Main c

LAB 5

```
#include LPC407x_8x_177x_8x.h
#include "Library/GPIO.h"
#include "Library/PWM.h"
#include "Library/Joystick.h"

int speed = 0;

void init() {
    Joystick_Init();
    //Initialize Direction Pins
    PORT1->DIR |= (1<<7);
    PORT1->DIR |= (1<<6);

    PWM init();
    PWM_Write(speed);
}

void update() {
// When joystick left B. is pressed, Motor rotation direction will be changed to forwad
if (Joystick_Left_Pressed()) {
    uint32_t value = PORT1->PIN;
    value |= (1<<7);
    value &= ~(1<<6);
    PORT1->PIN = value;
    PWM_Write(speed);

// When Joys. Up button is pressed, Motor speed will be increased
if (Joyst_Up_Pressed()) {
    speed++;
    for (int i=0; i<1000000; i++) {
    }
    PWM_Write(speed);

// When Down Button pressed, speed decr
if (Joystick_Down_Pressed()) {
    speed--;
    for (int j=0; j<1000000; j++) {
    }
    PWM_Write(speed);

//When CenterB is pressed => Stop
if (Joys_Center_Press()) {
    uint32_t value = PORT1->PIN;
    value &= ~(1<<7);
    value &= ~(1<<6);
    PORT1->PIN = value;

// When Right B is pressed, Motor rot dir. will be changed to backword.
if (Joystick_Right_Press()) {
    uint32_t value = PORT1->PIN;
    value |= (1<<6);
    value &= ~(1<<7);
    PORT1->PIN = value;
    PWM_Write(speed);

//When Joyst. no button is pressed, Motor will continue to perform the last actio

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

PWM.c

```
void PWM_Init () {
// Change the dir. of the pin in here
PCONP |=1<<5;
IOCON_MOTOR_SPEED &= ~(1<<2);
IOCON_MOTOR_SPEED |= (1<<0) | (1<<1);

//Enable PWM output for corresponding pin.
    PWM0->PCR |= (1<<10);

// Reset the PWM Timer Counter and the
PWM Prescale Counter on the next Pos. Edge
of PCLK
  PWM0->TCR |= (1<<1);
  PWM0->MR0 = (PERIPHERAL_CLOCK_FREQ
                /1000000) * 20 * 1000;

//Reset TC, when MR0 matches TC
  PWM0->MCR |= (1<<1);

//Enable PWM Match 0 Latch
  PWM0->LER |= (1<<0);

//Enable Counter and PWM and clear Reset
on the PWM
  PWM0->TCR |= (1<<0) | (1<<3);
  PWM0->TCR &= ~(1<<1);
   PWM_Write (0);
}
 void PWM_Cycle_Rate (uint32_t _period in
Cycles) {
 PWM0->MR0 = (Peripheral_Clock_Freq
                /1000000) *
            period in cycle * 1000;

//enable PWM Match 0 Latch
    PWM0->LER |= (1<<0); }
```

```
void PWMwrite (uint32_t T_ON) {
  if (T_ON > 100) {
  T_ON = 100; }

T_ON = (uint32_t) (((PWM0->MR0) * T_ON)/100);

if (T_ON == PWM0->MR0) {
   T_ON ++;
}
PWM0->MR2 = T_ON;

//Enable PWM Match Register Latch.
 PWM0->LER |= (1<<2);
```

LAB 6

Main.c

```
Void init () {

 LED_init ();
 Timer_Init ();
}
```

// When the time between rising and falling or falling - rising edge is less than 100 milis. and greater than 5 miliseconds, turn on the LED on the Quick Start Board Otherwise turn off the LED on the Quick start board.

```
Void update () {
  current-time = TIMER3 -> CR1;
 if ((current Time - previous Time) != 0)
   { captured Time = current-time - previ,time;

 if ( ( cap.Time > 5000 ) && (capT. <100000))
  }
  LED_ON ();
 }
 else {
   LED_OFF ();
   previ.time = current-time ;}

}}
int main () {
      init ();
    While (1) {
          update ();
      }
}
```

Timer.c

```
uint32_t current-time = 0;
         previous-time = 0;
         captured-time = 0;

void Timer_Init () {
// Give the correct Func. values to
 IOCON-OUT
 IOCONOUT |= (1 << 0) | (1<< 1);
 IOCONOUT &= ~ (1<< 2);

//Enable timer 3
  PCONP | = (1<< 23);
 // Change the mode of Timer 3 to Timer mode
Timer 3 -> CCR = 0;
// Disable Timer Counter and Res.C.
 TIMER3 -> TCR &= ~(1<< 0);
//Reset TimC. and Pres. C. for Timer 3
 TIMER 3 -> TCR |= (1<< 1);
//Change PR register value for 1micro second incrementing
 TIMER3 -> PR = 59;
//Capture rising and Falling Edge on CAP1
 Timer 3 -> CCR | = ((1<< 3 | (1<< 4));
// Remove the reset on counters of Tim3
 TIMER 3 -> TCR &= ~ (1<< 1);
//Enable Timer 3 counter and Prescale counter for counting
 TIMER 3 -> TCR | = (1<< 0);

