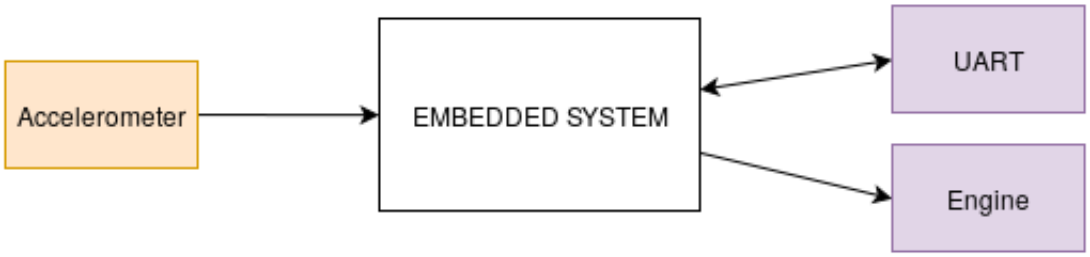
443 Final Project Report

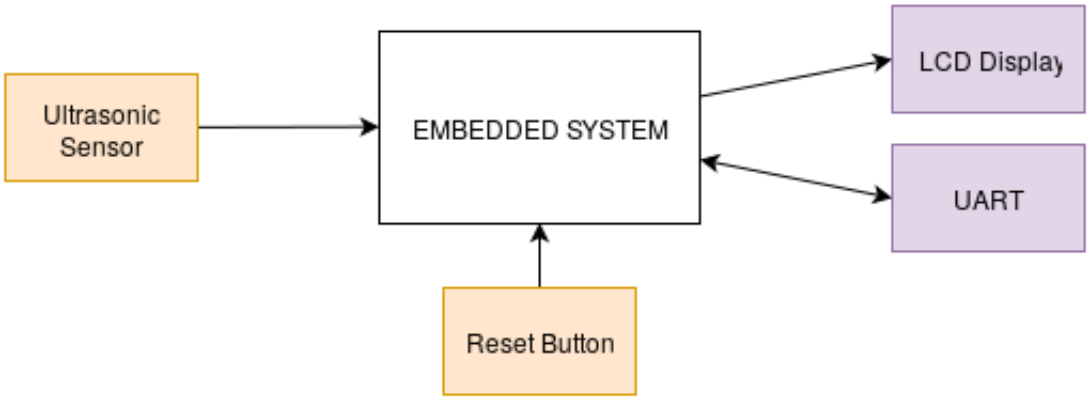
ADAMLAR

1-System Level Structural Diagram

Module: Lap Counter



Module: Getting Speed Data from Input Device

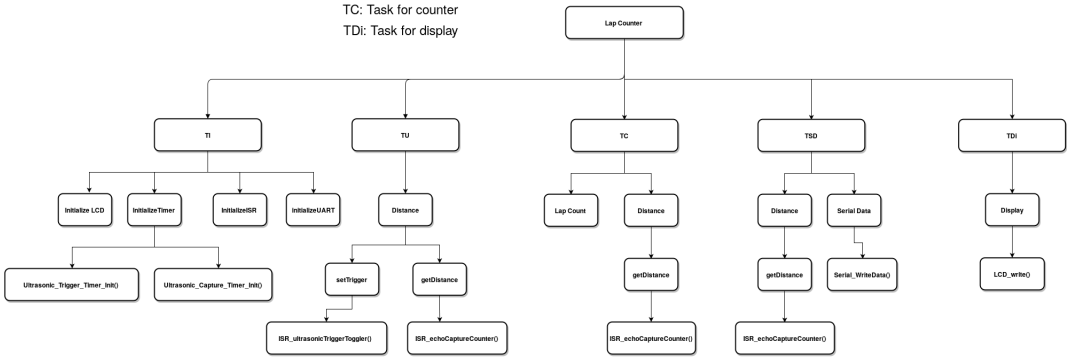


2-

Task Decomposition Graph

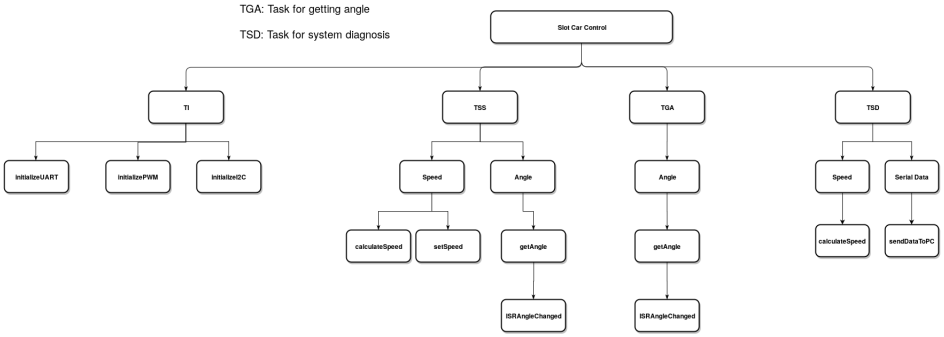
Module: Lap Counter

TU: Task for ultrasonic
TI: Task for initialization
TSD: Task for system diagnosis
TC: Task for counter
TDi: Task for display



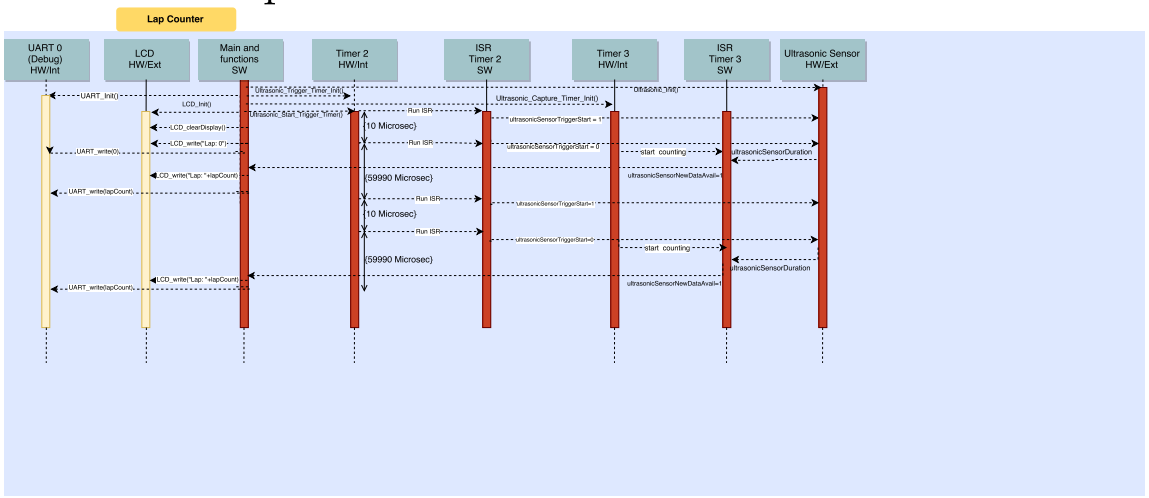
Module: Getting Speed Data from Input Device

TI: Task for initialisation
TSS: Task for setting speed
TGA: Task for getting angle
TSD: Task for system diagnosis

