LAB 10 - Serial Communication - UART

1) Initialize Serial Communication

Serial communication is the common method of transmitting data between a microcontroller and a peripheral device. LPC4088 includes 5 UARTs (Universal Asynchronous Receiver and Transmitter). In order to use serial communication, P0_2 and P0_3 pins and UART0 will be used. Therefore find and write into Library/Serial.h file the following information:

- Write the base address of the UART0. `0x4000C000`
- Write the IOCON address of TX Pin. `0x4002C008`
- Write the IOCON address of PX Pin `0x4002C00C`

In order to use, P0_2 and P0_3 as UART TX or RX, function of these pins should be changed. Find and write into Library/Serial.c file:

- Change the function of TX (P0_2) and RX (P0_3) pins for UART.

```
uint32_t func = Serial_UART_TX_PIN;
func |= 1<<0;
func &= ~((1<<1) | (1<<2));
Serial_UART_TX_PIN = func;
func = Serial_UART_RX_PIN;
func |= 1<<0;
func &= ~((1<<1) | (1<<2));
Serial_UART_RX_PIN = func;
```

In order to use UART0 for serial communications, UART0 should be configured correctly. In Library/Serial.c file:

- Turn on UART0. `PCONP |= (1<<3);`
- Enable FIFO for UART0. `Serial_UART->FDR |= (1<<0);`

In order to change the baudrate of serial communication, DLM, DLL and FDR registers should be correctly assigned as explained in page 508 of User Manual of LPC4088 (UM10562.pdf).

Find the baudrate of the following configurations by showing every step:

DLM	DLL	DIVADDVAL	MULVAL	BAUDRATE	BAUDRATE (normalized)
1	37	1	3	9598.976	9600
0	47	5	13	57624.11	57600
0	179	1	11	19203.91	19200
0	19	5	7	115131.6	115200

- DLM = 0x01; DLL = 0x25; FDR = 0x01 << 0 | 0x03 << 4; `BAUDRATE = (60000000) / (16 * (256*1 + 37) * (1 + (1/3))) = 9598.976`
- DLM = 0x00; DLL = 0x2F; FDR = 0x05 << 0 | 0x0D << 4; `BAUDRATE = (60000000) / (16 * (256*0 + 47) * (1 + (5/13))) = 57624.11`
- DLM = 0x00; DLL = 0xB3; FDR = 0x01 << 0 | 0x0B << 4;

`BAUDRATE = (60000000) / (16 * (256*0 + 179) * (1 + (1/11))) = 19203.91`

- DLM = 0x00; DLL = 0x13; FDR = 0x05 << 0 | 0x07 << 4; `BAUDRATE = (60000000) / (16 * (256*0 + 19) * (1 + (5/7))) = 115131.6`

In order to change these values for 9600 baudrate (60 MHz PCLK) in Library/Serial.c file, do the following:

- Enable access to Divisor Latches. `Serial_UART->LCR |= (1<<7);`
- What is DIVADDVAL value? `0x01`
- What is MULVAL value? `0x03`
- Write correct DLM, DLL and FDR values for 9600 baudrate.

```
Serial_UART->DLM |= (0x01<<0);
Serial_UART->DLL |= (0x25<<0);
Serial_UART->FCR = ((0x01<<0) | (0x03<<4));
```

- Disable access to Divisor Latches. `Serial_UART->LCR &= ~(1<<7);`

For Serial Communication, you will use 8-bit character transfer, 1 stop bits and even parity configuration. In Library/Serial.c file:

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

2) Reading and Writing via UART

Via serial communication, you can send 8 bit data (1 character) to other devices and you can get 8 bit data from other devices too.

In order to send a 8 bit data, you should write the data to Transmit Holding Register (THR). The next character to be transmitted is written in Transmit Holding Register. In Serial_WriteData method:

- Wait until Transmit Holding Register become empty.
  `while (!(Serial_UART->LSR & 1<<5)) {};`
- Write data to Transmit Holding Register
  `Serial_UART->THR = data;`

In order to get a 8 bit data, you should read the data from Receiver Buffer Register (RBR). This register contains the next received character to be read... In Serial_ReadData method:

- Wait until Receiver Data Ready.
  `while (!(Serial_UART->LSR & 1<<0)) {};`
- Read data from Receiver Buffer Register.
  `data = Serial_UART->RBR;`

Firstly, you should find the COM name by using Device Manager in Windows. In Device Manager:

- What is the COM name next to the mbed Serial Port in your computer? `COM6`

In Serial Communication, there are standard baudrates which are 1200,

2400, 4800, 9600, 19200, 38400, 57600, and 115200. You configured UART0 as 9600. Also you configured Serial Communication as 8 Data bits, 1 Stop bit and Even Parity.

