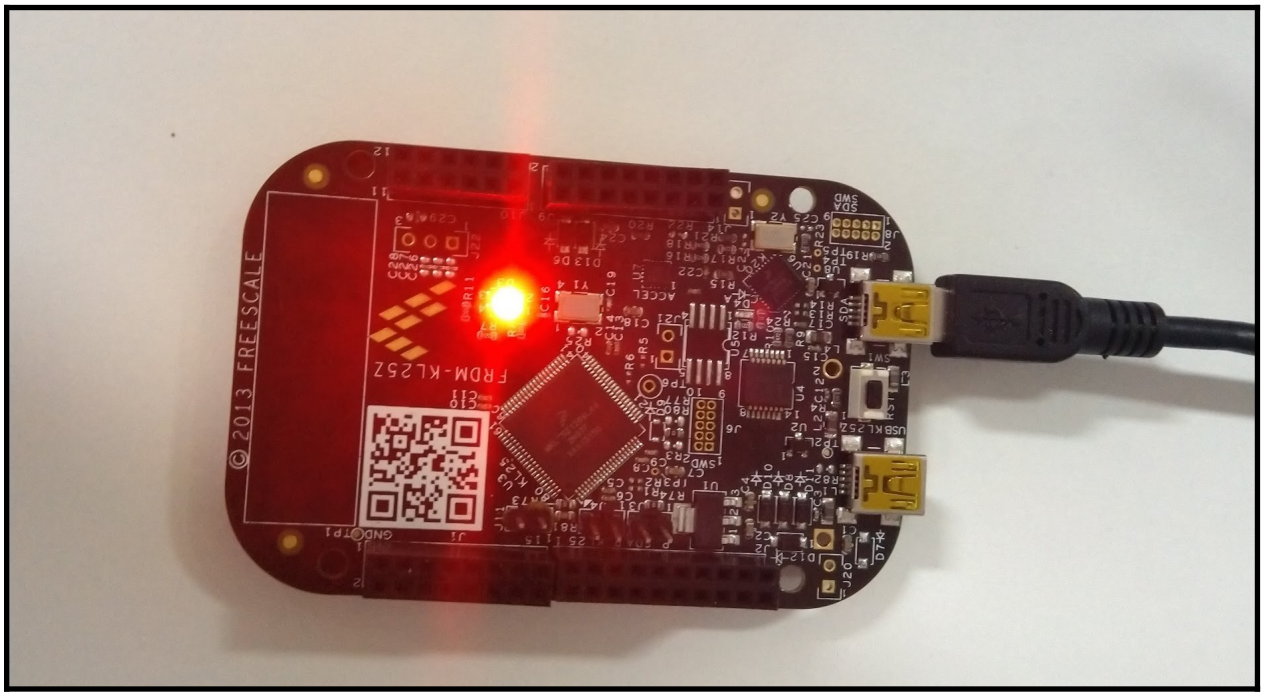
# Lab Report




# Embedded Systems

**Himanshu Rai**

**111601032**

**20th September, 2019**

# Objective

- Using timer interrupts to turn an LED on the FRDM board on and off.
- Using timer interrupts to turn blue and green LEDs on the FRDM board on and off in succession at different frequencies.
- Using UART interrupt to turn LEDs on the FRDM board on and off, where colour is determined by the user input.

# Theory

## Interrupts

Exception is anything that stops the normal execution of a program. Interrupts are special type of exceptions – they are desirable exceptions used to perform different kind of tasks according to their nature. For example a timer interrupt can, at every predetermined interval, signal the CPU which can then perform some required task like process scheduling. An external hardware can interrupt the CPU when it has finished its assigned job or is ready to take a new job.

## Timer Interrupts in FRDM KL25Z

SysTick timer generates and sends interrupt to the processor whenever its *current value register (CVR)* counts down (every processor cycle) from 1 to 0. The value in *reload value register (RVR)* is loaded in the *CVR* after which the countdown begins again. In order to enable the timer, we need to set the appropriate bits in *control and status register (CSR)*. Setting $0^{th}$ bit enables the timer, setting $1^{th}$ bit enables interrupts, and $2^{nd}$ bit selects the clock source (reference clock when 0 and processor clock when 1). We can initialise *CVR as 0*, *RVR* as (processor_frequency / 1000) for an interrupt every milli second. Also, we set the *CSR* as 0x7.