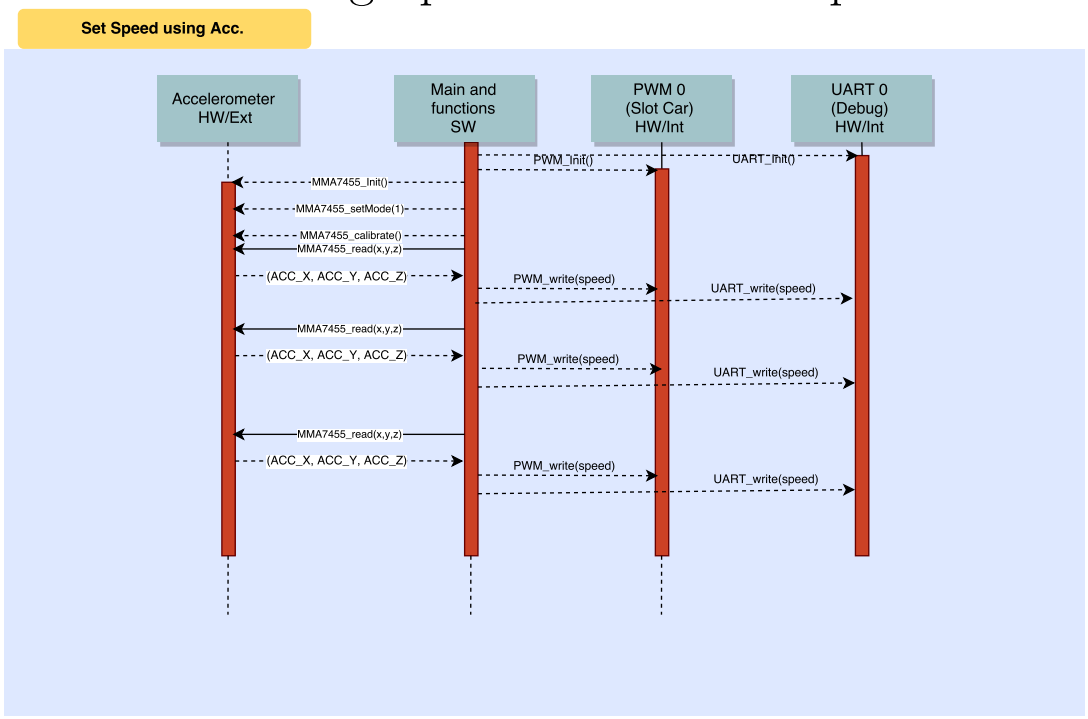


3-Sequence Diagram

Module: Lap Counter



Module: Getting Speed Data from Input Device



4-Coding

Module: Lap Counter

Function Name	**Function Definition**	**Objective**	**WCET** (simulation)
Ultrasonic_Init()	Initialization of the Ultrasonic sensor. Functions of the trigger and echo pins connected to the LPC board are defined in the IOCON registers	There are 2 data pins to communicate with the ultrasonic sensor. Enabling the corresponding pins in the main board, their directions' (input or output) is set for echo and trigger pins. So that, the board is able to trigger a calculation of the distance and get the incoming signals with echo pin.	18 usec
Ultra-sonic_Trigger_Timer_Init()	Trigger timer initializer. The output for the trigger pin is initialized in this function	Timer 2 is used to synchronize the value(either HIGH or LOW) to the Ultrasonic sensor. To initialized it, this function is used. First the power is enabled to this section with PCONP Register , then counters (Timer counters and Prescale counters) and match values are set appropriately. Last, the function on a match("toggle" in this case) is set.	31 usec
LCD_init()	Initialization of the LCD and its pins.	LCD is initialized by setting its pins' functionalities on the board, and sending correct values to initialize the external driver, (like 0x03, 0x03,0x03,0x02)	5.18 sec

Ultra-sonic_Capture_Timer_Init()	Echo timer initializer for the echo signal of the ultrasonic sensor.	First the using PCONP register, the timer is powered on. Then its match registers and timer and prescale timer registers are given the appropriate functionalities. This timer is used to count the elapsed time from the transmission of the signals to the echo of the sent signals. So that, the distance of the object could be detected.	21.3 usec
Serial_init()	Initialization of the UART 0	Using UART(Serial communication), the debug information of distance of the detected object and the lap count will be sent to the computer.	57.8 usec
TU()	Task for Ultrasonic, calculates the distance	The distance is calculated detected by ultrasonic sensor. This distance is used to detect whether the car is in the range or not.	30.6 microsec
TC()	Task for the lap counter	After getting the distance detected, this function checks whether the distance is smaller than the threshold, if it is, then the lap count is incremented by one.	23.0 microsec
TSD()	Task for system diagnosis, includes sending distance detected and the lap count data from UART to PC	In order to debug the code, we are using this function which will be printing the value of distance measured and the lap counted.	192 millisec
TDi()	This function sets the cursor to the appropriate position, then clears the display starting from the cursor, then writes the lap count into the LCD screen.	To meet the requirements specified, this function is used. It basically, writes the lap count into the LCD screen.	12.2 microsec

Module: Getting Speed Data from Input Device

Function Name	**Function Definition**	**Objective**	**WCET** (simulation)
TI()	Function initializes all the devices/modules used by the operating code.Basically consists of initializing accelerometer, PWM pin,	In order to use the devices in the board, we have to initialize them before using. Powering on the devices(PCONP) and giving the appropriate pin functionalities(IOCON) are done in this step.	5.86 secs
TSS()	Task for setting the speed of the car. To set a speed, first we are reading the angle of the board, from the accelerometer. Using the angle value of it, the pulse width of the output signal is modulated from 0 to 999 where 500 means that 50% duty cycle.	To meet the requirements specified, we are using the accelerometer data as input to set the speed of the slot car. After getting the angle value, the speed of the car is set by using PWM.	44.6 microsec
TGA()	Task for getting the angle of the board using the accelerometer device on it. This function reads the acceleration data from the device on the LPC board and finds the angle in the X dimension of the device.	To make the speed of the slot car controllable by an input device, accelerometer device is used and according to its angle in the X direction, the speed of the car is set. This function realizes the getting the angle data.	79.7 microsec
TSD()	Task for system diagnosis. This function is used to send the speed information to the PC using UART.	To debug the code that we wrote in this section, the data produced is sent to the PC to oversee the system working.	522 millisec

5-Scheduling

Module: Lap Counter

In this part of the project, we are using Polling with interrupts. When a new data is received using the ISR_echoCaptureCounter, the flag ultrasonicSensorNewDataAvailable is set to 1, the pseudo code is as follows:
ISR_echoCaptureCounter(){
if new data then
ultrasonicSensorDuration=duration; ultrasonicSensorNewDataAvailable=1;
}
This ISR is initiated after each 60 milliseconds and called when a signal is echoed back to the sensor. This process of repeated initialization and waiting functions are done via the usage of Timer 2 and Timer 3 together.
While the ISR is not running, the main function polls the flag ultrasonicSensorNewDataAvailable. The execution of the main loop is as follows:
while(true){
TU();//Poll the flag, if there is new data available, then calculate the distance
TC();//If the new distance is smaller then the threshold, increment the lap Counter
TDI();//Display the lap count value in the LCD
TSD();// System diagnosis, send relevant information using UART to debug
}

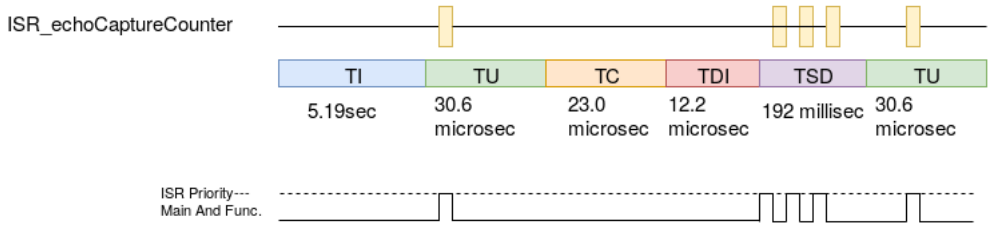
Module: Getting Speed Data from Input Device

In this section, there is no interrupt used, the code executes in a cycle and reads data from the accelerometer. The communication with the accelerometer is done via I2C. An interrupt algorithm could have been "Set a timer to repeat infinitely with a frequency, check if the value in accelerometer changed and set a flag if it is changed.". However, this has no benefits over polling the data itself with a frequency, so we decided that it is more convenient if it is not used. The main schedule of this module program is that:
while(true){
TGA();//Poll the value in accelerometer and calculate angle
TSS();//Set the speed of the car using the angle value from acc.
TSD();//Send the debug information to the PC
}

