PRINCIPLES OF EMBEDDED SOFTWARE – FINAL PROJECT

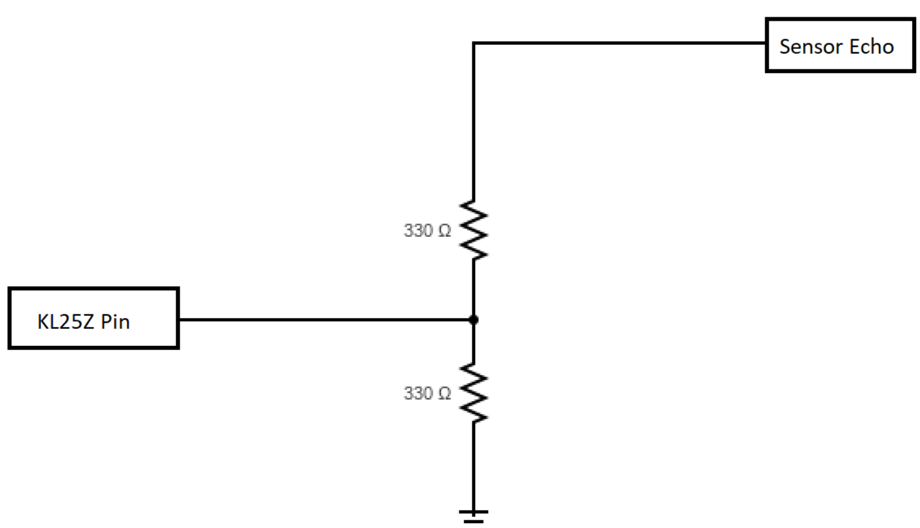
https://github.com/yogesh-1303/PES_FINAL_PROJECT.git

Introduction And Functionality:

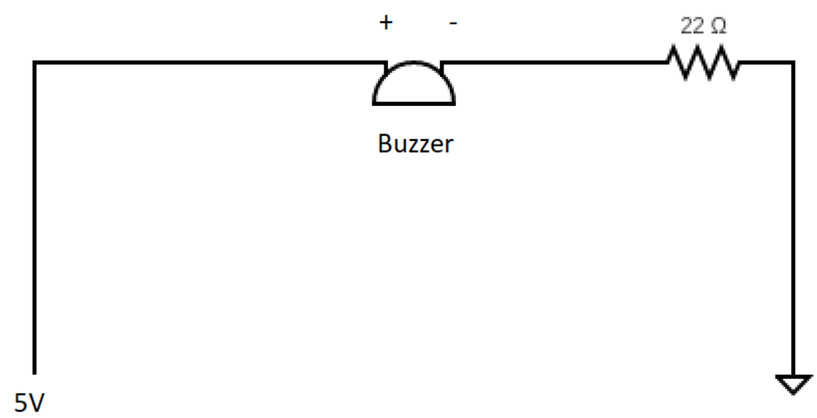
As described in the proposal, the project functions as the alarm for maintaining social distancing norms. When someone comes too close to the person (carrying sensors), the buzzer beeps. There are three ultrasonic sensors each on right, left, and center back to notify the person the maintain distancing. The intensity of the buzzer beep increases with person coming closer and decreases with a person going far.

Hardware Used:

- Three ultrasonic sensors are used for the detection of distance. These sensors work on 5V logic and hence a voltage divider circuit was used to bring down the voltage level coming to the KL25Z to 3.3V (as 5V might have damaged the GPIO pins). Following circuit was used to connect the sensors.



- Next hardware was the buzzer. The buzzer was used with a 22 ohm resistor to keep it from damaging. The buzzer's intensity increases and decreases as per the requirement. Following circuit was used for the buzze



- And finally, the whole setup was based on the KL25Z microcontroller.

Technologies Used:

Following Technologies have been used:

- TPM Input capture and detailed configuration:
  All the three timers TPM0,1 and 2 have ben used in the project. TPM1 is used for the input capture of sensor 1, TPM2, for the input capture of sensors 2 and 3, and TPM0 for PWM for buzzer operation. TPM1 and 2 are made to work on interrupts.
- PWM:
  PWM is used for the variation in intensity of the buzzer with distance. Timer 0's channel 4 is used for the purpose.

- Systick:
  Sysitck timer is used to keep track of the time and produce delays. It operates with the resolution of 1us.
- Interrupts:
  TPM1,2 and Systick, all three work on interrupts. In this project, this was one of the major challenge to synchronize the race conditions between the interrupts.
- GPIOs:
  GPIOs are used for testing and all other connections. The output to sensors is given through GPIO pins. The buzzer and LED tests are performed on GPIOs.

Testing:

The testing is mainly performed manually using components like LED. A file testing.c contains all the functions used for testing while making the code (most of which are hardware related).

