Lab 2 Report

By: Kaushik Ravikumar (kr437) , Eldor Bekpulatov (eb654)

Tutorial

Following the tutorial to configure the LED's was somewhat straightforward. Once we got the red led to turn on and off, it was very easy for us to modulate the red LED functions under helper functions, such as `red_led_init()`, `red_led_on()`, `red_led_off()`, and `red_led_toggle()`. Once we had the red LED working, getting the blue and green ones was as easy as copying&pasting the same code and changing some of the ports as specified in the lab diagram.

```
void red_led_init(){
    SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK; //Enable the clock to port B
    PORTB->PCR[22] = PORT_PCR_MUX(001); //Set up PTB22 as GPIO
    PTB->PDDR = 1 << 22; // enable Pin PTB22 as Output
    PTB->PSOR = 1 << 22; // turn off the LED
}
void red_led_on(){
    PTB->PCOR |= 1 << 22;
}
void red_led_toggle(){
    PTB->PTOR = 1 << 22;
}
void red_led_off(){
    PTB->PSOR = 1 << 22;
}
```

Part 1

To implement Periodic Interval Timer timer, we first had to enable the clock to PIT module.

```
SIM->SCGC6 |= SIM_SCGC6_PIT_MASK; // enable clock to PIT
```

Once that is done, we had to enable the PIT module by calling:

```
PIT->MCR &= ~PIT_MCR_MDIS_MASK;  // enable PIT module
```

Before we start the timer module, we first loaded the channel zero with a number that results in about 24 million cycles before raising an interrupt. This was determined from the clock frequency and our goal of achieving 1 second timer.

```
PIT->CHANNEL[0].LDVAL = 0x16e3600; // 24million
```

Lastly, we simply had to start the timer.

```
PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK; // Start timer
```

Now that we got our timer to start, we configured our LED's to respond to our polling function which simply checked the `PIT[0]'s TFLG` flag forever, and whenever the flag was set high, it would toggle the LED and reset the flag. This process would continue forever.

```
int main (void) {
    red_led_init(); // initialize the red led
    SIM->SCGC6 |= SIM_SCGC6_PIT_MASK; // Enable clock to PIT module
    PIT->CHANNEL[0].LDVAL = 0x16e3600; // Set load value of zeroth PIT
    PIT->MCR &= ~PIT_MCR_MDIS_MASK;  // Set MCR
    PIT->CHANNEL[0].TCTRL |= PIT_TCTRL_TEN_MASK;    /* Start timer*/
    while(1){
        if (PIT->CHANNEL[0].TFLG) {
            red_led_toggle();
            PIT->CHANNEL[0].TFLG &= PIT_TFLG_TIF_MASK; }}}
```

Part 2

We began by simply invoking the helper functions we defined earlier in part 1, `blue_led_init() and green_led_init()`, in order to use the led lights on the board for our program. We then enabled interrupts using NVIC and define our PIT Interrupt Handler. Then we must do some initial setup functions to enable the clock to the PIT module, so they are working in sync. We must also enable the PIT MCR module, as this is needed for interrupts.

After completing setup, we moved on by loading a value into channel 0. We loaded approximately 20 million using the following line. `PIT->CHANNEL[0].LDVAL = 0x13e3600` After careful visual observation, we noticed that this hex value corresponds to an interrupt every second. The main loop will be a blue light that toggles every second. In order to do this, we simply had an infinite while loop which had a for loop initialized at `0x500000`, and decremented to 0, taking approximately a second. Then we called toggle `blue_led_toggle()`.

```
while(1)
{
    for(int i=0x500000;i>0;i--){}; // approximately 1.0 second delay
    blue_led_toggle();
}
```

Next, we defined the void function that serves kind of like a callback function that is triggered when the interrupt flag is raised. In this case, we call `green_led_toggle()`, perform a delay of 0.1 seconds, which we found to be the hex value, `0x90000`, to be perfect for. Thus after similarly using a for loop with the counter initialized to this value, we created the delay and then called `green_led_toggle()` again.Then finally, we had to make sure the flag was removed, so the program is not stuck in the interrupt handler, and we did this with the line `PIT->CHANNEL[0].TFLG =0x1`. After finishing the interrupt handler, our program worked as desired.

Debugging

Some of the bugs we faced were the result of our flawed approach to part 2. In part 2, we were using two interrupt handlers and this was not right as we only needed one to know when to trigger the green led light. At times we forgot to enable the MCR module. Additionally, we faced issues with the timing of our delays. We were able to fix this, by performing calculations based on clock frequency and making adjustments based on visual observations.

Work Distribution

In order to collaborate, we met up once during February break to work together and bounce ideas off of each other. We practiced peer programming, where we took turns writing code and coming up with general ideas. Eldor worked on part 1 and figuring out how to get the LED lights working. Kaushik and Eldor worked together on part 2 and debugging it, as there were many issues initially in getting the interrupts to work and deciding on the right numbers to use to get the right length delays. Both worked together to complete the lab report.