## UART Communication (Universal asynchronous receiver/transmitter)

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs.

We will use UART0 ports on the FRDM board to read user input based on which we can decide the colour with which we can light the LED. The following steps

are required to configure the UART0 port so that we can use it to read the inputs.

- By default, UART modules are disabled to save power. So, we should first enable the clocks to the ports corresponding to UART0 to prevent hardware fault. The *system clocking gate control register 4 (SIM_SCGC4)* gates the clocks to UART ports. We can enable the clock for UART0 by setting the 10th bit in *SIM_SCGC4* control register.

- Next we need to choose from the different available clock sources. *SOPT2* contains the controls for selecting many of the module clock source options on this device. The clock for UART0 can be configured by modifying bit positions 26th and 27th.
    - 00 - Clock disabled
    - 01 - MCGFLLCLK clock or MCGPLLCLK/2 clock
    - 10 - OSCERCLK clock
    - 11 - MCGIRCLK clock

  We have to choose the second option, so we will configure the bits to have 01 at the mentioned bit positions.

- Next we need to set up the *SBR* and *OSR* (over sampling ratio). These two are related by the following equation

  SBR = clock frequence / (OSR * baud rate)

  For us, baud rate = 57600Hz, clock frequency = 22MHz, and we will have *OSR* of 16. So, SBR will be 24.
  We can set *OSR* through *UART0_C4* register using bits 0 to 4. For setting *SBR* we need to modify two registers - *UART0_BDL* which corresponds to *SBR[7:0]* and *UART0_BDH* whose first 5 bits corresponds to *SBR[12:8]* .
- Next we will activate *port A* by setting 9th bit in *SIM_SCGC5* register and then configure *port A1* to be used as UART_Tx by putting 010 at bit positions 10 to 8 in PCR for *port A1*.
- Finally activate the receiver of *UART0* by setting *UART0_C2* register as 4.

We can check whether we have received some input by checking 5th bit in *UART0_S1* register. If this is set it means the receive buffer is full and we can see the input in *UART0_D* register.

**UART Interrupts in FRDM KL25Z**

We can enable interrupts for UART0 by setting the 12$^{th}$ bit in *interrupt set enable register (ISER)* field of *nested vector interrupt controller (NVIC).*

**Lighting up the LEDs**

The following tables give the correspondence between the LEDs and port pins associated with them

| LED | GPIO PIN |
|---|---|
| Red | PortB18 |
| Green | PortB19 |
| Blue | PortD1 |

First we will have to follow the following procedure to configure the ports/pins corresponding to the LEDs, before using them

- By default, GPIO modules are disabled to save power. So, we should first enable the clocks to the ports corresponding to the LEDs to prevent hardware fault. The *system clocking gate control register 5 (SIM_SCGC5)* gates the clocks to GPIO ports. We can enable the clock for these ports by setting the respective bits in *SIM_SCGC5* control registers - which are bit 10 (for port B - red and green LEDs) and bit 12 (for port D - blue LED).
- Next configure the pins for GPIO . For this we must write 001 in the corresponding *program control registers (PCR)* at bit positions 10 to 8.
- Now set the pins as output pins by setting the corresponding pin position in the *port data direction register (PDDR)* of the corresponding ports.

After setting up the pins, we can turn the LEDs on and off by setting the corresponding bits in *port clear output register (PCOR)* and *port toggle output register (PTOR).*

# Procedure

- **Using timer interrupts to turn an LED on the FRDM board on and off**

A function to enable the timer interrupt is implemented as mentioned. A counter is used, which is decremented by interrupt handler function *SysTick_Handler* every time the interrupt occurs. This counter is used by a *delay* function to provide delay of specified miiliseconds. The delay function is used to turn one of the LED on the FRDM board on and off every second using the LED init and on/off function written in the previous labs.

- **Using timer interrupts to turn blue and green LEDs on the FRDM board on and off in succession at different frequencies**

This also can be implemented similar to previous step. But instead of one counter, we can have two counters one for turning each LED on/off.

- **Using UART interrupt to turn LEDs on the FRDM board on and off, where colour is determined by the user input**

For this enable the UART interrupt and the *UART0_IRQHandler* function can be used to light the LEDs on and off depending on the condition.

# Results

The desired results were obtained and shown during the lab.