 }
```

Timer.h

```
typedef struct {
 volatile uint32_t IR;
                   TCR;
                   TC;
                   PR;
                   PC;
                   :
                   :
                   CR1;
                   reseved[3];
                   :
} Timer_Typedef
```

```
define  TIMER0_BASE.   0x40004000.
          1   "            "
          2   "            "
          3   "            "
```

```
define  TIMER0   ((Timetypedef *) Timer0_Base )
          1          "              1
          2                         2
          3          "              3
                     "
```

```
define IOCON_OUT_ADDR   0x4002C060.
  "    IOCONOUT * ((volatile uint32_t *) (IOCON_OUT_ADDR))
```

```
extern uint32_t  currenttime
  "        "     prevtime
  "        "     capt. time

void  Timer_int (void);
```

Lab 07

Ultrasonic_Trigger_Timer_Init()

LOCON_Trigger |= 3; → 0x4002C026

PCONP |= 1<<22;

TM2→CTCR &= ~(3); → change mode to Timer

Tm2→TCR |= (1<<1) → Reset timer counter and prescale counter

TM2→PR = PER...CLOCK_FREQ / 1000000 - 1 → change PRRegister val

TM2→TCR &= ~(1<<1) → remove reset on counters

Ultrasonic_Capture_Timer_Init()

IOCON_Echo |= 3 → 0x4002C060

PCONP |= 1<<23;

Tm3→CTCR = 0x02

Tm3→TCR &= ~(1<<0)

Tm3→TCR |= (1<<1)

PR = → same

Tm3→CCR |= (1<<3)
Tm3→CCR |= (1<<5)
Tm3→CCR &= ~(1<<4)
→ interrupt when rising Edge occur

Tm3→TCR &= ~(1<<1)

TCR |= (1<<0)

NVIC_ClearPendingIRQ(TIMER3_IRQn);

NVIC_SetPriority(TIMER3_IRQn, 5);

NVIC_EnableIRQ(TIMER3_IRQn);

Ultrasonic_start_trigger()

Tm2→EMR |= (1<<3) → give high value to T2_MAT_3 pin

→MR3 = Tm2→TC + 10 → 10 ms HIGH value

→EMR &= ~(1<<11)
→EMR |= (1<<10) } making LOW output value when match occurs

→MCR |= (1<<11)
→MCR |= (1<<10) } Reset TC and Stop (TC and PC) if matches TC

→TCR |= (1<<0) → enable counter and PS counter for counting

ultrasonic Sensor EdgeCount = 0

Lab 7 Cont

TIMER3_IRQHandler()

```
if (sensorEdgeCount == 0) {
    sensorRaisingTime = TM3->CR1 -> store rising time
    TM3->ccR 1= (1<<4)
         ->ccR 8= ~(1<<3)
         ->ccR 1= (1<<5)        } -> getting interrupt when falling edge event is occured
    sensorEdgeCount = 1
    NVIC_ClearPendingIRQ(TIMER3_IRQn)
} else if (sensorEdgeCount == 1) {
    sensorfallingTime = TM3->CR1
    TM3->ccR 1= (1<<3)
         ->ccR 8= ~(1<<4)
         ->ccR 1= (1<<5)
    SensorEdgeCount = 2;
    clearPending
}
TIMER3->IR 1= (1<<5) -> clear interrupt flag for capture channel 1 event
```

In main

```
sensorDistance = (fallingTime - risingTime) / 58
wait(600);
```

LAB 8

US _Timer_Init()

IOCON_TRIGGER |= 0x03;
PCONP |= 1<<22;
TIMER2->CTCR = 0x00
TIMER2->TCR &= ~(1<<0)
TIMER->TCR |= (1<<1)
TIMER2->PR = Per---_Clock_Freq... /1000000-1

- Write correct Conf. for EMR

uint32_t value = Timer2->EMR;
value |= (1<<3)
value |= (1<<10)
value |= (1<<11)
TIMR2->EMR = value;

- Enable TIMER2_IRQn

NVIC_EnableIRQ(TIMER2_IRQn)

- Set Priority Timer2 IRQ as 5

NVIC_~~Set~~SetPriority(TIMER2_IRQn, 5)

- Clear Pending for TIMER2

NVIC_ClearPendingIRQ(TIMER2_IRQn)

void Ultrasonic_Capture_Thier_Init()

IOCON_ECHO |= 0x03
PCONP |= 1<<23
TIMER3->CTCR = 0x00
TIMER3->TCR &= ~(1<<0);
TIMER3->TCR |= (1<<1);
TIMER3->PR = Perip-Clock_freq /1000000-1;
TIMER3->CCR = (1<<3) | (1<<4) | (1<<5)
TIMER3->TCR &= ~(1<<1)

void Ultrasonic_Start_Trigger()

- Give correct value to MR3 Reg. for 10 mic.
TIMER2->MR3 = 10
- Enable interrupt for MR3 register, of MR3 reg.
TIMER2->MCR |= (1<<9);
- Remove the reset on counters of TIMER2
TIMER2->TCR &= ~(1<<1)
- Enable Tier2 Counter and Prescale Counter for
TIMER2->TCR |= (1<<0)

lab 8 Demo