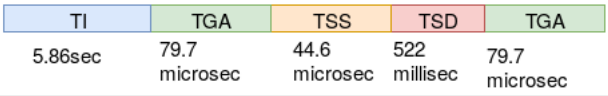
Shared Variable	Name of the function	Name of the ISR
ultrasonicNewDataAvailable	TU	ISR_echoCaptureCounter
ultrasonicSensorDuration	TU	ISR_echoCaptureCounter
ultrasonicSensorTriggerStart	TI	ISR_ultrasonicTriggerToggler

6-Timing Diagram

Module: Lap Counter



Module: Getting Speed Data From Input



Name of the ISR	Shared Variable	Priority	WCET	ACET
ISR_ultrasonicTriggerToggler	ultrasonicSensorTriggerStart	5	6.9 us	6.32 us
ISR_echoCaptureCounter	ultrasonicSensorDuration, ultrasonicSensorNewDataAvailable	1	8.5 us	6.1 us

7-Hardware Block Diagram

Component	Type ID	Quickstart Board	Base-board	Off-board
Ultrasonic	HC-SR04			x
LCD	LCM-S01601DSR			x
Potentiometre	B10K			x
Accelerometer	MMA7455		X	
LED				X
1K-Resistor				X

Expenses	Cost
120pcs 10cm male to male + male to female and female to female jumper wire	$2.5

8-Board Pin Table

LCD PINS	LPC4088 PINS
1	GND
2	VU
3	to 2. pin of potentiometer
4 RS	P0.8 (P12)
5 RW	P0.6 (P14)
6 EN	P0.7 (P13)
11 DATA0	P0.24 (P16)
12 DATA1	P0.25 (P17)
13 DATA2	P0.26 (P18)
14 DATA3	P1.30 (P19)

Potentiometer	LPC4088 PINS
left	Vin
right	GND

Ultrasonic	LPC4088 PINS
Vcc	Vin
GND	GND
Trig Trigger	P0.9(P11)
Echo Echo	P0.23(P15)

Slot Car	LPC4088 PINS
Vcc	P1.5(P28)
GND	GND

9-Appendix

Module: Lap Counter

main.c

```
#include "LPC407x_8x_177x_8x.h"

#include "Library/Ultrasonic.h"
#include "Library/SystemStructures.h"
#include "Library/Timer.h"
#include "Library/LCD.h"
#include "Library/Serial.h"
#include <stdio.h>

int distance = 0 ;
char l = 0, refresh = 0;
int flag = 0,lapCounter = 0, carDistanceLimit = 10;
char stringValue [30];

// Task Initialization
void TI()
{
        Ultrasonic_Init();
        Ultrasonic_Trigger_Timer_Init();
        Ultrasonic_Capture_Timer_Init();
        Ultrasonic_Start_Trigger_Timer();
        LCD_Init();
        LCD_clearDisplay();
        LCD_write("LAP: ");
        LCD_data('0');
        Serial_Init();
        sprintf(stringValue ,"SYSTEM DIAGNOSIS STARTED\r\n");
        serialTransmitData = stringValue;
        Serial_WriteData(*serialTransmitData++);
}

// Task Display
void TDI()
{
        if(refresh == 0) return;
        LCD_setCursorPositionFirstLine(5);
        if(lapCounter > 9)
                LCD_data('0' + (lapCounter/10));
        LCD_data('0' + (lapCounter%10));
        refresh = 0;
}
```

```
// Task Ultrasonic
void TU()
{
        if(ultrasonicSensorNewDataAvailable == 1)
        {
                distance = ultrasonicSensorDuration / 58;
                ultrasonicSensorNewDataAvailable = 0;
                return;
        }
        distance = -1;
}

// Task Lap Counter
void TC()
{
                if(flag == 0 && distance < carDistanceLimit)
                {
                        flag = 1;
                        lapCounter++;
                        refresh = 1;
                        if(lapCounter == 100) lapCounter = 0;
                }
                if(flag == 1 && distance >= carDistanceLimit) flag = 0;
}

// Task system diagnosis
void TSD()
{
        sprintf(stringValue ,"LAP:%d \t%d\r\n" , lapCounter, distance);
        serialTransmitData = stringValue;
        Serial_WriteData(*serialTransmitData++);
        while(!serialTransmitCompleted);
}

void update() {
        TU();
        if(distance == -1) return;
        TC();
        TDI();
        TSD();
}

int main() {
        TI();
        __enable_irq();

        while(1) {
```

```
                update();
                __WFI();
        }
}
```

DataStructure.h

```
#ifndef DATASTRUCTURE_H
#define DATASTRUCTURE_H

#include "LPC407x_8x_177x_8x.h"

#define GPIO_ADDRESS      0x20098000

#define PORT0    ((PORT_TypeDef*) PORT0_BASE)
#define PORT1    ((PORT_TypeDef*) PORT1_BASE)
#define PORT2    ((PORT_TypeDef*) PORT2_BASE)
#define PORT3    ((PORT_TypeDef*) PORT3_BASE)
#define PORT4    ((PORT_TypeDef*) PORT4_BASE)
#define PORT5    ((PORT_TypeDef*) PORT5_BASE)

#define PORT0_BASE                (GPIO_ADDRESS + 0x000)
#define PORT1_BASE                (GPIO_ADDRESS + 0x020)
#define PORT2_BASE                (GPIO_ADDRESS + 0x040)
#define PORT3_BASE                (GPIO_ADDRESS + 0x060)
#define PORT4_BASE                (GPIO_ADDRESS + 0x080)
#define PORT5_BASE                (GPIO_ADDRESS + 0x0A0)

typedef struct {
  volatile       uint32_t DIR;
                                                    uint32_t RESERVED0[3];
  volatile       uint32_t MASK;
  volatile       uint32_t PIN;
  volatile       uint32_t SET;
  volatile   uint32_t CLR;
} PORT_TypeDef;

#endif
```

GPIO.c

```
#include "GPIO.h"

void GPIO_PIN_Write(PORT_TypeDef* PORT, uint32_t MASK, uint8_t value) {
        if(value == 0) {
                PORT->PIN &= ~MASK;
        }
        else {
                PORT->PIN |= MASK;
        }
```

```
}
```

GPIO.h

```
#ifndef GPIO_H
#define GPIO_H

#include "LPC407x_8x_177x_8x.h"
#include "DataStructure.h"

void GPIO_PIN_Write(PORT_TypeDef* PORT, uint32_t MASK, uint8_t value);

#endif
```