You can check the Serial Communication by using this code in update:

```
Serial_Write(“Hello World\r\n”);
wait(1000);
```

- Change the charToInt method for returning integer value of a character.

```
int charToInt(char c) {
    int value;
    value = c - '0';
    return value;
}
```

Implement a simple calculator

```
void update() {
    char resultString[12];
    int result = 0;
    int num0 = 0;
    int num1 = 0;
    char op;
    int isSecond = 0;
```

```
    for(int i=0; i<12; i++) {
        char c = Serial_ReadData();

        if (c == '=') {
            if (op == '+') {
                result = num0 + num1;
            }
            else if (op == '-') {
                result = num0 - num1;
            }
            else if (op == 'x') {
                result = num0 * num1;
            }
            else if (op == '/') {
                result = num0 / num1;
            }
            sprintf(resultString, "\r\n%d\r\n", result);

            Serial_WriteData(c);
            Serial_Write(resultString);
            break;

        } else if (c == '+' || c == '-' || c == 'x' || c ==
= '/') {
            op = c;
            isSecond = 1;
        } else {
            if (!isSecond) {
```

```
                num0 = num0 * 10 + charToInt(c);
            } else {
                num1 = num1 * 10 + charToInt(c);
            }
        }
        Serial_WriteData(c);
    }

    wait(1000);
}
```

LAB 9 - Analog to Digital Converter

We will read light value of the LDR sensor via ADC.

Light Density artarsa LDR Resistor Azalır Voltaj da azalır.

- Write the IOCON address of ADC Pin which is connected to the board: `0x4002C064`

A/D Converter will work slowly in this experiment and A/D Converter Clock should be 1MHz. (PCLK = 60 MHz)

- Write the max value of ADC to ADC_MAX_VALUE: `0xFFF`
- Write the CLKDIV of ADC for 1 MHz. `59`

Using a pin for A/D Converter, you should change functionality of the pin. However, this is not enough for ADC. You should also change MODE and ADMODE of the pin.

- Change the function value of pin to ADC:

```
uint32_t v = ANALOG_PIN_IOCON;
v |= (1<<0);
v &= ~((1<<1) | (1<<2));
ANALOG_PIN_IOCON = v;
```

- Change the mode value of pin to mode which should be selected if Analog mode is used:
  `ANALOG_PIN_IOCON &= ~((1<<4) | (1<<3));`
- Change Analog/Digital mode of pin to Analog.
  `ANALOG_PIN_IOCON &= ~(1<<7);`
  After pin initialization, you should initialize ADC:
- Turn on ADC (PCONP).
  `PCONP |= (1<<12);`
- Set the CLKDIV and make the A/D converter operational without Burst mode.

```
uint32_t va = ADC->CR;
va |= ((1<<14) |(1<<13) |(1<<12) |(1<<9));
va &= ~((1<<15) |(1<<11) |(1<<10) |(1<<8));
va |= (1<<21);
va &= ~(1<<16);
ADC->CR = va;
```

In this experiment ADC should have a stop and start functionality. For this case:

- Write a code for starting A/D conversion. (In ADC_Start method):

```
uint32_t va = ADC->CR;
va &= ~((1<<26) | (1<<25));
va |= (1<<24);
ADC->CR = va;
```

- Write a code for stopping A/D conversion. (In ADC_Stop method):
  `ADC->CR &= ~((1<<26) | (1<<25) | (1<<24));`
  In software-controlled mode, only one of CR SEL bits should be 1.
- Configure CR SEL bits for sampled and converting corresponding pin. (In ADC_Read)

```
uint32_t v = ADC->CR;
v |= (1<<2);
v &= ~((1<<0) | (1<<1) | (1<<3) | (1<<4) | (1<<5) | (1<<6)
 | (1<<7));
ADC->CR = v;
```

Data Registers (DRs) contains the converted data. However, only specific bit range has the converted data. (ADC_Read)

- Which DR register is used for storing the result of corresponding pin?: `DR2[15:4]`
- Wait until A/D conversion completed.

```
uint32_t mask = 4;
while(!ADC->STAT & mask){};
```