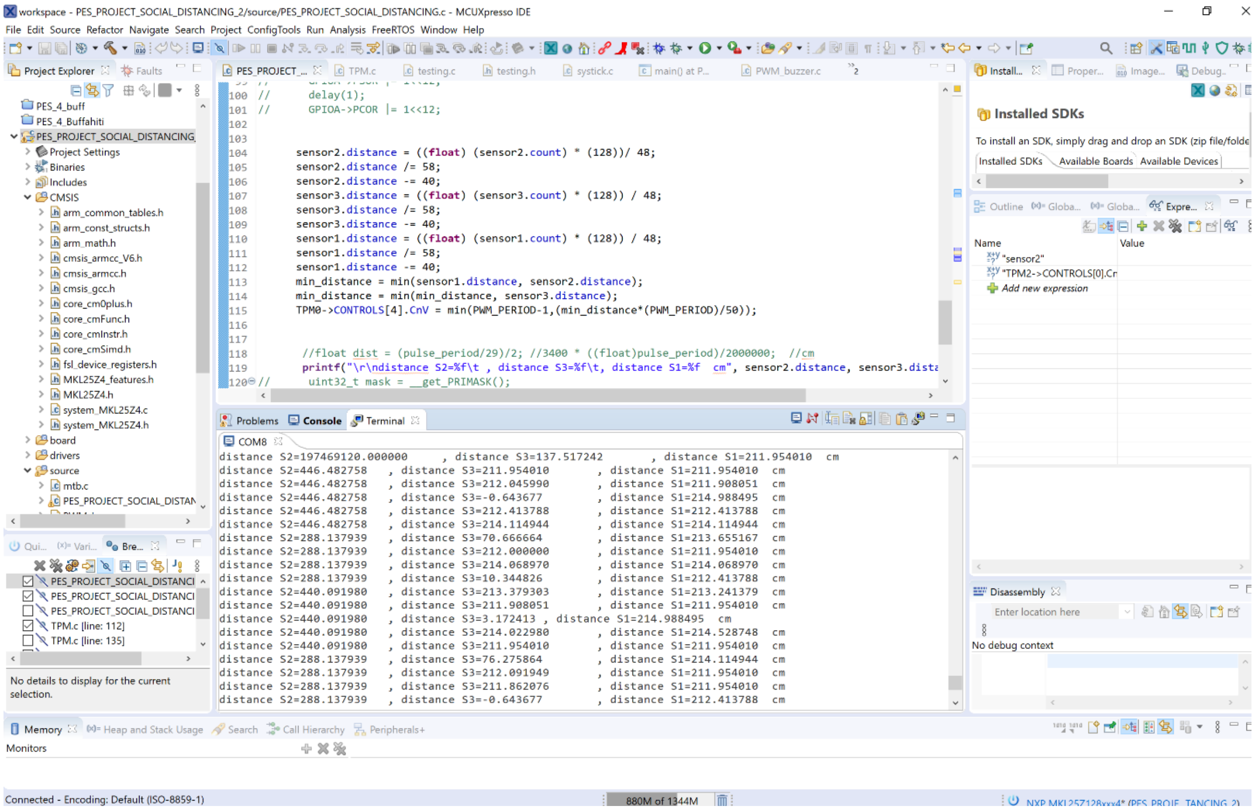
For the testing of sensors using TPM_init() functions, in TPM2, 2 variables are provided which configure for the individual testing of the sensors. All one has to do is set the variable values correctly and run the init function. If one wants to test all the three sensors, then that can also be done by testing both, sensor 2 and 3, simultaneously along with sensor one (which is connected to TPM1).

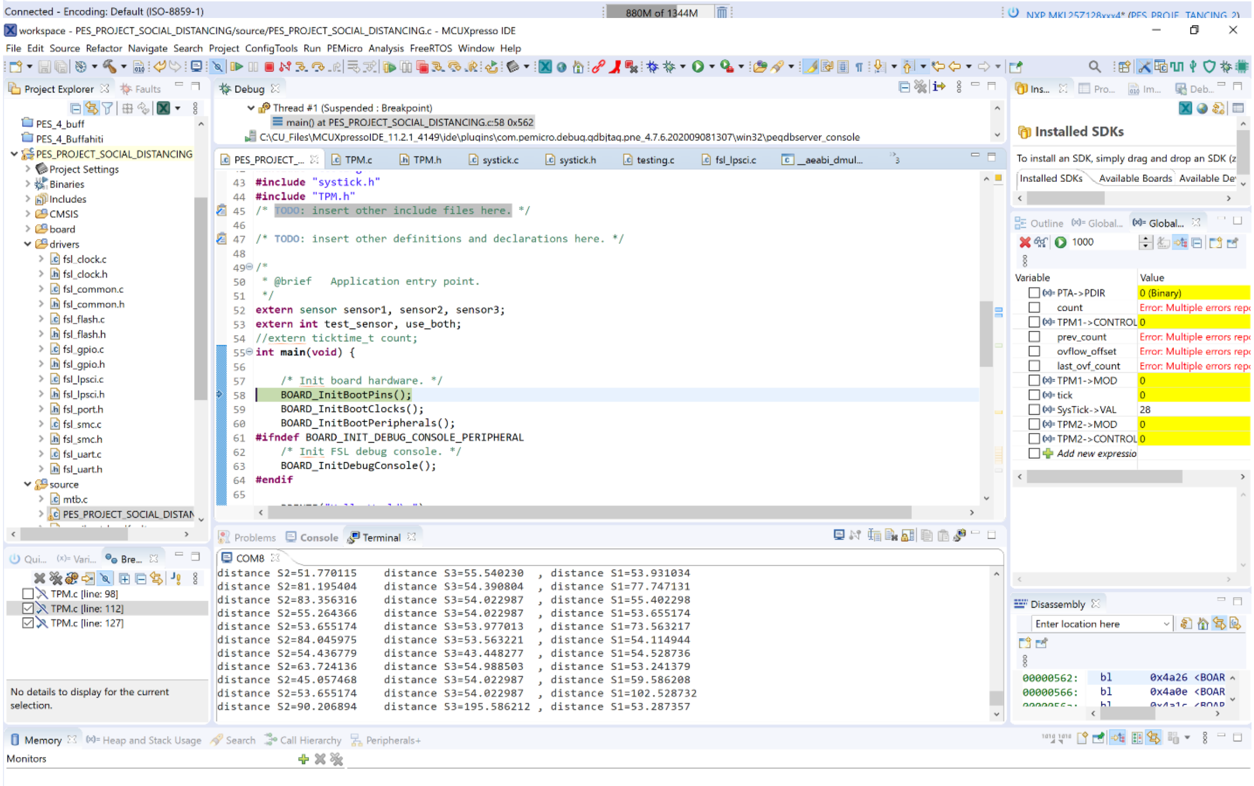
The strategy for testing the hardware outputs and input was to define a pin on top for a LED and change it accordingly for testing of output or input data. For output data, LED can directly test the working of the pin, and for input data, LED logic can be changed on detection of the input. For interrupts too, I used LEDs to make sure that the program is entering there.