LCD.c

```
#include "LCD.h"

void LCD_Init(void) {
        RS_PORT -> DIR |= RS_MASK;
        RW_PORT -> DIR |= RW_MASK;
        EN_PORT -> DIR |= EN_MASK;
        DATA0_PORT -> DIR |= DATA0_MASK;
        DATA1_PORT -> DIR |= DATA1_MASK;
        DATA2_PORT -> DIR |= DATA2_MASK;
        DATA3_PORT -> DIR |= DATA3_MASK;


        LCD_command(0x03);
        LCD_command(0x03);
        LCD_command(0x03);
        LCD_command(0x02);

        LCD_command(0x28);

        LCD_cursorBlinking();

        LCD_cursorAutoIncrement();

        LCD_clearDisplay();

        LCD_setCursorHome();
}

void LCD_command(uint8_t data) {
        GPIO_PIN_Write(RS_PORT,RS_MASK,0);

        GPIO_PIN_Write(DATA0_PORT,DATA0_MASK,(data >> 4 & (0x01)));
        GPIO_PIN_Write(DATA1_PORT,DATA1_MASK,(data >> 5 & (0x01)));
        GPIO_PIN_Write(DATA2_PORT,DATA2_MASK,(data >> 6 & (0x01)));
```

```
        GPIO_PIN_Write(DATA3_PORT,DATA3_MASK,(data >> 7 & (0x01)));

        LCD_toogle();

        GPIO_PIN_Write(DATA0_PORT,DATA0_MASK,(data >> 0 & (0x01)));
        GPIO_PIN_Write(DATA1_PORT,DATA1_MASK,(data >> 1 & (0x01)));
        GPIO_PIN_Write(DATA2_PORT,DATA2_MASK,(data >> 2 & (0x01)));
        GPIO_PIN_Write(DATA3_PORT,DATA3_MASK,(data >> 3 & (0x01)));

        LCD_toogle();
}

void LCD_data(uint8_t data) {
        GPIO_PIN_Write(RS_PORT,RS_MASK,1);

        GPIO_PIN_Write(DATA0_PORT,DATA0_MASK,(data >> 4 & (0x01)));
        GPIO_PIN_Write(DATA1_PORT,DATA1_MASK,(data >> 5 & (0x01)));
        GPIO_PIN_Write(DATA2_PORT,DATA2_MASK,(data >> 6 & (0x01)));
        GPIO_PIN_Write(DATA3_PORT,DATA3_MASK,(data >> 7 & (0x01)));

        LCD_toogle();

        GPIO_PIN_Write(DATA0_PORT,DATA0_MASK,(data >> 0 & (0x01)));
        GPIO_PIN_Write(DATA1_PORT,DATA1_MASK,(data >> 1 & (0x01)));
        GPIO_PIN_Write(DATA2_PORT,DATA2_MASK,(data >> 2 & (0x01)));
        GPIO_PIN_Write(DATA3_PORT,DATA3_MASK,(data >> 3 & (0x01)));

        LCD_toogle();
}

void LCD_write(char* data) {
        while(*data > 0)  {
                LCD_data(*data++);
        }
}

void LCD_toogle() {
        wait(10);
        EN_PORT->PIN |= EN_MASK;
        wait(10);
        EN_PORT->PIN &= ~EN_MASK;
        wait(10);
}

void LCD_clearDisplay() {
        LCD_command(0x01);
}
```

```
void LCD_clearDisplayWithoutRAM() {
        LCD_command(0x08);
}

void LCD_cursorON() {
        LCD_command(0x0E);
}

void LCD_cursorOFF() {
        LCD_command(0x0C);
}

void LCD_cursorBlinking() {
        LCD_command(0x0F);
}

void LCD_cursorAutoIncrement() {
        LCD_command(0x06);
}

void LCD_shiftLeft() {
        LCD_command(0x18);
}

void LCD_shiftRight() {
        LCD_command(0x1C);
}

void LCD_moveCursorLeft() {
        LCD_command(0x10);
}

void LCD_moveCursorRight() {
        LCD_command(0x14);
}

void LCD_setCursorHome() {
        LCD_command(0x02);
}

void LCD_setCursorPositionFirstLine(uint8_t position) {
        LCD_command(0x80 + position);
}

void LCD_setCursorPositionSecondLine(uint8_t position) {
        LCD_command(0xC0 + position);
}
```

LCD.h

```
#ifndef LCD_H
#define LCD_H

#include "GPIO.h"
#include "Wait.h"

#define RS_PORT                 PORT0
#define RS_MASK                 ((uint32_t) 1 << 8)

#define RW_PORT                 PORT0
#define RW_MASK                 ((uint32_t) 1 << 6)

#define EN_PORT                 PORT0
#define EN_MASK                 ((uint32_t) 1 << 7)

#define DATA0_PORT      PORT0
#define DATA0_MASK      ((uint32_t) 1 << 24)

#define DATA1_PORT      PORT0
#define DATA1_MASK      ((uint32_t) 1 << 25)

#define DATA2_PORT      PORT0
#define DATA2_MASK      ((uint32_t) 1 << 26)

#define DATA3_PORT      PORT1
#define DATA3_MASK      ((uint32_t) 1 << 30)

void LCD_Init(void);

void LCD_command(uint8_t data);

void LCD_data(uint8_t data);
void LCD_write(char* data);

void LCD_toogle(void);

void LCD_clearDisplay(void);
void LCD_clearDisplayWithoutRAM(void);
void LCD_cursorON(void);
void LCD_cursorOFF(void);
void LCD_cursorBlinking(void);
void LCD_cursorAutoIncrement(void);
void LCD_shiftLeft(void);
void LCD_shiftRight(void);
void LCD_moveCursorLeft(void);
void LCD_moveCursorRight(void);
void LCD_setCursorHome(void);
```

```
void LCD_setCursorPositionFirstLine(uint8_t position);
void LCD_setCursorPositionSecondLine(uint8_t position);

#endif
```

Serial.c

```
#include "Serial.h"

char serialReceivedCharacter = 0;
char* serialTransmitData = 0;
uint8_t serialTransmitCompleted = 0;

void Serial_Init() {
        //Change the function of TX and RX pins for UART.
        Serial_UART_TX_PIN |= (1<<0);
        Serial_UART_TX_PIN &= ~(1<<1);
        Serial_UART_TX_PIN &= ~(1<<2);

        Serial_UART_RX_PIN |= (1<<0);
        Serial_UART_RX_PIN &= ~(1<<1);
        Serial_UART_RX_PIN &= ~(1<<2);

        //Turn on UART0.
        PCONP |= (1<<3);

        //Enable FIFO for UART0.
        Serial_UART->FCR |= (1<<0);

        //In order to change the DLM, DLL and FDR values, Write correct code for enabl
        Serial_UART->LCR |= (1<<7);

        //Write correct DLM, DLL and FDR values for 9600 baudrate
        Serial_UART->DLM = 0x01;
        Serial_UART->DLL = 0x25;
        Serial_UART->FDR = 0x01 << 0 | 0x03 <<4;

        //Write correct code for disabling the access to Divisor Latches.
        Serial_UART->LCR &= ~(1<<7);

        //Change LCR register value for 8-bit character transfer, 1 stop bits and Even
        Serial_UART->LCR = 3<<0 | 0<<2| 1<<3 | 1<<4;

        //Enable the Receive Data Available and THRE Interrupt.
        Serial_UART->IER |= (1<<0);
        Serial_UART->IER |= (1<<1);

        //Enable UART0_IRQn Interrupt.
        NVIC_EnableIRQ(UART0_IRQn);
```

```
        //Set UART0_IRQn Priority to 5.
        NVIC_SetPriority(UART0_IRQn, 5);

}

void UART0_IRQHandler() {
        uint32_t currentInterrupt = ((Serial_UART->IIR & (0x7 << 1)) >> 1);

        //For Receive Data Available interrupt.
        if(currentInterrupt == 0x2) {
                serialReceivedCharacter = Serial_ReadData();
        }
        //For THRE interrupt
        else if(currentInterrupt == 0x1) {
                if(*serialTransmitData > 0) {
                        Serial_WriteData(*serialTransmitData++);
                }
                else {
                        serialTransmitCompleted = 1;
                }
        }
}

char Serial_ReadData() {
        return Serial_UART->RBR;
}

void Serial_WriteData(char data) {
        serialTransmitCompleted = 0;
        Serial_UART->THR = data;
}
```