- Convert the data RESULT to 0 - 1000 range and return the ADC data.

```
uint32_t mask1 = 0;
mask1 |= (ADC_MAX_VALUE<<4);
data = ((ADC->DR[2] & mask1)>>4);
data = (data * 1000) / ADC_MAX_VALUE;
```

Changing LED On/Off State by LDR

Under the LAB Lighting condition, LED1 and LED2 should be turned on.
When the light intensity is decreased, LED2 will be turned off and then LED1.
When the light intensity is increased, LED3 will be turned on and then LED4.

```
LDRDataValue = ADC_Read();

    if (state == 0) {
        LED1_On();
        LED2_On();
        LED3_Off();
        LED4_Off();
    }
```

```
    if (LDRDataValue > 950 && state != 1) {
        LED2_Off();
        wait(500);
        LED1_Off();
        LED3_Off();
        LED4_Off();
        state = 1;
    }
    else if (LDRDataValue < 770 && state != 2) {
        LED1_On();
        LED2_On();
        LED3_On();
        wait(500);
        LED4_On();
        state = 2;
    }
    else if (LDRDataValue > 770 && LDRDataValue < 950){
        state = 0;
    }
```

LAB 8 - Power Saving & Multiple Interrupts

When you pressed the push button on the LPC4088 which is shown in the figure, you can give External Interrupt to board.

External Interrupt

- What is the pin name of the pin which is connected to Push Button? `P2_0`
- Write the IOCON address of the push button to IOCON_PUSH_BUTTON_ADDRESS. `0x4002C128`
- Write the address of the External Interrupt Flag register to EXT_ADDRESS `0x400FC140`
- Change the functionality of the push button as EINT0

```
uint32_t value = EXT->EXTINT;
value &= ~((1<<1) | (1<<2));
value |= (1<<0);
EXT->EXTINT = value;
```

- Change the External interrupt mode as Edge Sensitive: `EXT->EXTMODE |= (1<<0);`
- Enable interrupt for EINT0_IRQn. `NVIC_EnableIRQ(EINT0_IRQn);`
- Whenever an interrupt occurs, it should be cleared in handler. Clear interrupt for EINT0. `EXT->EXTINT &= ~(1<<0);`

Methods for Low Power

In order to change the power mode of the board. You will use SCR (System Control Register) and PCON register.

- Write the address of these registers to the SystemStructures.h file. `SCR: 0xE000ED10`

`PCON:0x400FC0C0`

a) Sleep Mode

In this part, you should change the LAB_8 code for entering the sleep mode. Your code should go to Sleep Mode after you pressed Joystick Left Button and turn on all the 4 LEDs. (When you pressed the push button on the LPC4088 which is shown in the figure, the EINT0_IRQHandler will be called and board will wake up)

- What should be the value of SCR register value for Sleep Mode? `0`
- What should be the value of PCON register value for Sleep Mode? `00`
- Write the necessary code in SleepMode method and call it when Joystick Left Button is pressed.

```
SCR &= ~(1<<2);
PCON &= ~((1<<0) | (1<<1));
__WFI();
```

- Find Current while LED on in the Sleep Mode. `3.1mV`

(Hint: You can check it is in sleep mode or not by pressing joystick center button. If no led is turned off, it means it is in sleep mode) (Hint: Do not forget __WFI();)

b) Deep Sleep Mode

Your code should go to Deep Sleep Mode after you pressed Joystick

Right Button and turn on all the 4 LEDs.

- What should be the value of SCR register value for Deep Sleep Mode? `1`
- What should be the value of PCON register value for Deep Sleep Mode? `00`
- Write the necessary code in DeepSleepMode method and call it when Joystick Right Button is pressed.

```
SCR |= (1<<2);
PCON &= ~((1<<0) | (1<<1));
__WFI();
```

- Find Current while LED on in the Deep Sleep Mode: `2.2 mV`

c) Power Down Mode

Your code should go to Power Down Mode after you pressed Joystick Up Button and turn on all the 4 LEDs.

- What should be the value of SCR register value for Power Down Mode? `1`
- What should be the value of PCON register value for Power Down Mode? `01`
- Write the necessary code in PowerDownMode method and call it when Joystick Up Button is pressed.

```
SCR |= (1<<2);
uint32_t value = PCON;
value |= (1<<0);
```

```
value &= ~(1<<1);
PCON = value;
__WFI();
```

- Find Current while LED on in the Power Down Mode: `2.2 mV`

d) Deep Power Down Mode

Your code should go to Deep Power Down Mode after you pressed Joystick Up Button and turn on all the 4 LEDs.