# Source Code

- **Using timer interrupts to turn an LED on the FRDM board on and off**

```
#include<MKL25Z4.h>

static unsigned volatile int delayCounter = 0;

void green_led_init() {
    SIM->SCGC5 |= 1 << 10;          // will activate the port B

    // set the pin as gpio
    PORTB->PCR[19] &= 0xFFFFF8FF;   // set 8th, 9th and 10th bit of PCR[19] to 0
    PORTB->PCR[19] |= 1 << 8;       // set 8th bit of PCR[19] to 1

    // set the port as output port
    PTB->PDDR |= 1 << 19;           // set the 19th bit of PDDR to 1 for output
}

void green_led_toggle() {
    // toggle the 19th bit of PDOR register
    PTB->PTOR |= 1 << 19;
}

void SysTick_init() {
    SysTick->LOAD = 20971 - 1;      // load the RVR of SysTick
    SysTick->VAL = 0x00;            // clear the current value of SysTick counter
    SysTick->CTRL = 0x7;            // enable the counter and SysTick Exception
                                    //  request (1 at bit positions 0 and 1), also
                                    //  use processor clock (1 at bit position 3)
}

void SysTick_Handler(void) {
    delayCounter++;
}

void delay(int ms) {
    delayCounter = 0;
    while(delayCounter < ms);
}

int main() {
    SystemCoreClockUpdate();
    green_led_init();
    SysTick_init();

    while(1) {
        delay(1000);
        green_led_toggle();
    }
}
```

- **Using timer interrupts to turn blue and green LEDs on the FRDM board on and off in succession at different frequencies**

```
#include<MKL25Z4.h>


static unsigned volatile int delayCounter = 0;
static unsigned volatile int ticker[4] = {0};

void green_led_init() {
    SIM->SCGC5 |= 1 << 10;          // will activate the port B

    // set the pin as gpio
    PORTB->PCR[19] &= 0xFFFFF8FF;   // set 8th, 9th and 10th bit of PCR[19] to 0
    PORTB->PCR[19] |= 1 << 8;       // set 8th bit of PCR[19] to 1

    // set the port as output port
    PTB->PDDR |= 1 << 19;           // set the 19th bit of PDDR to 1 for output
}

void blue_led_init() {
    SIM->SCGC5 |= 1 << 12;          // will activate the port D

    // set the pin as gpio
    PORTD->PCR[1] &= 0xFFFFF8FF;    // set 8th, 9th and 10th bit of PCR[1] to 0
    PORTD->PCR[1] |= 1 << 8;        // set 8th bit of PCR[1] to 1

    // set the port as output port
    PTD->PDDR |= 1 << 1;            // set the 1th bit of PDDR to 1 for output
}


void green_led_toggle() {
    // toggle the 19th bit of PDOR register
    PTB->PTOR |= 1 << 19;
}

void blue_led_toggle() {
    // toggle the 18th bit of PDOR register
    PTD->PTOR |= 1 << 1;
}

void SysTick_init() {
    SysTick->LOAD = 20971 - 1;      // load the RVR of SysTick
    SysTick->VAL = 0x00;            // clear the current value of SysTick counter
    SysTick->CTRL = 0x7;            // enable the counter and SysTick Exception
                                    // request (1 at bit positions 0 and 1), also
                                    // use processor clock (1 at bit position 3)
}

void SysTick_Handler(void) {
    for(int i = 0; i < 4; i++) {
        ticker[i]++;
    }
}


int main() {
    SystemCoreClockUpdate();
```

```
    green_led_init();
    blue_led_init();
    SysTick_init();

    while(1) {
        if(ticker[0] > 500) {
            ticker[0] = 0;
            green_led_toggle();
        }
        if(ticker[1] > 1000) {
            ticker[1] = 0;
            blue_led_toggle();
        }
    }
}
```