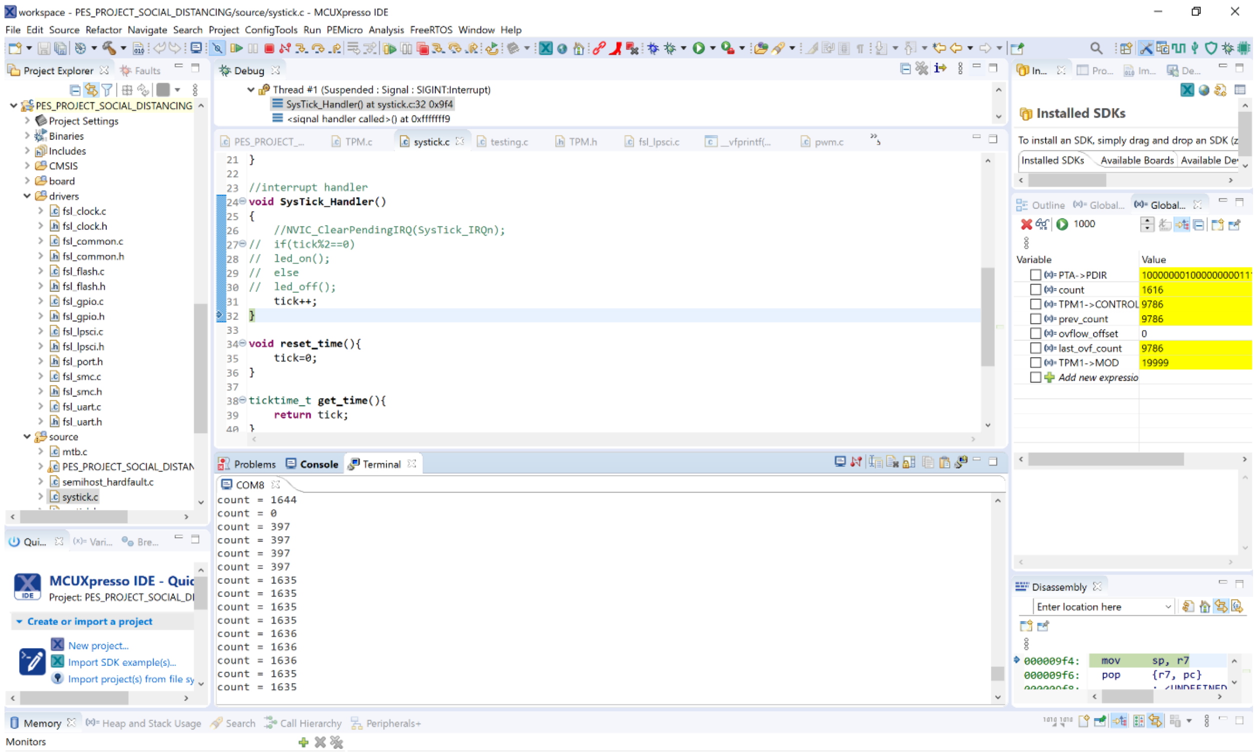
However, the major testing was done manually on the basis of distance calculated from the pulses coming back to the sensor. Once the sensor is detected, it is triggered using GPIOs which leads to input capturing. Calculating and printing the Distance often proved whether the sensor is working fine or not (as the formula for calculation of the distanxe is **simple.)**

Distance = (Speed of Sound * Time for pulse to come back) / 2

Following are the screenshots of some of the measurements.