Serial.h

```
#ifndef SERIAL_H
#define SERIAL_H

#include "LPC407x_8x_177x_8x.h"

#include "SystemStructures.h"

#pragma anon_unions

typedef struct
{
        union
        {
                volatile  uint8_t  RBR;
```

```
                volatile  uint8_t  THR;
                volatile         uint8_t  DLL;
                                                        uint32_t RESERVED0;
        };
        union
        {
                volatile         uint8_t  DLM;
                volatile         uint32_t IER;
        };
        union
        {
                volatile  uint32_t IIR;
                volatile  uint8_t  FCR;
        };
        volatile         uint8_t  LCR;
                                                uint8_t  RESERVED1[7];
        volatile  uint8_t  LSR;
                                                uint8_t  RESERVED2[7];
        volatile         uint8_t  SCR;
                                                uint8_t  RESERVED3[3];
        volatile         uint32_t ACR;
        volatile         uint8_t  ICR;
                                                uint8_t  RESERVED4[3];
        volatile         uint8_t  FDR;
                                                uint8_t  RESERVED5[7];
        volatile         uint8_t  TER;
                                                uint8_t  RESERVED8[27];
        volatile         uint8_t  RS485CTRL;
                                                uint8_t  RESERVED9[3];
        volatile         uint8_t  ADRMATCH;
                                                uint8_t  RESERVED10[3];
        volatile         uint8_t  RS485DLY;
                                                uint8_t  RESERVED11[3];
        volatile  uint8_t  FIFOLVL;
}UART_TypeDef;

//Write the base address of the UART0.
#define Serial_UART_BASE         0x4000C000
#define Serial_UART      ((UART_TypeDef*) Serial_UART_BASE)

//Write the IOCON address of TX Pin
#define Serial_UART_TX_PIN_ADDRESS       0x4002C008
#define Serial_UART_TX_PIN       *((volatile uint32_t*)(Serial_UART_TX_PIN_ADDRESS))

//Write the IOCON address of RX Pin
#define Serial_UART_RX_PIN_ADDRESS       0x4002C00C
#define Serial_UART_RX_PIN       *((volatile uint32_t*)(Serial_UART_RX_PIN_ADDRESS))
```

```
extern char serialReceivedCharacter;
extern char* serialTransmitData;
extern uint8_t serialTransmitCompleted;

void Serial_Init(void);
char Serial_ReadData(void);
void Serial_WriteData(char data);

#endif
```

SystemStructures.h

```
#ifndef SYSTEM_STRUCTURES_H
#define SYSTEM_STRUCTURES_H

//Write the address of Power Control for Peripherals Register
#define PCONP_ADDRESS   0x400FC0C4
#define PCONP    *((volatile uint32_t*)(PCONP_ADDRESS))

//Write PCLK Frequency
#define PERIPHERAL_CLOCK_FREQUENCY 0x3938700

#endif
```

Timer.c

```
#include "Timer.h"

void Ultrasonic_Capture_Timer_Init() {
        PCONP |= 1 << 23;

        TIMER3->CTCR = 0x0;

        TIMER3->TCR &= ~(1 << 0);

        TIMER3->TCR |= (1 << 1);

        TIMER3->PR = PERIPHERAL_CLOCK_FREQUENCY / 1000000 - 1;

        TIMER3->CCR = (1 << 0) | (1 << 2);

        TIMER3->TCR &= ~(1 << 1);

        TIMER3->TCR |= (1 << 0);

        NVIC_EnableIRQ(TIMER3_IRQn);
}

void ISR_echoCaptureCounter() {
        if(ultrasonicSensorCaptureRisingEdge == 1) {
```

```
                LPC_TIM3->CCR = (1 << 1) | (1 << 2);
                ultrasonicSensorCaptureRisingEdge = 0;
        }
        else {
                ultrasonicSensorDuration = TIMER3->CR0;
                ultrasonicSensorNewDataAvailable = 1;

                LPC_TIM3->CCR = (1 << 0) | (1 << 2);
                ultrasonicSensorCaptureRisingEdge = 1;
        }

        TIMER3->IR = 1 << 4;
        TIMER3->TC = 0;
}
```

Timer.h

```
#ifndef TIMER_H
#define TIMER_H

#include "LPC407x_8x_177x_8x.h"
#include "Ultrasonic.h"

typedef struct
{
  volatile       uint32_t IR;
  volatile       uint32_t TCR;
  volatile       uint32_t TC;
  volatile       uint32_t PR;
  volatile       uint32_t PC;
  volatile       uint32_t MCR;
  volatile       uint32_t MR0;
  volatile       uint32_t MR1;
  volatile       uint32_t MR2;
  volatile       uint32_t MR3;
  volatile       uint32_t CCR;
  volatile       uint32_t CR0;
  volatile       uint32_t CR1;
                                                uint32_t RESERVED0[2];
  volatile       uint32_t EMR;
                                                uint32_t RESERVED1[12];
  volatile       uint32_t CTCR;
} TIMER_TypeDef;

#define TIMER0_BASE      0x40004000
#define TIMER1_BASE      0x40008000
#define TIMER2_BASE      0x40090000
#define TIMER3_BASE      0x40094000
```

```
#define TIMER0   ((TIMER_TypeDef*) TIMER0_BASE)
#define TIMER1   ((TIMER_TypeDef*) TIMER1_BASE)
#define TIMER2   ((TIMER_TypeDef*) TIMER2_BASE)
#define TIMER3   ((TIMER_TypeDef*) TIMER3_BASE)

#endif
```

Ultrasonic.c

```
#include "Ultrasonic.h"

uint32_t ultrasonicSensorDuration = 0;
uint32_t ultrasonicSensorDistance = 0;
uint8_t ultrasonicSensorNewDataAvailable = 0;

uint8_t ultrasonicSensorTriggerStart = 0;
uint8_t ultrasonicSensorCaptureRisingEdge = 0;

void Ultrasonic_Init() {
        //Give the Correct Function Values to IOCON_TRIGGER and IOCON_ECHO
        IOCON_TRIGGER &= ~(1 << 2); IOCON_TRIGGER |= (1 << 1); IOCON_TRIGGER |= (1 <<

        IOCON_ECHO &= ~(1 << 2); IOCON_ECHO |= (1 << 1); IOCON_ECHO |= (1 << 0);
}

void Ultrasonic_Trigger_Timer_Init() {
        //Enable Timer2.
        PCONP |= (1 << 22);

        //Change the mode of Timer2 to Timer Mode.
        TIMER2 -> CTCR &= ~(1 << 0);
        TIMER2 -> CTCR &= ~(1 << 1);

        //Disable Timer Counter and Prescale Counter for Timer2.
        TIMER2 -> TCR &= ~(1 << 0);

        //Reset Timer Counter and Prescale Counter for Timer2.
        TIMER2 -> TCR |= (1 << 1);

        //Change PR Register value for 1 microsecond incrementing
        TIMER2 -> PR = 59;

        //Write the Correct Configuration for EMR (Toggle Output Value and Initial val
        TIMER2 -> EMR |= (1 << 3);
        TIMER2 -> EMR |= (1 << 10);
        TIMER2 -> EMR |= (1 << 11);

        //Enable TIMER2_IRQn (Interrupt Request).
        NVIC_EnableIRQ(TIMER2_IRQn);
```

```
        //Set Priority Timer2 IRQ as 5.
        NVIC_SetPriority(TIMER2_IRQn, 5);

        //Clear pendings for Timer2.
        NVIC_ClearPendingIRQ(TIMER2_IRQn);
}

void Ultrasonic_Start_Trigger_Timer() {
        //Give correct value to MR3 Register for 10 microsecond
        TIMER2 -> MR3 = 10;

        //Enable interrupt for MR3 register, if MR3 register matches the TC.
        TIMER2 -> MCR |= (1 << 9);

        //Remove the reset on counters of Timer2.
        TIMER2 -> TCR &= ~(1 << 1);

        //Enable Timer Counter and Prescale Counter for counting.
        TIMER2 -> TCR |= (1 << 0);
}

void ISR_ultrasonicTriggerToggler() {
        if(ultrasonicSensorTriggerStart == 0) {
                //Change MR3 Register Value for Suggested Waiting
                TIMER2->MR3 = 59990;
                ultrasonicSensorTriggerStart = 1;
        }
        else {
                TIMER2->MR3 = 10;
                ultrasonicSensorTriggerStart = 0;
        }

        //Write HIGH bit value to IR Register for Corresponding Interrupt
        TIMER2->IR |= (1 << 3);
        TIMER2->TC = 0;
}
```