- What should be the value of SCR register value for Deep Power Down Mode? `1`
- What should be the value of PCON register value for Deep Power Down Mode? `11`
- Write the necessary code in DeepPowerDownMode method and call it when Joystick Down Button is pressed.

```
SCR |= (1<<2);
PCON |= ((1<<0) | (1<<1));
__WFI();
```

- Find Current in the Deep Power Down Mode. `1.9 mV`

Ultrasonic Sensor with Interrupts

We will use Timer2 and Timer3 interrupts and also you will send trigger signal via Timer2 interrupt.

HC - SR04 Pins	LPC4088 Pins

Vcc	VU
GND	GND
Trig	P11 (P0_9)
Echo	P16 (P0_24)

In Ultrasonic_Trigger_Timer_Init (in Ultrasonic.c file) method:

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

In Ulrasonic_Start_Trigger (in Ultrasonic.c file) method:

- 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 Timer2 Counter and Prescale Counter for counting.

`TIMER2->TCR |= (1<<0);`

In TIMER2_IRQHandler (in Ultrasonic.c file):

- Change MR3 Register Value for Suggested Waiting
  `TIMER2->MR3 = 60000;`
- Clear pendings for Timer3.
  `NVIC_ClearPendingIRQ(TIMER3_IRQn);`
- Enable TIMER3_IRQn (Interrupt Request).
  `NVIC_EnableIRQ(TIMER3_IRQn);`
- Clear IR Register Flag for Corresponding Interrupt
  `TIMER2->IR |= (1<<3);`

In TIMER3_IRQHandler (in Ultrasonic.c file):

- Clear pendings for Timer3.
  `NVIC_ClearPendingIRQ(TIMER3_IRQn);`
- Disable TIMER3_IRQn (Interrupt Request).
  `NVIC_DisableIRQ(TIMER3_IRQn);`

4) Using Ultrasonic Sensor

In this part, you will write codes into update method for:

- When Ultrasonic Sensor detects obstacles which are closer than 7cm, turn on all the LEDs.
- When Ultrasonic Sensor detects obstacles which are in the range of 7cm and 12cm , turn on LED1, LED2 and LED3.
- When Ultrasonic Sensor detects obstacles which are in the range of 12cm and 20cm , turn on LED1 and LED2.

- When Ultrasonic Sensor detects obstacles which are in the range of 20cm and 30cm , turn on LED1.
- When Ultrasonic Sensor detects obstacles which are far from 30 cm, turn off all the LEDs.

```
PowerConsumptionTest();
    ultrasonicSensorDuration = ultrasonicSensorFallingTime
 - ultrasonicSensorRisingTime;
    ultrasonicSensorDistance = ultrasonicSensorDuration /
58;

    if (ultrasonicSensorDistance < 7) {
        LED1_On();
        LED2_On();
        LED3_On();
        LED4_On();
    }
    else if (7 < ultrasonicSensorDistance && ultrasonicSen
sorDistance < 12) {
        LED1_On();
        LED2_On();
        LED3_On();
        LED4_Off();
    }
    else if (12 < ultrasonicSensorDistance && ultrasonicSe
nsorDistance < 20) {
        LED1_On();
        LED2_On();
```

```
        LED3_Off();
        LED4_Off();
    }
    else if (20 < ultrasonicSensorDistance && ultrasonicSe
nsorDistance < 30) {
        LED1_On();
        LED2_Off();
        LED3_Off();
        LED4_Off();
    }
    else if (ultrasonicSensorDistance > 30) {
        LED1_Off();
        LED2_Off();
        LED3_Off();
        LED4_Off();
    }
```

Ekstras

```
//Implement Sleep Mode
void SleepMode() {
    SCR &= ~(1<<2);
    PCON &= ~((1<<0) | (1<<1));
    __WFI();
}
//Implement Deep Sleep Mode
void DeepSleepMode() {

```

```
    SCR |= (1<<2);
    PCON &= ~((1<<0) | (1<<1));
    __WFI();
}
//Implement Power Down Mode
void PowerDownMode() {
        SCR |= (1<<2);
        uint32_t value = PCON;
        value |= (1<<0);
        value &= ~(1<<1);
        PCON = value;
        __WFI();
}
//Implement Deep Power Down Mode
void DeepPowerDownMode() {
    SCR |= (1<<2);
    PCON |= ((1<<0) | (1<<1));
    __WFI();
}
```

LAB 7 - Ultrasonic Sensor & Timers

1) Ultrasonic Sensor

HC - SR04 Ultrasonic sensor has 4 pins which are VCC, Trigger, Echo and GND.