workspace - PES_PROJECT_SOCIAL_DISTANCING_2/source/PES_PROJECT_SOCIAL_DISTANCING.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run Analysis FreeRTOS Window Help
Project Explorer
PES_4_buff
PES_4_Buffahiti
PES_PROJECT_SOCIAL_DISTANCING
Project Settings
Binaries
Includes
CMSIS
arm_common_tables.h
arm_const_structs.h
arm_math.h
cmsis_armcc_V6.h
cmsis_armcc.h
cmsis_gcc.h
core_cm0plus.h
core_cmFunc.h
core_cmInstr.h
core_cmSimd.h
fsl_device_registers.h
MKL25Z4_features.h
MKL25Z4.h
system_MKL25Z4.c
system_MKL25Z4.h
board
drivers
source
mtb.c
PES_PROJECT_SOCIAL_DISTAN
//   delay(1);
//   GPIOA->PCOR |= 1<<12;
sensor2.distance = ((float) (sensor2.count) * (128))/ 48;
sensor2.distance /= 58;
sensor2.distance -= 40;
sensor3.distance = ((float) (sensor3.count) * (128)) / 48;
sensor3.distance /= 58;
sensor3.distance -= 40;
sensor1.distance = ((float) (sensor1.count) * (128)) / 48;
sensor1.distance /= 58;
sensor1.distance -= 40;
min_distance = min(sensor1.distance, sensor2.distance);
min_distance = min(min_distance, sensor3.distance);
TPM0->CONTROLS[4].CnV = min(PWM_PERIOD-1,(min_distance*(PWM_PERIOD)/50));
//float dist = (pulse_period/29)/2; //3400 * ((float)pulse_period)/2000000; //cm
printf("\r\ndistance S2=%f\t , distance S3=%f\t, distance S1=%f cm", sensor2.distance, sensor3.dista
// uint32_t mask = __get_PRIMASK();
Installed SDKs
To install an SDK, simply drag and drop an SDK (zip file/folde
Installed SDKs Available Boards Available Devices
Name Value
"sensor2"
"TPM2->CONTROLS[0].Cr
Add new expression
Problems Console Terminal
COM8
distance S2=197469120.000000   , distance S3=137.517242   , distance S1=211.954010 cm
distance S2=446.482758 , distance S3=211.954010 , distance S1=211.954010 cm
distance S2=446.482758 , distance S3=212.045990 , distance S1=211.908051 cm
distance S2=446.482758 , distance S3=-0.643677 , distance S1=214.988495 cm
distance S2=446.482758 , distance S3=212.413788 , distance S1=212.413788 cm
distance S2=446.482758 , distance S3=214.114944 , distance S1=214.114944 cm
distance S2=288.137939 , distance S3=70.666664 , distance S1=213.655167 cm
distance S2=288.137939 , distance S3=212.000000 , distance S1=211.954010 cm
distance S2=288.137939 , distance S3=214.068970 , distance S1=214.068970 cm
distance S2=288.137939 , distance S3=10.344826 , distance S1=212.413788 cm
distance S2=440.091980 , distance S3=213.379303 , distance S1=213.241379 cm
distance S2=440.091980 , distance S3=211.908051 , distance S1=211.954010 cm
distance S2=440.091980 , distance S3=3.172413 , distance S1=214.988495 cm
distance S2=440.091980 , distance S3=214.022980 , distance S1=214.528748 cm
distance S2=440.091980 , distance S3=211.954010 , distance S1=211.954010 cm
distance S2=288.137939 , distance S3=76.275864 , distance S1=214.114944 cm
distance S2=288.137939 , distance S3=212.091949 , distance S1=211.954010 cm
distance S2=288.137939 , distance S3=211.862076 , distance S1=211.954010 cm
distance S2=288.137939 , distance S3=-0.643677 , distance S1=212.413788 cm
Disassembly
Enter location here
No debug context
No details to display for the current selection.
Memory Heap and Stack Usage Search Call Hierarchy Peripherals+
Monitors
Connected - Encoding: Default (ISO-8859-1)
880M of 1344M


workspace - PES_PROJECT_SOCIAL_DISTANCING/source/PES_PROJECT_SOCIAL_DISTANCING.c - MCUXpresso IDE
File Edit Source Refactor Navigate Search Project ConfigTools Run PEMicro Analysis FreeRTOS Window Help
Debug
Thread #1 (Suspended : Breakpoint)
main() at PES_PROJECT_SOCIAL_DISTANCING.c:58 0x562
#include "systick.h"
#include "TPM.h"
/* TODO: insert other include files here. */
/* TODO: insert other definitions and declarations here. */
/*
 * @brief   Application entry point.
 */
extern sensor sensor1, sensor2, sensor3;
extern int test_sensor, use_both;
//extern ticktime_t count;
int main(void) {
/* Init board hardware. */
BOARD_InitBootPins();
BOARD_InitBootClocks();
BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
/* Init FSL debug console. */
BOARD_InitDebugConsole();
#endif
Variable Value
PTA->PDIR 0 (Binary)
count Error: Multiple errors repo
TPM1->CONTROL 0
prev_count Error: Multiple errors repo
ovflow_offset Error: Multiple errors repo
last_ovf_count Error: Multiple errors repo
TPM1->MOD 0
tick 0
SysTick->VAL 28
TPM2->MOD 0
TPM2->CONTROL 0
Add new expressio
COM8
distance S2=51.770115   distance S3=55.540230 , distance S1=53.931034
distance S2=81.195404   distance S3=54.390804 , distance S1=77.747131
distance S2=83.356316   distance S3=54.022987 , distance S1=55.402298
distance S2=55.264366   distance S3=54.022987 , distance S1=53.655174
distance S2=53.655174   distance S3=53.977013 , distance S1=73.563217
distance S2=84.045975   distance S3=53.563221 , distance S1=54.114944
distance S2=54.436779   distance S3=43.448277 , distance S1=54.528736
distance S2=63.724136   distance S3=54.988503 , distance S1=53.241379
distance S2=45.057468   distance S3=54.022987 , distance S1=59.586208
distance S2=53.655174   distance S3=54.022987 , distance S1=102.528732
distance S2=90.206894   distance S3=195.586212 , distance S1=53.287357
Disassembly
Enter location here
No details to display for the current selection.
Memory Heap and Stack Usage Search Call Hierarchy Peripherals+
Monitors