Ultrasonic.h

```
#ifndef ULTRASONIC_H
#define ULTRASONIC_H

#include "GPIO.h"
#include "Timer.h"
#include "SystemStructures.h"

//Write IOCON Register Address of Trigger Pin.
#define IOCON_TRIGGER_ADDRESS    0x4002C024
```

```
#define IOCON_TRIGGER    *((volatile uint32_t*)(IOCON_TRIGGER_ADDRESS))

//Write IOCON Register Address of Echo Pin.
#define IOCON_ECHO_ADDRESS       0x4002C05C
#define IOCON_ECHO       *((volatile uint32_t*)(IOCON_ECHO_ADDRESS))

extern uint32_t ultrasonicSensorDuration;
extern uint32_t ultrasonicSensorDistance;
extern uint8_t ultrasonicSensorNewDataAvailable;

extern uint8_t ultrasonicSensorTriggerStart;
extern uint8_t ultrasonicSensorCaptureRisingEdge;

void Ultrasonic_Init(void);
void Ultrasonic_Trigger_Timer_Init(void);
void Ultrasonic_Capture_Timer_Init(void);

void Ultrasonic_Start_Trigger_Timer(void);

#endif
```

Wait.c

```
#include "Wait.h"

void wait(uint32_t miliseconds) {
        uint32_t i;
        uint32_t totalDuration = miliseconds*24000;
        for(i=0;i<totalDuration;i++);
}

void waitMicroseconds(uint32_t microseconds) {
        uint32_t i;
        uint32_t totalDuration = microseconds*24;
        for(i=0;i<totalDuration;i++);
}
```

Wait.h

```
#ifndef WAIT_H
#define WAIT_H

#include "LPC407x_8x_177x_8x.h"

void wait(uint32_t miliseconds);

void waitMicroseconds(uint32_t microseconds);

#endif
```

Module: Getting Speed Data from Input Device

main.c

```
#include "LPC407x_8x_177x_8x.h"

#include "CarSpeedOutput.h"
#include "PWM.h"
#include "Wait.h"
#include "MMA7455.h"
#include "Serial.h"
#include <stdio.h>

char stringValue [30];
int angleX = 0, carOutput = 0;
int values[3] = {0,0,0};

void TI() {
        MMA7455_Init();
        MMA7455_setMode(1);
        MMA7455_calibrate();
        CarOutputInit();
        PWM_Init();
        Serial_Init();
}

uint32_t mapAccValue(int acc){
        return (uint32_t)(acc/63.0*999);
}

// Task Get Acceleration
void TGA()
{
        MMA7455_read(&values[0], &values[1], &values[2]);
        angleX = values[0];
        if(angleX < 0) angleX = 0;
        else if(angleX > 63) angleX = 63;
}

//Task Set Speed
void TSS()
{
        carOutput = mapAccValue(angleX);
        PWM_Write(carOutput);
}

// Task System Diagnosis
void TSD()
{
        sprintf(stringValue ,"x, y, z, carOutput= \t%d, \t%d, \t%d \t%d\r\n" , values[
        serialTransmitData = stringValue;
```

```
        Serial_WriteData(*serialTransmitData++);
        while(!serialTransmitCompleted);
}

void update() {
        TGA();
        TSS();
        TSD();
}

int main() {
        TI();
        while(1) {
                update();
        }
}
```

CarSpeedOutput.c

```
#include "CarSpeedOutput.h"

/*uint8_t clapCount = 0;
uint8_t clapCountRisingEdge = 0;
uint8_t clapCountChanged = 0;
*/
void CarOutputInit() {
        IOCON_CAR_OUTPUT_PIN |= (1 << 0);
        IOCON_CAR_OUTPUT_PIN |= (1 << 1);
        IOCON_CAR_OUTPUT_PIN &= ~(1 << 2);
}
```

CarSpeedOutput.h

```
#ifndef CARSPEEDOUTPUT_H
#define CARSPEEDOUTPUT_H

#include "LPC407x_8x_177x_8x.h"

/*extern uint8_t clapCount;
extern uint8_t clapCountRisingEdge;
extern uint8_t clapCountChanged;
*/
#define IOCON_CAR_OUTPUT_PIN_ADDRESS     0x4002C094
#define IOCON_CAR_OUTPUT_PIN     *((volatile uint32_t*)(IOCON_CAR_OUTPUT_PIN_ADDRESS))

void CarOutputInit(void);

#endif
```

I2C.c

```
#include "I2C.h"

void I2C_Init() {
        //Turn on I2C0
        PCONP |= 1 << 7;

        I2C0->SCLL = I2CDutyCycle;
        I2C0->SCLH = I2CDutyCycle;

        I2C0->CONCLR =  (1 << 5)
                                                                | (1 << 3)
                                                                | (1 << 2);

        //In Initialization routine, Write Correct value to CONSET register for Master
        I2C0->CONSET = 0x40;

        I2C_Data_PIN &= 0x03;
        I2C_Data_PIN |= 0x01;

        I2C_Clock_PIN &= 0x03;
        I2C_Clock_PIN |= 0x01;
}

int I2C_Start() {
        int status = 0;

        I2C0->CONCLR =  (1 << 5)
                                                                | (1 << 3)
                                                                | (1 << 2);

        I2C0->CONSET =  (1 << 2);

        //In Start Master Transmit function, Write Correct Value to CONSET register
        I2C0->CONSET = 0x20;

        I2C_Wait_SI();

        status = I2C0->STAT;

        I2C0->CONCLR =  (1 << 5);

        return status;
}

int I2C_Stop() {
        int timeout = 0;

        I2C0->CONSET =  (1 << 4);
```

```
        I2C0->CONCLR =  (1 << 3);

        while(I2C0->CONSET & (1 << 4)) {
                timeout ++;
                if (timeout > 100000) return 1;
        }

        return 0;
}

int I2C_Write(uint32_t address, const char* data,int length, int repeated) {
        int stop;
        int written;

        if(repeated == 0) {
                stop = 1;
        }
        else {
                stop = 0;
        }

        written = I2C_WriteData(address, data, length, stop);

        return length != written;
}

int I2C_WriteData(uint32_t address, const char* data,int length, int stop) {
        int status;
        int i;

        status = I2C_Start();

        //Stop and Return When A START or A repeated START Condition has not been tran
        if ((status != 0x08) && (status != 0x10)) {
                I2C_Stop();
                return -1;
        }

        status = I2C_DoWrite(address & 0xFE);

        //Stop and Return When First Data Byte is not Transmitted
        if (status != 0x18) {
                I2C_Stop();
                return -1;
        }

        for (i=0; i<length; i++) {
```

```
                status = I2C_DoWrite(data[i]);
                //Stop and Return When Data is not Transmitted
                if (status != 0x28) {
                        I2C_Stop();
                        return i;
                }
        }

        if (stop) {
                I2C_Stop();
        }

        return length;
}

int I2C_Read(uint32_t address, char* data, int length, int repeated) {
        int stop;
        int readed;
        if(repeated == 0) {
                stop = 1;
        }
        else {
                stop = 0;
        }

        readed = I2C_ReadData(address, data, length, stop);

        return length != readed;
}

int I2C_ReadData(uint32_t address, char* data, int length, int stop) {
        int status;
        int count;

        int value;
        status = I2C_Start();

         if ((status != 0x10) && (status != 0x08)) {
                I2C_Stop();
                return -1;
        }

        status = I2C_DoWrite(address | 0x01);

        //Stop and Return When ACK is not received
        if (status != 0x40) {
                I2C_Stop();
                return -1;
```

```
        }

        for (count = 0; count < (length - 1); count++) {
                int value = I2C_DoRead(0);
                status = I2C0->STAT;

                //Stop and Return When Data is not received
                if (status != 0x50) {
                        I2C_Stop();
                        return count;
        }

                data[count] = value;
        }

        value = I2C_DoRead(1);

        status = I2C0->STAT;

        //Stop and Return When Last Data is not received
        if (status != 0x58) {
                I2C_Stop();
                return length - 1;
        }

        data[count] = value;

        if (stop) {
                I2C_Stop();
        }

        return length;
}

int I2C_DoWrite(int value) {
        I2C0->DAT = value;

        I2C0->CONCLR = (1 << 3);

    I2C_Wait_SI();
    return I2C0->STAT;
}

int I2C_DoRead(int last) {
        if(last) {
                I2C0->CONCLR = (1 << 2);

        }
```

```
        else {
                I2C0->CONSET = (1 << 2);
        }

        I2C0->CONCLR =  (1 << 3);

        I2C_Wait_SI();

        return (I2C0->DAT & 0xFF);
}

int I2C_Wait_SI() {
    int timeout = 0;
    while (!(I2C0->CONSET & (1 << 3))) {
        timeout++;
        if (timeout > 100000) return -1;
    }
    return 0;
}
```

