); //E


	//Setting pins as output
	PTC->PDDR |=  MASK(0);
	PTC->PDDR |=  MASK(1);
	PTC->PDDR |=  MASK(2);
	PTC->PDDR |=  MASK(3);
	PTC->PDDR |=  MASK(4);
	PTC->PDDR |=  MASK(5);
	PTC->PDDR |=  MASK(6);
	PTC->PDDR |=  MASK(7);
	PTB->PDDR |=  MASK(RS);
	PTB->PDDR |=  MASK(RW);
	PTB->PDDR |=  MASK(E);
}

void LCD_command(uint8_t command){
	PTB-> PCOR = MASK(E);
	PTC-> PDOR = command;
	PTB-> PSOR = MASK(E);
	delay_ms(1);

	PTB-> PCOR = MASK(E);
```

```
	delay_ms(1);

}

void LCD_set_cursor(uint8_t row, uint8_t column){
	uint8_t command = 0x00;
	// error handling
	if(row > ROWS)
	{
		row = 2;
	}
	if(column > COLUMNS)
	{
		column = 16;
	}
	if(row == 1)
	{
		row = ROW_1;
	}
	else
	{
		row = ROW_2;
	}
	column -= 1;
	command |= (row | column);
	PTB->PDOR &= ~MASK(RS);
	PTB->PDOR &= ~MASK(RW);
	LCD_command(command);
}

void LCD_init(){
	delay_ms(30);

	PTB->PDOR &= ~MASK(RS);
	PTB->PDOR &= ~MASK(RW);

	LCD_command(0x30);

	delay_ms(10);
	LCD_command(0x30);

	delay_us(200);
	LCD_command(0x30);
	LCD_command(0x3C);


	//LCD DisplayOff;
	PTB->PDOR &= ~MASK(RS);
	PTB->PDOR &= ~MASK(RW);
	LCD_command(0X08);

	//Clear Display;
	PTB->PDOR &= ~MASK(RS);
	PTB->PDOR &= ~MASK(RW);
	LCD_command(0x01);
```

```
	//LCD DisplayOn;
	PTB->PDOR &= ~MASK(RS);
	PTB->PDOR &= ~MASK(RW);
	LCD_command(0x0F);

	LCD_command(0x06); //Entry mode
}

void Screen_display(char abc[]){
	LCD_init();

	LCD_write_string(abc);
}

void LCD_write_string(char string[]){
	int i=0;
	for(i=0; i<strlen(string); i++){
		PTB->PDOR |= MASK(RS);
		PTB->PDOR &=~MASK(RW);
		LCD_command(string[i]);
	}
}

void LCD_write_char(char letter){

 PTB->PDOR |= (1 << RS);
 PTB->PDOR &= ~(1 << RW);
 LCD_command(letter);

}

void LCD_write_nextln(){

	PTB -> PDOR &= ~MASK(RS);
	PTB -> PDOR &= ~(1 << RW);
	delay_ms(50);
	LCD_command(0xC0);
}
```

Timer.h

```
#include "timer.h"
#include "mkl25z4.h"

#define PTE20 (20) //PTB0 associated with TPM1 CH0 //PWM trigger.
#define PTE21 (21) //PTB1 associated with TPM1 CH1
#define PTE22 (22)
#define PTE23 (23)

void init_PWMpins(){
	//set up pin PTB1 for input capture
	SIM->SCGC5 |=SIM_SCGC5_PORTE_MASK;
	PORTE->PCR[PTE21] &=~PORT_PCR_MUX_MASK;
```

```
	PORTE->PCR[PTE21] |=PORT_PCR_MUX(3);
	PORTE->PCR[PTE23] &=~PORT_PCR_MUX_MASK;
	PORTE->PCR[PTE23] |=PORT_PCR_MUX(3);

	//set up pin PTB0 for PWM
	SIM->SCGC5 |=SIM_SCGC5_PORTE_MASK;
	PORTE->PCR[PTE20] &=~PORT_PCR_MUX_MASK;
	PORTE->PCR[PTE20] |=PORT_PCR_MUX(3);
	PORTE->PCR[PTE22] &=~PORT_PCR_MUX_MASK;
	PORTE->PCR[PTE22] |=PORT_PCR_MUX(3);
}
```

Led.h

```
#include "MKL25Z4.h"
#include "led.h"

void init_LED(){
	SIM->SCGC5 |=SIM_SCGC5_PORTB_MASK;
	PORTB->PCR[RED_LED] &= ~0X700;   //Clear mux
	PORTB->PCR[RED_LED] |= MASK(8);  //setup to be GPIO
	PORTB->PCR[GREEN_LED] &= ~0X700; //Clear mux
	PORTB->PCR[GREEN_LED] |= MASK(8);      //setup to be GPIO

	PTB->PDDR |= MASK(RED_LED);
	PTB->PDDR |= MASK(GREEN_LED);
}
```