In order to use HC - SR04 Ultrasonic sensor, you should give a trigger signal which is a square signal. After that ultrasonic will send 8 cycle

burst of ultrasound at 40KHz and it will give HIGH value to its Echo pin. The Echo is a distance object that is pulse width and the range in proportion which means that:

> *If obstacle goes away, width of the HIGH value in Echo will increase.*
> *If obstacle approaches, width of the HIGH value in Echo will decrease.*

- What is the min range of HC - SR04? **2 cm**
- What is the max range of HC - SR04? **400 cm**
- What is the min time length of HIGH level signal for triggering the HC - SR04? **10 uS**
- What is the min suggested measurement cycle length for HC - SR04? **60 ms**

2) Connecting Ultrasonic Sensor to LPC4088

Connect the HC - SR04 pins to LPC4088:

HC - SR04 Pins	LPC4088 Pins
Vcc	VU
GND	GND
Trig	P11 (P0_9)
Echo	P16 (P0_24)

3) Initialize Ultrasonic Sensor

In order to initialize Ultrasonic Sensor, you should find and write into the Library/Ultrasonic.h file:

- IOCON register address of Trigger Pin. **0x4002C024**
- IOCON register address of Echo Pin. **0x4002C060**

Trigger Pin will be used as T2_MAT_3 and Echo Pin will be used as T3_CAP_1:

- For Trigger Pin, change the value of FUNC field in IOCON register? **011**
- For Echo Pin, change the value of FUNC field in IOCON register? **011**

(Write the necessary code for Trigger and Echo functionality into Ultrasonic_Trigger_Timer_Init and Ultrasonic_Capture_Timer_Init method which is in Library/Ultrasonic.c file.)

```
uint32_t value = IOCON_TRIGGER;
value |= (1<<0);
value |= (1<<1);
value &= ~(1<<2);
IOCON_TRIGGER = value;

uint32_t value = IOCON_ECHO;
value |= (1<<0);
value |= (1<<1);
value &= ~(1<<2);
```

```
IOCON_ECHO = value;
```

4) Initialize Timer for Trigger

In LPC4088, there are 4 Timer peripherals which are Timer0, Timer1, Timer2 and Timer3.

In order to use Timer, Timer should be turned on. In order to turn on any peripherals, Power Control for Peripherals register (PCONP) is used. When a peripheral control bit is 1 in PCONP, that peripheral is enabled. Find and write into the code:

Trigger Pin is used as T2_MAT_3 and for that, Timer2 should be enabled.

- Enable Timer2 (Library/Ultrasonic.c) (In Ultrasonic_Trigger_Timer_Init).

```
PCONP |= (1<<22);
```

Timer2 will be used as Timer Mode which means that the TC is incremented when the Prescale Counter matches the Prescale Register.

- Change the mode of Timer2 to Timer Mode.

```
TIMER2->CTCR &= ~((1<<0) | (1<<1));
```

- Reset Timer Counter and Prescale Counter for Timer2

```
TIMER2->TCR |= (1<<1);
```

Prescale Counter register which is PC register of Timer is used for incrementing the value of the Timer Counter (TC). The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the

Prescale Register (PR), the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK. This causes the Timer Counter to increment on every PCLK when PR = 0, every 2 PCLK when PR = 1, etc. (every PR+1 cycles of PCLK)

Timer Counters (TC) of Timer2 and Timer3 should be incremented at every 1 microsecond (1 second = 1000000 microsecond).