I2C.h

```
#ifndef I2C_H
#define I2C_H

#include "LPC407x_8x_177x_8x.h"
#include "SystemStructures.h"
#include "Wait.h"

typedef struct
{
  __IO uint32_t CONSET;
  __I  uint32_t STAT;
  __IO uint32_t DAT;
  __IO uint32_t ADR0;
  __IO uint32_t SCLH;
  __IO uint32_t SCLL;
  __O  uint32_t CONCLR;
  __IO uint32_t MMCTRL;
  __IO uint32_t ADR1;
  __IO uint32_t ADR2;
  __IO uint32_t ADR3;
  __I  uint32_t DATA_BUFFER;
  __IO uint32_t MASK0;
  __IO uint32_t MASK1;
  __IO uint32_t MASK2;
  __IO uint32_t MASK3;
} I2C_TypeDef;
```

```
//Write the Base Address of I2C0
#define I2C0_BASE        0x4001C000
#define I2C0     ((I2C_TypeDef*) I2C0_BASE)

//Write the IOCON address of  I2C0 Data Input/Output
#define I2C_Data_PIN_ADDRESS     0x4002C06C
#define I2C_Data_PIN     *((volatile uint32_t*)(I2C_Data_PIN_ADDRESS))

//Write the IOCON address of  I2C0 Clock Input/Output
#define I2C_Clock_PIN_ADDRESS    0x4002C070
#define I2C_Clock_PIN    *((volatile uint32_t*)(I2C_Clock_PIN_ADDRESS))

//Write I2CDutyCycle which is (I2CSCLL + I2CSCLH) / 2
#define I2CDutyCycle 300

void I2C_Init(void);

int I2C_Start(void);

int I2C_Stop(void);

int I2C_Write(uint32_t address, const char* data,int length, int repeated);

int I2C_WriteData(uint32_t address, const char* data,int length, int stop);

int I2C_Read(uint32_t address, char* data, int length, int repeated);

int I2C_ReadData(uint32_t address, char* data, int length, int stop);

int I2C_DoWrite(int value);

int I2C_DoRead(int last);

int I2C_Wait_SI(void);

#endif
```

MMA7455.c

```
#include "MMA7455.h"

void MMA7455_Init() {
        I2C_Init();
}

int MMA7455_setMode(int mode) {
        int isModeStarted = 0;
        int controlMode = 0;
```

```
        do {
                controlMode = MMA7455_getModeControl();
                if (controlMode < 0) break;

                controlMode &= ~(0x03 << 0);
                controlMode |= mode << 0;

                if (MMA7455_setModeControl(controlMode)) {
                        break;
                }

                isModeStarted = 1;
        } while(0);

        return isModeStarted;
}

int MMA7455_getModeControl() {
    int result = -1;
    char data[1];

    do {
        data[0] = MMA7455ModeControlRegister;
        if (I2C_Write(MMA7455IC2Address, data, 1, 0) != 0) break;

        if (I2C_Read(MMA7455IC2Address, data, 1, 0) != 0) break;

        result = data[0];
    } while (0);

    return result;
}

int MMA7455_setModeControl(uint8_t controlMode) {
        int result = -1;
   char data[2];

    do {
        data[0] = MMA7455ModeControlRegister;
        data[1] = (char)controlMode;
        if (I2C_Write(MMA7455IC2Address, data, 2, 0) != 0) break;

        result = 0;

    } while (0);
        return result;
}
```

```
int MMA7455_calibrate() {
                int result = 0;
    int failed = 0;
                int i;

    int32_t x = 0;
    int32_t y = 0;
    int32_t z = 0;

    int32_t xr = 0;
    int32_t yr = 0;
    int32_t zr = 0;

    int xOff = 0;
    int yOff = 0;
    int zOff = 16;

    do {
        for (i = 0; i < 6; i++) {
            if (!MMA7455_read(&xr, &yr, &zr)) {
                failed = 1;
                break;
            }
            x += xr;
            y += yr;
            z += zr;

            wait(100);
        }

        if (failed) break;

        x /= 6;
        y /= 6;
        z /= 6;

        xOff -= x;
        yOff -= y;
        zOff -= z;

        xOffset = xOff;
        yOffset = yOff;
        zOffset = zOff;

        result = 1;
    } while (0);
    return result;
}
```

```
int MMA7455_read(int* x, int* y, int* z) {
        int result = 0;
        int status = 0;
        char buf[6];

    do {
      status = MMA7455_getStatus();
    } while (status >= 0 && (status & (1 << 0)) == 0);


    do {
        if (status < 0) break;

        buf[0] = 0x00;
        if (I2C_Write(MMA7455IC2Address, buf, 1,0) != 0) break;
        if (I2C_Read(MMA7455IC2Address, buf, 6,0) != 0) break;

        if (buf[1] & 0x02) buf[1] |= 0xFC;
        if (buf[3] & 0x02) buf[3] |= 0xFC;
        if (buf[5] & 0x02) buf[5] |= 0xFC;

        *x = (int16_t)((buf[1] << 8) | buf[0]) + xOffset;
        *y = (int16_t)((buf[3] << 8) | buf[2]) + yOffset;
        *z = (int16_t)((buf[5] << 8) | buf[4]) + zOffset;


        result = 1;

    } while(0);

    return result;
}

int MMA7455_getStatus() {
    int result = -1;
    char data[1];

    do {
        data[0] = MMA7455StatusRegister;
        if (I2C_Write(MMA7455IC2Address, data, 1,0) != 0) break;

        if (I2C_Read(MMA7455IC2Address, data, 1,0) != 0) break;

        result = data[0];

    } while (0);
```

```
    return result;
}
```

MMA7455.h

```
#ifndef MMA7455_H
#define MMA7455_H

#include "I2C.h"
#include "Wait.h"

//Write the 1 left shifted version of 7-bit I2C address of MMA7455
#define MMA7455IC2Address 0x1D << 1

//Write the correct value of Mode Control register
#define MMA7455ModeControlRegister 0x16 //????????????????????? bune

//Write the correct value of Status register of MMA7455
#define MMA7455StatusRegister 0x09 //????????????????????? bune

void MMA7455_Init(void);

int MMA7455_setMode(int mode);
int MMA7455_getModeControl(void);
int MMA7455_setModeControl(uint8_t controlMode);

int MMA7455_calibrate(void);

int MMA7455_read(int* x, int* y, int* z);
int MMA7455_getStatus(void);

static int xOffset = 0;
static int yOffset = 0;
static int zOffset = 0;

#endif
```

PWM.c

```
#include "PWM.h"

void PWM_Init() {
        //Turn on PWM
        PCONP |= (1<<5);

        //Enable PWM output for corresponding pin.
        PWMX->PCR |= (1<<11);

        //PWM gives pulse every 20 ms.
        PWMX->MR0 = (PERIPHERAL_CLOCK_FREQUENCY / 1000000) * 20 * 1000;
```

```
        PWMX->MR3 = 0;

        //Enable PWM Match Latch 0 and Latch 3.
        PWMX->LER |= (1<<0);
        PWMX->LER |= (1<<3);

        //Write the Correct Values to TCR for Enabling Counter and PWM
        PWMX->TCR |= (1<<0);
        PWMX->TCR |= (1<<3);
        PWM_Write(0);
}

void PWM_Write(uint32_t value) {
        uint32_t trueValue;

        if(value > 1000) {
                value = 1000;
        }

        trueValue = (uint32_t)(((PWMX->MR0) * value) / 1000);

        if (trueValue == PWMX->MR0) {
                trueValue++;
        }

        PWMX->MR3 = trueValue;

        PWMX->LER |= 1 << 3;
}
```