Challenges Faced:

- The primary challenge was to synchronize the interrupts. Since there were 3 interrupts working over the same span of time, it was really difficult to debug and correct. However, the learning was great as well.
- One more difficult part was to connect and configure the sensors correctly for working. For this, it was required to read datasheet and working principle of the sensor.
- To find out the pin configuration was also one of the hurdles though not as hard as the ones above.

The Code:

The code consists of five .c and their corresponding .h files as follows:

PES_PROJECT_SOCIAL_DISTANCING.c: Contains the main function which calls the init functions.

PWM_buzzer.c: Contains PWM_init() function for the initialization of PWM with TPM0 on channel 4. This file also contains the buzzer_n_distance() function which calculates the distance of the obstacle from the sensor and also sets the buzzer intensity accordingly.

Systick.c: This file contains the initialization of systick timer at 1us resolution. The timer is used for all the timings in the project and producing delays. The API also has functions for retrieving current time, resetting the time, and delaying.

TPM_sensors: this file is the crux of the project where main thing lies. The project contains initialization of the TPM1 and 2 timers for the input capture mode used for capture the input by sensors through interrupts. This file contains the interrupt handlers of the two timers. One timer is used for the input capture of sensor1 and the second is used for the input capture of sensors 2 and 3 on different channels.

Testing.c: This file contains the functions used during the progression of the code for testing purposes. Functions that help test the input and output through a pin. A GPIO init function is also present.

Apart from this, the TPM_sensors.h file contains the structure which where the variables used to manage the sensors are present.

CODE:

```
/*
 * Copyright 2016-2021 NXP
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * @file    PES_PROJECT_SOCIAL_DISTANCING.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include <TPM_sensors.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
```

```
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "testing.h"
#include "systick.h"
#include "PWM_buzzer.h"
/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

/*
 * @brief   Application entry point.
 */


extern int test_sensor, use_both;
extern sensor sensor1, sensor2, sensor3;
int main(void) {


       /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();
#endif
    //initilialization of GPIOs used (including sensors),
    //systick and PWM
    GPIO_init();
    systick_init();
    PWM_init();

    //tests the buzzer by playint it once from high to low sound and vice versa
    test_buzzer();

    //thses two variables are used to test the working of sensors throughout the
coding
    //when test_sensor is 2, SENSOR 2 is tested (initialized and then IRQ handling)
    //when test sensor is 3,  SENSOR 2 is tested (initialized and then IRQ handling)
    //when use_both is 1, both are initilized irrespective of the value of
test_sensor
    test_sensor=2;
    use_both=1;

    //trigger the sensors by giving a 10 us pulse (note: the pin used is active high)
    GPIOA->PSOR |= 1<<SENSOR2_TRIG | 1<<SENSOR3_TRIG | 1<<SENSOR1_TRIG;
    delay(10);
    GPIOA->PCOR |= 1<<SENSOR2_TRIG | 1<<SENSOR3_TRIG | 1<<SENSOR1_TRIG;

    //TPM1 and TPM2 initialization
    tpm2_init();
    tpm1_init();
```

```
    while(1) {
      //calculate the distance and buzzer intensity according to the distance of a
body from sensors
      buzzer_n_distance();
       printf("\r\ndistance S2=%f\t , distance S3=%f\t, distance S1=%f  cm",
sensor2.distance, sensor3.distance, sensor1.distance);
              // test_input(PIN_IN);
    }
    return 0 ;
}
```