- What is the PR register value? (Also change the PR register value of Timer2 in Ultrasonic_Trigger_Timer_Init) (You can use PERIPHERAL_CLOCK_FREQUENCY variable)
  `TIMER2->PR = (PERIPHERAL_CLOCK_FREQUENCY / 1000000) - 1;`
- Remove the reset on counters of Timer2.
  `TIMER2->TCR &= ~(1<<1);`

Trigger pin will be used as External Match output for Timer2. External Match means that when a match occurs between the TC and MR, this bit can toggle, go LOW, go HIGH, or do nothing. Trigger Pin is working as T2_MAT_3. Therefore in order to make square signal when TC matches to MR3, it should go low and its initial output value should be HIGH. In Ultrasonic_Trigger_Timer_Start method:

- Give HIGH value to T2_MAT_3 pin by using (TIMER2 register)
  `TIMER2->EMR |= (1<<3);`

Trigger pin should give HIGH value 10 microseconds for activating the Ultrasonic sensor:

- Calculate the MR3 register value for giving 10 microsecond HIGH value

  `TIMER2->MR3 = 10;`
- Write to correct values to Timer2 EMR register for making LOW output value of Trigger Pin when match occurs.

```
uint32_t value = TIMER2->EMR;
value |= (1<<10);
value &= ~(1<<11);
TIMER2->EMR = value;    0.5 pts
```

- Reset TC and Stop (TC and PC), if MR3 register matches the TC. (MCR register)

```
uint32_t v = TIMER2->MCR;
v |= (1<<11);
v |= (1<<10);
v &= ~(1<<9);
TIMER2->MCR = v;
```

- Enable Timer2 Counter and Prescale Counter for counting.

  `TIMER2->TCR |= (1<<0);`

5) Initialize Interrupts

From the previous lab, you already initialized the Timer capture. However, you used the polling method for measuring the sensor data. However, in this lab you will use interrupt for capturing input data.

Interrupt mechanism is used for events which need immediate attention. In our case, Ultrasonic sensor should work at the same frequency apart from main program and when distance value is changed immediately our program should know. Therefore Timer3 is used for capturing Echo Signal.

Every peripheral of the LPC4088 has a different type of external interrupt (See lecture slides for more details). For example, in Timer case, you can configure interrupts by using Match Control Register (MCR) or Capture Control Register (CCR) (Echo pin will use CCR for interrupt) and you have to enable the related interrupts, clear pending interrupts and set the priority of the interrupts with NVIC functions found in CMSIS-Core API. You can access CMSIS-Core API functions in cortex_m4.h. Explanation is given in lecture slides.

External interrupts fall into the category of maskable interrupts. In this category, enabling individual interrupts is necessary but not sufficient for the activation of the interrupt mechanism. You have to remove the system mask. This is provided by __enable_irq() function of CMSIS-Core API.

After enabling interrupts, whenever an interrupt occurs, the Handler method of that component is called. For example, Timer3 interrupt handler is TIMER3_IRQHandler. These names are the default names which are already written in startup_LPC407x_8x.s file. If you open that file, you can see the list of interrupt handlers.

When an interrupt occurs and the Handler method is called, there can be different interrupt sources. For example, TIMER3_IRQHandler can be called from input capture or output compare signals. Therefore for understanding, which interrupt is called, we need additional register for checking it. Every component has different register for that. For example, Timer is using IR (Interrupt Register) for identify which of eight possible interrupt sources are pending.

After getting the interrupt and using it, interrupt should be cleared. This clearing mechanism also changes. For example, in order to clear Timer interrupt, HIGH value should be written to corresponding bit in IR register. If you do not clear it, that same interrupt will be called again.

Therefore for initializing the Timer Interrupts:
In Ultrasonic_Echo_Timer_Init method (Do not forget that you are using Echo pin as T3_CAP_1):

- Write the Correct Value to CCR register for getting interrupt when Rising Edge Occur
  `TIMER3->CCR = ((1<<3) | (1<<5));`
- Clear pending interrupts for Timer3.
  `NVIC_ClearPendingIRQ(TIMER3_IRQn);`
- Set Priority as 5 to this Timer3 IRQ.
  `NVIC_SetPriority(TIMER3_IRQn, 5);`
- Enable TIMER3_IRQn (Interrupt Request).
  `NVIC_EnableIRQ(TIMER3_IRQn);`

Timer3 interrupt will call the TIMER3_IRQHandler method. You already know that distance value is related to width of the falling edge and rising edge of the echo signal. For this purpose in TIMER3_IRQHandler method:

- Store the event time to ultrasonicSensorRisingTime variable, when rising edge is detected.
  `ultrasonicSensorRisingTime = TIMER3->CR1;`
- Change the CCR register value for getting interrupt when falling edge event is occured.
  `TIMER3->CCR |= ((1<<4) | (1<<5));`
- Store the event time to ultrasonicSensorFallingTime variable, when falling edge is detected.
  `ultrasonicSensorFallingTime = TIMER3->CR1;`
- Change the CCR register value for getting interrupt when rising edge event is occurred.
  `TIMER3->CCR |= ((1<<3) | (1<<5));`
- Clear the interrupt flag for capture channel 1 event
  `TIMER3->IR |= (1<<5);`

With this method, we can detect the time distance for the rising and falling edge.