PWM.h

```
#ifndef PWM_H
#define PWM_H

#include "LPC407x_8x_177x_8x.h"

#include "SystemStructures.h"

typedef struct
{
  volatile      uint32_t IR;
  volatile      uint32_t TCR;
  volatile      uint32_t TC;
  volatile      uint32_t PR;
  volatile      uint32_t PC;
  volatile      uint32_t MCR;
  volatile      uint32_t MR0;
  volatile      uint32_t MR1;
```

```
  volatile       uint32_t MR2;
  volatile       uint32_t MR3;
  volatile       uint32_t CCR;
  volatile  uint32_t CR0;
  volatile  uint32_t CR1;
  volatile  uint32_t CR2;
  volatile  uint32_t CR3;
                                                    uint32_t RESERVED0;
  volatile       uint32_t MR4;
  volatile       uint32_t MR5;
  volatile       uint32_t MR6;
  volatile       uint32_t PCR;
  volatile       uint32_t LER;
                                                    uint32_t RESERVED1[7];
  volatile       uint32_t CTCR;
} PWM_TypeDef;

#define PWM0_BASE        0x40014000
#define PWM1_BASE        0x40018000

#define PWM0     ((PWM_TypeDef*) PWM0_BASE)
#define PWM1     ((PWM_TypeDef*) PWM1_BASE)

//Change PWMX with that correct PWM.
#define PWMX_BASE        PWM0_BASE
#define PWMX     ((PWM_TypeDef*) PWMX_BASE)

void PWM_Init(void);
void PWM_Write(uint32_t value);

#endif
```

Serial.c

```
#include "Serial.h"

char serialReceivedCharacter = 0;
char* serialTransmitData = 0;
uint8_t serialTransmitCompleted = 0;

void Serial_Init() {
        //Change the function of TX and RX pins for UART.
        Serial_UART_TX_PIN |= (1<<0);
        Serial_UART_TX_PIN &= ~(1<<1);
        Serial_UART_TX_PIN &= ~(1<<2);

        Serial_UART_RX_PIN |= (1<<0);
        Serial_UART_RX_PIN &= ~(1<<1);
        Serial_UART_RX_PIN &= ~(1<<2);
```

```
        //Turn on UART0.
        PCONP |= (1<<3);

        //Enable FIFO for UART0.
        Serial_UART->FCR |= (1<<0);

        //In order to change the DLM, DLL and FDR values, Write correct code for enabl
        Serial_UART->LCR |= (1<<7);

        //Write correct DLM, DLL and FDR values for 9600 baudrate
        Serial_UART->DLM = 0x01;
        Serial_UART->DLL = 0x25;
        Serial_UART->FDR = 0x01 << 0 | 0x03 <<4;

        //Write correct code for disabling the access to Divisor Latches.
        Serial_UART->LCR &= ~(1<<7);

        //Change LCR register value for 8-bit character transfer, 1 stop bits and Even
        Serial_UART->LCR = 3<<0 | 0<<2| 1<<3 | 1<<4;

        //Enable the Receive Data Available and THRE Interrupt.
        Serial_UART->IER |= (1<<0);
        Serial_UART->IER |= (1<<1);

        //Enable UART0_IRQn Interrupt.
        NVIC_EnableIRQ(UART0_IRQn);

        //Set UART0_IRQn Priority to 5.
        NVIC_SetPriority(UART0_IRQn, 5);

}

void UART0_IRQHandler() {
        uint32_t currentInterrupt = ((Serial_UART->IIR & (0x7 << 1)) >> 1);

        //For Receive Data Available interrupt.
        if(currentInterrupt == 0x2) {
                serialReceivedCharacter = Serial_ReadData();
        }
        //For THRE interrupt
        else if(currentInterrupt == 0x1) {
                if(*serialTransmitData > 0) {
                        Serial_WriteData(*serialTransmitData++);
                }
                else {
                        serialTransmitCompleted = 1;
                }
```

```
        }
}

char Serial_ReadData() {
        return Serial_UART->RBR;
}

void Serial_WriteData(char data) {
        serialTransmitCompleted = 0;
        Serial_UART->THR = data;
}
```

Serial.h

```
#ifndef SERIAL_H
#define SERIAL_H

#include "LPC407x_8x_177x_8x.h"

#include "SystemStructures.h"

#pragma anon_unions

typedef struct
{
        union
        {
                volatile  uint8_t  RBR;
                volatile  uint8_t  THR;
                volatile        uint8_t  DLL;
                                                uint32_t RESERVED0;
        };
        union
        {
                volatile        uint8_t  DLM;
                volatile        uint32_t IER;
        };
        union
        {
                volatile  uint32_t IIR;
                volatile  uint8_t  FCR;
        };
        volatile        uint8_t  LCR;
                                                uint8_t  RESERVED1[7];
        volatile  uint8_t  LSR;
                                                uint8_t  RESERVED2[7];
        volatile        uint8_t  SCR;
                                                uint8_t  RESERVED3[3];
        volatile        uint32_t ACR;
```

```
        volatile        uint8_t  ICR;
                                                uint8_t  RESERVED4[3];
        volatile        uint8_t  FDR;
                                                uint8_t  RESERVED5[7];
        volatile        uint8_t  TER;
                                                uint8_t  RESERVED8[27];
        volatile        uint8_t  RS485CTRL;
                                                uint8_t  RESERVED9[3];
        volatile        uint8_t  ADRMATCH;
                                                uint8_t  RESERVED10[3];
        volatile        uint8_t  RS485DLY;
                                                uint8_t  RESERVED11[3];
        volatile  uint8_t  FIFOLVL;
}UART_TypeDef;

//Write the base address of the UART0.
#define Serial_UART_BASE        0x4000C000
#define Serial_UART     ((UART_TypeDef*) Serial_UART_BASE)

//Write the IOCON address of TX Pin
#define Serial_UART_TX_PIN_ADDRESS      0x4002C008
#define Serial_UART_TX_PIN      *((volatile uint32_t*)(Serial_UART_TX_PIN_ADDRESS))

//Write the IOCON address of RX Pin
#define Serial_UART_RX_PIN_ADDRESS      0x4002C00C
#define Serial_UART_RX_PIN      *((volatile uint32_t*)(Serial_UART_RX_PIN_ADDRESS))

extern char serialReceivedCharacter;
extern char* serialTransmitData;
extern uint8_t serialTransmitCompleted;

void Serial_Init(void);
char Serial_ReadData(void);
void Serial_WriteData(char data);

#endif
```

SystemStructures.h

```
#ifndef SYSTEM_STRUCTURES_H
#define SYSTEM_STRUCTURES_H

#define PCONP_ADDRESS   0x400FC0C4
#define PCONP   *((volatile uint32_t*)(PCONP_ADDRESS))

#define PERIPHERAL_CLOCK_FREQUENCY 60000000

#endif
```

Wait.c

```
#include "Wait.h"

void wait(uint32_t miliseconds) {
        uint32_t i;
        uint32_t totalDuration = miliseconds*24000;
        for(i=0;i<totalDuration;i++);
}

void waitMicroseconds(uint32_t microseconds) {
        uint32_t i;
        uint32_t totalDuration = microseconds*24;
        for(i=0;i<totalDuration;i++);
}
```

Wait.h

```
#ifndef WAIT_H
#define WAIT_H

#include "LPC407x_8x_177x_8x.h"

void wait(uint32_t miliseconds);

void waitMicroseconds(uint32_t microseconds);

#endif
```