PWM_buzzer.c:

```
#include <stdio.h>
#include "MKL25Z4.h"
#include <stdint.h>
#include <TPM_sensors.h>
#include "PWM_Buzzer.h"

#define BUZZER_PIN 8U   //buzzer is attached to PORTC pin 8

#define min(x,y) (((x)<(y))?(x):(y))
extern sensor sensor1, sensor2, sensor3;

//initializes PWM for for buzzer intensity on output compare mode of TPM0 channel 4
//takes and returns void
void PWM_init(){
      //providing clock to port c and TPM0 timer
      SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;
      SIM->SCGC6 |= SIM_SCGC6_TPM0_MASK;

      //set MUX to TPM0 channel 4
      PORTC->PCR[BUZZER_PIN] &= ~PORT_PCR_MUX_MASK;
      PORTC->PCR[BUZZER_PIN] |= PORT_PCR_MUX(3);

      //select the clock of 48MHz
      SIM->SOPT2 |= (SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK);

      //MOD of full capacity, prescalar of 8
      TPM0->MOD = PWM_PERIOD-1;
      TPM0->SC = TPM_SC_PS(3);

      //toggle on compare match on channel 4 of TPM0 and set CnV
      TPM0->CONTROLS[4].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK;
      TPM0->CONTROLS[4].CnV = 0;

      //start the timer
      TPM0->SC |= TPM_SC_CMOD(1);
}

//Calculates the distance of the obstruction
//sets buzzer intensity
```

```c
//prints minimum distance
//takes and returns void
void buzzer_n_distance(){
	uint32_t min_distance;
	//the formula for distance is (speed of sound * time taken by wave to travel 
back)/2
	//converted and solved for us time as input the below given formula gives the 
distance
	//40 is the offset value that is subtracted
    sensor2.distance = ((float) (sensor2.count) * (128))/ 48;
	sensor2.distance /= 58;
	sensor2.distance -= 40;
    sensor3.distance = ((float) (sensor3.count) * (128)) / 48;
    sensor3.distance /= 58;
    sensor3.distance -= 40;
    sensor1.distance = ((float) (sensor1.count) * (128)) / 48;
    sensor1.distance /= 58;
    sensor1.distance -= 40;
    //calculating the minimum distance of obstruction for the buzzer intensity
    min_distance = min(sensor1.distance, sensor2.distance);
    min_distance = min(min_distance, sensor3.distance);
    //setting buzzer intensity
    TPM0->CONTROLS[4].CnV = min(PWM_PERIOD-1,(min_distance*(PWM_PERIOD)/50));
    printf("\n\rminimum distance = %u", min_distance);
}
```