6) Using Ultrasonic Sensor Distance Value

In this part, you will write codes into update method for:

- Convert time to distance (cm) and write the found value into the ultrasonicSensorDistance (Also write the formula here)

- When Ultrasonic Sensor detects obstacles which are closer than 7cm, turn on all the LEDs.
- When Ultrasonic Sensor detects obstacles which are in the range of 7cm and 12cm, turn on LED1, LED2 and LED3. Look below 0.5 pts
- When Ultrasonic Sensor detects obstacles which are in the range of 12cm and 20cm, turn on LED1 and LED2.
- When Ultrasonic Sensor detects obstacles which are in the range of 20cm and 30cm, turn on LED1.
- When Ultrasonic Sensor detects obstacles which are far from 30 cm, turn off all the LEDs.

```
Ulrasonic_Start_Trigger();
wait(600);
ultrasonicSensorDuration = ultrasonicSensorFallingTime - u
ltrasonicSensorRisingTime;
ultrasonicSensorDistance = ultrasonicSensorDuration / 58;
if (ultrasonicSensorDistance < 7) {
LED1_On();
LED2_On();
LED3_On();
LED4_On();
}
else if (7<ultrasonicSensorDistance && ultrasonicSensorDis
tance<12) {
LED1_On();
LED2_On();
LED3_On();
```

```
LED4_Off();
}
else if (12<ultrasonicSensorDistance && ultrasonicSensorDi
stance<20) {
LED1_On();
LED2_On();
LED3_Off();
LED4_Off();
}
else if (20<ultrasonicSensorDistance && ultrasonicSensorDi
stance<30) {
LED1_On();
LED2_Off();
LED3_Off();
LED4_Off();
}
else if (30<ultrasonicSensorDistance) {
LED1_Off();
LED2_Off();
LED3_Off();
LED4_Off();
}
```

(Hint: You should consider the max suggested frequency of the sensor in order to use sensor correctly)
(Hint: You should use Ultrasonic_Trigger_Timer_Start method for starting trigger)

(Hint: ultrasonicSensorEdgeCount become 2 when the falling edge is detected)

(Hint: You can use wait($milisecond$) function for waiting)

LAB 6 - Speedometer & Captures

2) Motor Speed Sensor

Motor Speed Sensor has 3 pins which are VCC, OUT and GND. User manual of the Motor Speed Sensor is given in the Files folder in Canvas. In order to use Motor Speed Sensor sensor, giving power to sensor is sufficient. The signal on the OUT pin gives HIGH and LOW values.

4) Initialize Motor Speed Sensor

In order to initialize OUT pin which is connected to Motor Speed Sensor, you find the IOCON register addresses (Timer.h):

- IOCON register address of OUT Pin: `0x4002C060`

OUT Pin will be used as T3_CAP_1:

- Change the functionality of the pin in Timer.c file

```
uint32_t value = IOCON_OUT;
value |= 1<<0;
value |= 1<<1;
value &= ~(1<<2);
IOCON_OUT = value;
```

5) Initialize Timer for Capture

- In Timer_Init method (Timer.c), power up Timer3
  `PCONP |= 1<<23;`
- Change the mode of Timer3 to Timer Mode
  `TIMER3->CTCR &= ~((1<<0) | (1<<1));`

While making some changes on the Timer, we need to change the Timer state to reset state.

- Disable Timer Counter and Prescale Counter for Timer3
  `TIMER3->TCR &= ~(1<<0);`
- Reset Timer Counter and Prescale Counter for Timer3
  `TIMER3->TCR |= 1<<1;`

Prescale Counter register which is PC register of Timer is used for incrementing the value of the Timer Counter (TC). The Prescale Counter is incremented on every PCLK. When it reaches the value stored in the Prescale Register (PR), the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK. This causes the Timer Counter to increment on every PCLK when PR = 0, every 2 PCLK when PR = 1, etc. (every PR+1 cycles of PCLK).

Timer Counters (TC) of Timer3 should be incremented at every 1 microsecond (1 second = 1000000 microsecond) (PCLK is 60MHz and you can use PERIPHERAL_CLOCK_FREQUENCY variable for this) .

- What is the required number of PCLK cycles for 1 microsecond?