- **Using UART interrupt to turn LEDs on the FRDM board on and off, where colour is determined by the user input**

```
#include<MKL25Z4.h>


static unsigned volatile int delayCounter = 0;

void UART0_init() {
    // enabling UART0 by setting the corresponding bit in SCGC4 register
    SIM->SCGC4 |= 1 << 10;

    // enable the clock at UART0 from FLL or PLL/2 by putting 01 at bit
    // positions 27 and 26 in SOPT2 registers
    SIM->SOPT2 |= 1 << 26;
    SIM->SOPT2 &= ~(1 << 27);

    /*
    SBR = (clock frequency = 22000000) / (OSR * (baud rate = 57600))
    clock frequency and baud rate are fixed so we need only OSR or SBR
    SBR is of 13 bits and we set its value using the following registers -
    SBR[7:0] = BDL[7:0] and SBR[12:8] = BDH[4:0]
    */

    // disable all the operation modes in UART0
    UART0->C2 = 0x00;

    // SBR setting for baud rate of 57600 (and OSR = 16)
    UART0->BDH = 0x00;
    UART0->BDL = 0x18;
    UART0->C4 = 0x0F; // for an OSR of 16
    UART0->C1 = 0x00; // no parity

    // activate the receiver of UART0 (with interrupt enabled)
    UART0->C2 = 0x36;

    // activating port A
    SIM->SCGC5 |= 1 << 9;
    // configuring PTA1 as UART_Tx by putting 010 at bit positions 10 to 8 in PCR
    // for PORT A1
    /*
    Note that we can also do this using alternative 4 for PORTE21
```

```
    But here we are using alternative 1 for PORTA1 because it is connected to USB
    port whereas PORTE20 is not
    */

    PORTA->PCR[1] &= ~(7 << 8);
    PORTA->PCR[1] |= 1 << 9;


    NVIC->ISER[0] = 0x00001000;
}

void led_red_init() {
    SIM->SCGC5 |= 1 << 10; // will activate the port B

    // set the pin as gpio
    PORTB->PCR[18] &= 0xFFFFF8FF; // set 8th, 9th and 10th bit of PCR[18] to 0
    PORTB->PCR[18] |= 1 << 8;     // set 8th bit of PCR[18] to 1

    // set the port as output port
    PTB->PDDR |= 1 << 18;         // set the 18th bit of PDDR to 1 for output
}

void led_red_on() {
    // clear the 18th bit of PDOR register
    PTB->PCOR |= 1 << 18;         // set the 18th bit of PCOR to 1
}

void led_red_off() {
    // toggle the 18th bit of PDOR register
    PTB->PTOR |= 1 << 18;
}

void led_green_init() {
    SIM->SCGC5 |= 1 << 10;        // will activate the port B

    // set the pin as gpio
    PORTB->PCR[19] &= 0xFFFFF8FF; // set 8th, 9th and 10th bit of PCR[19] to 0
    PORTB->PCR[19] |= 1 << 8;     // set 8th bit of PCR[19] to 1

    // set the port as output port
    PTB->PDDR |= 1 << 19;         // set the 19th bit of PDDR to 1 for output
}

void led_green_on() {
    // clear the 18th bit of PDOR register
    PTB->PCOR |= 1 << 19;         // set the 19th bit of PCOR to 1
}

void led_green_off() {
    // toggle the 18th bit of PDOR register
    PTB->PTOR |= 1 << 19;
}

void led_blue_init() {
    SIM->SCGC5 |= 1 << 12;        // will activate the port D

    // set the pin as gpio
    PORTD->PCR[1] &= 0xFFFFF8FF;  // set 8th, 9th and 10th bit of PCR[1] to 0
    PORTD->PCR[1] |= 1 << 8;      // set 8th bit of PCR[1] to 1
```

```
    // set the port as output port
    PTD->PDDR |= 1 << 1;         // set the 1th bit of PDDR to 1 for output
}

void led_blue_on() {
    // clear the 18th bit of PDOR register
    PTD->PCOR |= 1 << 1;         // set the 1th bit of PCOR to 1
}

void led_blue_off() {
    // toggle the 18th bit of PDOR register
    PTD->PTOR |= 1 << 1;
}

void delay(int ms) {
    while(ms--);
}

void UART0_IRQHandler() {
    char ch = UART0->D;
    if(ch == 'A') led_red_on(); delay(1000000); led_red_off();
    else if(ch == 'B') led_green_on(); delay(1000000); led_green_off();
    else if(ch == 'C') led_blue_on(); delay(1000000); led_blue_off();
}

int main() {
    SystemCoreClockUpdate();      // updating the clock from PLL
    UART0_init();

    led_red_init(); led_red_off();
    led_green_init(); led_green_off();
    led_blue_init(); led_blue_off();

    while(1);
}
```