TPM_sensors.c

```c
#include <TPM_sensors.h>
#include "MKL25Z4.h"
#include "testing.h"
#include "systick.h"
#include "testing.h"

//#define S3

extern ticktime_t tick;

int test_sensor, use_both;
int last_sensor_flag;

sensor sensor1={0}, sensor2={0}, sensor3={0};

//tpm1 initializtion function: works on Input capture mode on channel 0
//used for capturing Input from Sensor 1
//works using channel and overflow interrupts
//takes and returns void
void tpm1_init(){

	//provide clock to TPM1 and PORTE
	SIM->SCGC6 |= SIM_SCGC6_TPM1_MASK;
	SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;
```

```
	//select the clock
	SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK;

	//select MOD value
	TPM1->MOD = 20000-1;

	//set the CnSC values for falling edge capture with interrupt
	TPM1->CONTROLS[0].CnSC = TPM_CnSC_ELSB_MASK | TPM_CnSC_ELSA(0) |
TPM_CnSC_CHIE_MASK;

	//set MUX value for input capture
	PORTE->PCR[SENSOR1_IN] &= ~PORT_PCR_MUX(7);
	PORTE->PCR[SENSOR1_IN] |= PORT_PCR_MUX(3);

	//prescalar of 128, overflow Interrupt enable, start the timer
	TPM1->SC = TPM_SC_CMOD(1) | TPM_SC_PS(7)  | TPM_SC_TOIE_MASK |
TPM_SC_CPWMS(0);

	//set priority, clear pending interrupts, and enable interrupt
	NVIC_SetPriority(TPM1_IRQn, 2);
	NVIC_ClearPendingIRQ(TPM1_IRQn);
	NVIC_EnableIRQ(TPM1_IRQn);
}

//IRQ Handler for TPM1
void TPM1_IRQHandler() {
	//out_off();
	//when overflow occurs, adjust the offset as the count will reset
	if(TPM1->STATUS & TPM_STATUS_TOF_MASK){
		sensor1.ovflow_offset = 20000 - sensor1.prev_count;
		sensor1.prev_count = 0;
	}
	//when the capture occurs, take count and store this value for next cycle
	if(TPM1->STATUS & TPM_STATUS_CH0F_MASK) {
		sensor1.last_ovf_count = TPM1->CONTROLS[0].CnV;
		sensor1.count = TPM1->CONTROLS[0].CnV - sensor1.prev_count +
sensor1.ovflow_offset;
		sensor1.prev_count = TPM1->CONTROLS[0].CnV;
		//trigger the sensor for next capture
		GPIOA->PSOR |= 1<<SENSOR1_TRIG;
		delay(10);
		GPIOA->PCOR |= 1<<SENSOR1_TRIG;
		sensor1.ovflow_offset = 0;
	}
	//Reset all the flags
	TPM1->STATUS |=  TPM_STATUS_TOF_MASK | TPM_STATUS_CH0F_MASK;
	//out_on();
}

//tpm1 initializtion function: works on Input capture mode on channels 0 and 1
//used for capturing Input from Sensors 2 and 3
//works using channel and overflow interrupts
//takes and returns void
void tpm2_init() {
```

```
	SIM->SCGC6 |= SIM_SCGC6_TPM2_MASK;
	SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

	SIM->SOPT2 |= SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK;

	TPM2->MOD = 20000-1;

	//if call is for sensor 2 only, this block initializes the sensor 2 and leads
to its IRQ
	if((test_sensor==2)||(use_both==1)) {
	TPM2->CONTROLS[0].CnSC = TPM_CnSC_ELSB_MASK | TPM_CnSC_ELSA(0) |
TPM_CnSC_CHIE_MASK;
	PORTE->PCR[SENSOR2_IN] &= ~PORT_PCR_MUX(7);
	PORTE->PCR[SENSOR2_IN] |= PORT_PCR_MUX(3);
	}

	//if call is for sensor 3 only, this block initializes the sensor 2 and leads
to its IRQ
	if((test_sensor==3)||(use_both==1)){
	TPM2->CONTROLS[1].CnSC = TPM_CnSC_ELSB_MASK | TPM_CnSC_ELSA(0) |
TPM_CnSC_CHIE_MASK;
	PORTE->PCR[SENSOR3_IN] &= ~PORT_PCR_MUX(7);
	PORTE->PCR[SENSOR3_IN] |= PORT_PCR_MUX(3);
	}
	//prescalar of 128, overflow Interrupt enable, start the timer
	TPM2->SC = TPM_SC_CMOD(1) | TPM_SC_PS(7)  | TPM_SC_TOIE_MASK |
TPM_SC_CPWMS(0);

	//set priority, clear pending interrupts, and enable interrupt
	NVIC_SetPriority(TPM2_IRQn, 2);
	NVIC_ClearPendingIRQ(TPM2_IRQn);
	NVIC_EnableIRQ(TPM2_IRQn);
}

//TPM2 Interrupt Handler
void TPM2_IRQHandler() {
	//out_on();
	//when overflow occurs, adjust the offset as the count will reset
	//to keep the track of last active sensor, a variable is used
	if(TPM2->STATUS & TPM_STATUS_TOF_MASK){
		if(last_sensor_flag==3){
		sensor3.ovflow_offset = 20000 - sensor3.prev_count;
		sensor3.prev_count = 0;
		}
		if(last_sensor_flag==2){
			sensor2.ovflow_offset = 20000 - sensor2.prev_count;
			sensor2.prev_count = 0;
		}
	}

	//if called for sensor 3 or both sensor 2 and 3
	if((test_sensor==3)||(use_both==1)){
		//when the capture occurs, take count and store this value for next
cycle
		if(TPM2->STATUS & TPM_STATUS_CH1F_MASK) {
```

```
            last_sensor_flag = 3;
            sensor3.last_ovf_count = TPM2->CONTROLS[1].CnV;
            sensor3.count = TPM2->CONTROLS[1].CnV - sensor3.prev_count +
sensor3.ovflow_offset;
            sensor3.prev_count = TPM2->CONTROLS[1].CnV;
            //trigger the sensor for next capture
            GPIOA->PSOR |= 1<<SENSOR3_TRIG;
            delay(10);
            GPIOA->PCOR |= 1<<SENSOR3_TRIG;
            sensor3.ovflow_offset = 0;
      }
      }
      //if called for sensor 2 or both 2 and 3
      if((test_sensor==2)||(use_both==1)){
            //when the capture occurs, take count and store this value for next
cycle
            if(TPM2->STATUS & TPM_STATUS_CH0F_MASK){
            last_sensor_flag = 2;
            sensor2.last_ovf_count = TPM2->CONTROLS[0].CnV;
            sensor2.count = TPM2->CONTROLS[0].CnV - sensor2.prev_count +
sensor2.ovflow_offset;
            sensor2.prev_count = TPM2->CONTROLS[0].CnV;
            //trigger the sensor for next capture
            GPIOA->PSOR |= 1<<SENSOR2_TRIG;
                  delay(10);
            GPIOA->PCOR |= 1<<SENSOR2_TRIG;
            sensor2.ovflow_offset = 0;
            }
      }
      //Reset all the flags
      TPM2->STATUS |=  TPM_STATUS_TOF_MASK | TPM_STATUS_CH0F_MASK |
TPM_STATUS_CH1F_MASK;
      //out_off;
}
```

Systick.c:

```
#include <stdio.h>
#include <MKL25Z4.h>
#include <stdint.h>
#include "core_cm0plus.h"
#include "systick.h"
#include "testing.h"

ticktime_t tick = 0;
ticktime_t temp = 0;
//extern uint16_t buffer[STEPS];
//extern uint16_t samples;

//Systick timer initialization
void systick_init()
{
```

```
	SysTick->LOAD = (48L/16);		//1 us resolution

	SysTick->VAL = 0;					//Initial value of counter
	SysTick->CTRL &= ~(SysTick_CTRL_CLKSOURCE_Msk);		//frequency = 3Mhz (ext clock)
	SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk; //enable interrupt
	NVIC_ClearPendingIRQ(SysTick_IRQn); //for 10us res
	NVIC_SetPriority (SysTick_IRQn, 1);
}

//interrupt handler
int a;
void SysTick_Handler()
{
	if(tick==6)
		a = 5;
	tick++;
}

//resets the time
void reset_time(){
	tick=0;
}

//returns the current time
ticktime_t get_time(){
	return tick;
}

//produces the delay in microseconds
//takes number of microseconds to be delayed as argument
//returns void
void delay(ticktime_t us){
	reset_time();
	while(tick!=us){
		__asm volatile ("nop");
	}

}
```