`PERIPHERAL_CLOCK_FREQUENCY / 1000000`

- Change Timer3 PR Register value for 1 microsecond incrementing
  `TIMER3->PR = PERIPHERAL_CLOCK_FREQUENCY / 1000000 - 1;`

OUT pin of motor speed sensor will be used as capturing input. The Capture Control Register (CCR) is used to control whether one of the two Capture Registers (CR0 and CR1) is loaded with the value in the Timer Counter when the capture event occurs, and whether an interrupt is generated by the capture event. OUT Pin is working as T3_CAP_1 and also for calculating speed, falling edge time and rising edge time difference should be know. In Timer_Init method:

- Capture on CAP1 rising edge and falling edge
  `TIMER3->CCR |= ((1<<3) | (1<<4));`
- Remove the reset on counters of Timer3
  `TIMER3->TCR &= ~(1<<1);`
- Enable Timer3 Counter and Prescale Counter for counting
  `TIMER3->TCR |= (1<<0);`

6) Speed Measurement

In this part, you will write codes into update method for:

- When the time between rising - falling or falling - rising edge is less than 100 milliseconds and greater than 5 milliseconds, turn on the LEDs on the Quick Start Board.
- Otherwise turn off the LEDs on the Quick Start Board.

(Hint: You can check time when the rising or falling edge occur from

CR1 register)

(Hint: You can use LED_On, LED_Off etc. methods which are in the LED.c file)

```
current = TIMER3->CR1;


    v = current - prev;
    if (v == 0) {
        return;
    } else {
    }
    if (5000<v && v<100000) {
        LED_ON();
    } else {
        LED_OFF();
    }
    prev = current;
```

LAB 5 - Motor Driver

Motor Driver should take the commands from the board. On L298N Motor Driver, there are 2 Enable pin and 4 Logic pin. In order to use Motor Controller with Board, there should be 5 pin connections:

- Two pin connections for giving power to motor controller and motor,
- One pin connection for changing the speed of the motor

- Two pin connections for changing the direction of the motor.

In this lab, P29 pin is used for PWM. You should determine the 2 pins for the direction control. For this case:

- Write Pin Name of pins which will be used for changing motor rotation direction.
  `P28 (P1_5), P27 (P1_6)`
- Write the Ports of these pins.
  `P28, P27`
- Write the MASK of these pins. (1 << Y)
  `MASK = ((1<<5) | (1<<6));`
- You should connect these 2 pins to IN1 and IN2 pins of the Motor Controller.

The direction pins for the motor controller should be initialized as output.

- Initialize these pins in your init method in the main file.
  `uint32_t mask = ((1<<5) | (1<<6));`
  `GPIO_DIR_Write(PORT1, mask, 0);`

	IN1 = LOW	**IN1 = HIGH**
IN2 = LOW	BREAK	FORWARD
IN2 = HIGH	BACKWARD	BREAK

3) Initialize Pulse Width Modulators (PWM)

Initialize the current PWM pin.

- What is the IOCON register address of the P29 (P1_3) pin?
  `#define IOCON_MOTOR_SPEED_ADDRESS 0x4002C08C`
- Change the function of P29 (P1_3) pin as PWM.

```
uint32_t value = IOCON_MOTOR_SPEED;
value |= (1<<0);
value |= (1<<1);
value &= ~(1<<2);
IOCON_MOTOR_SPEED = value;
```

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

- Reset The PWM Timer Counter and the PWM Prescale Counter on the next positive edge of PCLK.
  `PWM0->TCR |= (1<<1);`

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

MR0 register is used for controlling the PWM Cycle Rate. The Peripheral Clock Frequency for the LPC4088 is set as 60 MHz. Whenever you change the value of MR registers, you should inform the PWM that you changed the MR register value.

- Enable PWM Match 0 Latch.

```
PWM0->LER &= ~(1<<0);
```

- Enable Counter and PWM and Clear Reset on the PWM.

```
PWM0->TCR |= (1<<0);
```

```
PWM0->TCR |= (1<<3);
```

```
PWM0->TCR &= ~(1<<1);
```

Match Register values are changed in PWM_Cycle_Rate and PWM_Write methods too. Therefore:

- Enable PWM Match Register Latch in PWM_Cycle_Rate

```
PWM0->LER |= (1<<0);
```

- Enable PWM Match Register Latch in PWM_Write

```
PWM0->LER |= (1<<2);
```