```
uint8_t  isUltrasonicSensorTriggerEnded = 0
int      isUltrasonicSensorEdgeCount = 0

Void TIMER2_IRQHandler()
    if(isUltrasonicSensorTriggerEnded == 0){
        - Change MR3 Register Value for Suggested
        TIMER2->MR3 = 60*1000
                    = 1
                    = 0

    // Clear pending for Timer3
    NVIC_ClearPendingIRQ(TIMER3_IRQn);

    // Enable Timer
    NVIC_EnableIRQ(TMR3_IRQn)
    } else {
        TM2->MR3 = 10
        isUltrasonicSensorTriggerEnded = 0;
    }
    TM2->IR |= (1<<3) -> clear IR Flag
    TM1->TC = 0;

void TM3_IRQHandler()
    if (sensorEdgeCount == 0){
        risingTime = TM3->CR1
        edgeCount = 1;
        NVIC_ClearPendingIRQ(TM3_IRQn);
    } else if (sensorEdgeCount == 1){
        fallingTime = TM3->CR1
        edgeCount = 2
        NVIC_ClearPendingIRQ(TIMER3_IRQn);
        NVIC_DisableIRQ(TIMER3_IRQn);
    }
    TM3->IR = 1<<5;
```

Modes

Sleep mode

```
SCR &= ~4
PCON &= ~3
__WFI();
```

Deep sleep Mode

```
SCR |= 4;
PCON &= ~3
__WFI()
```

Power Down

```
SCR |= 4
PCON &= ~(1<<1)
PCON |= (1<<0)
```

Deep Power Down → WFI()

```
SCR |= 4
PCON |= 3
__WFI()
```

Lab9

ADC.c

- Change the mode value of pin to mode

  //inactive = 00 (4:3 bits)

  ANALOG_PIN_IOCON &= ~(24);

- Change Analog/Digital mode of pin to Analog

  ANALOG ---- &= ~(1<<7)

- Turn on ADC

  PCONP |= (1<<12);

- Set the CLKDIV and make the A/D converter

```
uint_32_t  val2 = ADC->CR
           val2 &= ~(0xFF<<8)
           val 2 |= [ADC_CLKDIV<<8)
           ADC->CR = val2

           ADC->CR |= (1<<21)
           ADC->CR &= ~(1<<16)
```

```
void ADC_Start()
  -Write a code for starting A/D conversion
  uint_32  val = ADC->CR
           val &= ~(7<<24)
           val |= (1<<24)
           ADC->CR = value;
```

```
void ADC_Stop()
  uint32  val = ADC->CR;
          val &= ~(7<<24)
          ADC->CR = val
```

```
uint32_t ADC_Read()
    uint32_t data;
    -Configure CR SEL bits for sampled and converting correspond.
     ADC->CR |= (1<<12) //ADO[2]
    ADC_Start();
    uint32_t  DONE_MASK = (1<<31);
    while(!(ADC->DR[2] & DONE_MASK));
    -Convert the data RESULT to 0-1000 rage and return ...
     uint32_t RESULT_MASK = (0xFFF <<4);
      //        result = (ADC->DR[2] & Result_MASK)>>4;
                data = (result * 1000.0)/ADC_MAX_VALUE;
                ADC_Stop();
```

Lab 10

Serial_init()

Set pin func for TX and RX

PCONP |= (1<<3)

Serial_UART->FCR |= (1<<0) → enable fifo

UART->LCR |= (1<<7) → enable Divisor Latches

→ DLM = 0x01

→ DLL = 0x75

→ FDR = 0x01 << 0 | 0x03 << 3

→ LCR &= ~(1<<7) → disable divisor latches

→ LCR = 3<<0 | 0<<2 | 1<<3 | 1<<4; → change 8 bit
1 Stop
even parity

serial_Write Data(char data)

while((Serial_UART->LSR & (1<<5)) == 0); → wait until THR become empty

serial_UART->THR = data; → write data to THR

char Serial_Read Data()

char data;

while((Serial_UART->LSR & (1<<0)) == 0); → wait until Receiver Data Ready

data = Serial_UART->RBR; → Read data from receive buffer register

return data;

int charToInt(char data)

int val = data - '0'

return val;

Serial_Write("\r\n") → new line

char resultString[12];
sprintf(resultString, "%d\r\n", result)
→ int to string