Testing.c

```
#include <stdio.h>
#include "MKL25Z4.h"
#include "testing.h"
#include "systick.h"
#include "PWM_buzzer.h"

//initializes all the GPIOs for testing and working
```

```
//takes and returns void
void GPIO_init(){
	//provide clock to portA
	SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

	//used for testing with LED
	PORTA->PCR[LED_PIN_OUT] &= ~PORT_PCR_MUX_MASK;
	PORTA->PCR[LED_PIN_OUT] |= PORT_PCR_MUX(1);

	//set MUX value to 1 for GPIO
	PORTA->PCR[SENSOR1_TRIG] &= ~PORT_PCR_MUX_MASK;
	PORTA->PCR[SENSOR1_TRIG] |= PORT_PCR_MUX(1);

	PORTA->PCR[PIN_IN] &= ~PORT_PCR_MUX_MASK;
	PORTA->PCR[PIN_IN] |= PORT_PCR_MUX(1);
	PORTA->PCR[PIN_IN] |= PORT_PCR_PE_MASK | PORT_PCR_PS(0);


	//set direction
	PTA->PDDR &= ~PIN_IN_MASK;
	PTA->PDDR |= LED_PIN_OUT_MASK;
	PTA->PDDR |= 1<<SENSOR1_TRIG;
}

//turns the signal(led) off (used in active high)
void out_off(){
	GPIOA->PCOR |= LED_PIN_OUT_MASK;
}

//turns the signal(led) on (used in active high)
void out_on(){
	GPIOA->PSOR |= LED_PIN_OUT_MASK;
}

//A way of testing the input detection using LED, for a continuous pulse
void test_input(int pin){
	ticktime_t pulse_period=0;
	reset_time();
	while(PTA->PDIR & PIN_IN_MASK){
		out_off();
	}
	pulse_period = get_time();
	//else
	ticktime_t dist = 340 * pulse_period/2;
	printf("\r\ndistance = %u", dist);
		out_on();
}

//testing buzzer by playing it from low to high and then high to low intensity
void test_buzzer(){
	uint32_t i;
	for(uint32_t i=0; i<PWM_PERIOD; i++){
	TPM0->CONTROLS[4].CnV =  i;
	delay(10);
	}
```

```
        for(i=PWM_PERIOD-1; i>0; i--){
        TPM0->CONTROLS[4].CnV = i;
        delay(10);
        }
}
```

Testing.h

```
#define PIN_IN  12U
#define SENSOR1_TRIG 4U
#define LED_PIN_OUT 5U
#define LED_PIN_OUT_MASK  (1<<LED_PIN_OUT)
#define PIN_IN_MASK   (1<<PIN_IN)

//initializes all the GPIOs for testing and working
//takes and returns void
void GPIO_init();

//turns the signal(led) off (used in active high)
void out_on();

//turns the signal(led) on (used in active high)
void out_off();

//A way of testing the input detection using LED, for a continuous pulse
//the function tells whether the input has been received or not by switching LED
logic
void test_input(int);

//testing buzzer by playing it from low to high and then high to low intensity
void test_buzzer();
```

---

systick.h:

```
#include "stdint.h"

typedef uint32_t ticktime_t;

//Systick timer initialization for 1 us resolution at 3Mhz external frequency
void systick_init();

//interrupt handler
void SysTick_Handler();

//resets the time to 0
void reset_time();

//return current time
ticktime_t get_time();

//produces the delay in microseconds
```

```
//takes number of microseconds to be delayed as argument
//returns void
void delay(ticktime_t);
```

TPM_sensors.h:

```
#include "systick.h"

#define SENSOR1_IN 20
#define SENSOR2_IN 22
#define SENSOR3_IN 23
#define SENSOR2_TRIG 2
#define SENSOR3_TRIG 5

//structure conatining all the needed variables for the sensor management and working
typedef struct SENSOR_VARS{
	ticktime_t count, prev_count;
	ticktime_t ovflow_offset, last_ovf_count;
	float distance;
}sensor;

//tpm1 initializtion function: works on Input capture mode on channel 0
//used for capturing Input from Sensor 1
//works using channel and overflow interrupts
//takes and returns void
void tpm1_init();

//tpm1 initializtion function: works on Input capture mode on channels 0 and 1
//used for capturing Input from Sensors 2 and 3
//works using channel and overflow interrupts
//takes and returns void
void tpm2_init();
```

PWM_buzzer.h:

```
#define PWM_PERIOD 0xffff

//initializes PWM for for buzzer intensity on output compare mode of TPM0 channel 4
//takes and returns void
void PWM_init();

//Calculates the distance of the obstruction
//sets buzzer intensity
//prints minimum distance
//takes and returns void
void buzzer_n_distance();
```

CREDITS:

- Howdy Pierce for providing vital support
- Book: Embedded Systems Fundamentals by Alexander Dean
- KL25Z reference manual
- HR SC04 datasheet