ECE 3140/CS3420 Lab 2

LEDs, Polling, and Interrupts

Goal

The purpose of this lab is to give you experience with I/O and interrupts on the FRDM-K64F microcontroller. To successfully complete this lab, you will need to understand the following:

- Interrupts
- Polling
- Memory-mapped I/O

For each part of the lab, we recommend that you create a separate project in Keil. Before you begin, you should work through the included "Creating a C file tutorial" file to familiarize yourself with Keil.

Part 1: Lighting up the Board

To start, let's look at how the RGB LED is implemented in the board. Navigate here to the FRDM-K64F User's Guide, which contains overviews of the various modules. Search the User's Guide for information relevant to the LED, including this circuit schematic shown below:

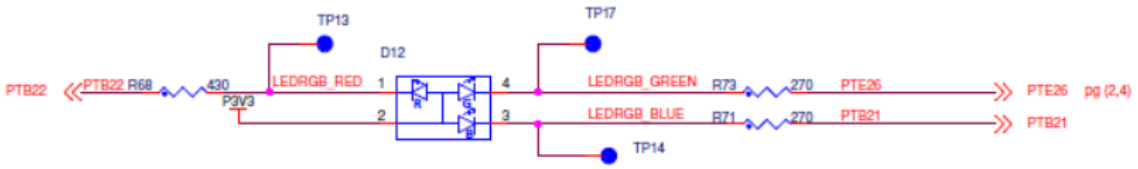


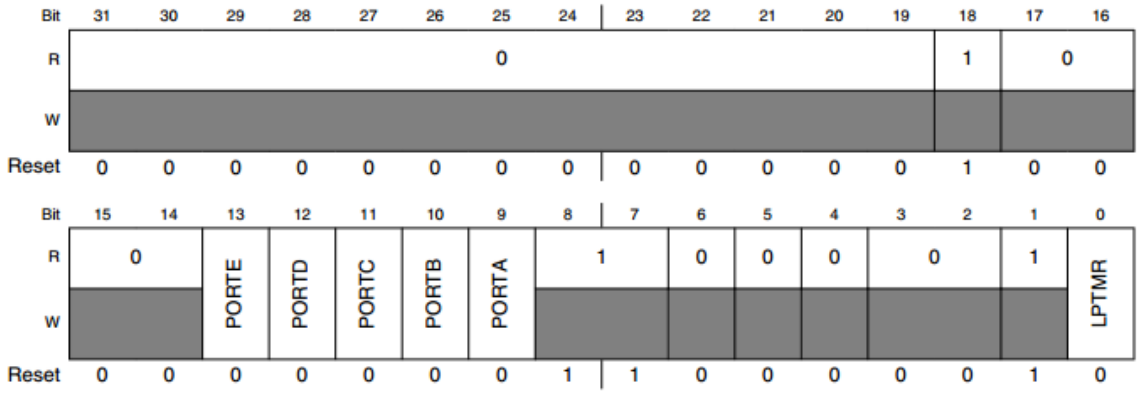
Figure 14. Tricolor LED

The schematic from that document shows us that the red LED is attached to PTB22. PTB22 is a single pin from one of the ports (PTA, PTB, PTC, PTD, PTE). Each port has 32 pins. This means that to turn the red LED on or off, you need to change the 22nd pin of port B. Before we can do this, however, we have to gate the clock. By default, the gates to all the ports are switched off to save energy. In order to use port B (to enable the red LED), we have to turn them on.

To do so, we have to set some bits inside of the SIM (System Integration Module). If you look at page 314 in the reference manual for the K64F (link here), you will see the diagram below with explanations of each of the pins. As you can see, some of the bits of the SIM » SCGC5 register control power to ports A through E. More specifically, bit 9 is the Port A clock gate control, bit 10 is the Port B clock gate control, and so on. Since, at least for now, we care about the red LED, we need to set the 10th bit to 1. We know to use 1, and not 0, to indicate a clock enable from the table below the diagram on page 314.

12.2.12 System Clock Gating Control Register 5 (SIM_SCGC5)

Address: 4004_7000h base + 1038h offset = 4004_8038h



The following line of C code will accomplish this.

```
SIM->SCGC5 |= (1 << 10); //Enable the clock to port B
```

While this code works, "(1 << 10)" isn't that easy to understand. Luckily, there are certain masks defined to make our lives easier in MK64F12.h (the header file!). If you open up MKL25Z4.h (it is under Device » system_MK64F12.c (Startup) » MK64F12.h in the Project window on the left), you can see a ton of macros which may be helpful for the rest of the course. Specifically, on line 11424:

```
#define SIM_SCGC5_PORTB_MASK         0x400u;
```

This means that instead of simply setting a 1 to the 10th bit, we can use the following line:

```
SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK; //Enable the clock to port B
```

There are 2 more things we must do before we can toggle the LED. The first of these is to set PTB22 (the pin we want to use to toggle the LED) as a GPIO pin. In the K64F, some pins can have up to eight different uses. To choose the function we want, we have to set the appropriate control signals to an internal mux. This can be done with the PCR (Pin Control Register) detailed on page 282 of the manual. Again, the table below the chart is very helpful. Towards the bottom of page 283, we see the MUX bits indicate which use the pin will have. We have to set the MUX bits (10-8) to 001 to enable pin 22 as GPIO. This can be done as follows:

```
PORTB->PCR[22] = (1 << 8); //Set up PTB18 as GPIO
```

Or, using macros:

```
PORTB->PCR[22] = PORT_PCR_MUX(001); //Set up PTB18 as GPIO
```

Finally, we have to enable PTB22 as an output. This can be done through the PDDR (Port Data Direction) explained on the very bottom of page 1763 of the same document. This can be implemented by setting the appropriate bit of the `PTB->PDDR` register. You will have to figure out the C code for yourself.

Once this is done, we can toggle the red LED! Look through the manual around page 1761 for the `PSOR` and `PCOR` registers which will be used to actually turn the LED on or off. Keep in mind that the way the LED is set up, a logic level of 0 will cause the LED to turn on. You will probably have to assign values to `PTB->PSOR` and `PTB->PCOR`.

Try it out! You should be able to write a few quick C programs that make use of the LED. It may be helpful to define a few functions, for instance, `LEDRed_On()` and `LEDRed_Off()`. After the red LED works, try implementing the blue LED (Note: the green LED comes from a different port, so it may be harder to implement!).

Part 2: Polling the Periodic Interval Timer

The goal of this part of the lab is to implement a polling-based approach to see if the timer has finished. For this lab, we will be using a PIT (periodic interval timer). The K64F has four periodic interval timers, though the code is set up for you to use the first one: PIT[0]. For reference, look near page 1086 in the reference manual. You will most likely need to access the `TCTRL`, `TFLG`, `LDVAL`, and `MCR` registers. For instance, in order to set the load value of the timer (what the timer counts down from), we use:

```
PIT->CHANNEL[0].LDVAL = 0x30000; //Set the load value of the zeroth PIT
```

In order to implement the PIT, you must first enable the clock to the PIT module. This is done through one line of code (this is actually done through the SIM, the same way we enabled a clock for the LED).

```
SIM->SCGC6 = SIM_SCGC6_PIT_MASK; //Enable the clock to the PIT module
```

By default, the PIT will count down from the specified value, and will change the value of the TFLG when the counter reaches zero. To implement polling, you should periodically check this value to see if you have to reset the timer. The diagrams on page 1090 may help explain how the timers work on a functional level.

The PIT is connected to the peripheral bus clock. The startup files set an appropriate frequency for this clock. Use this information to choose appropriate load values for the PIT.

TASK 1:

Using lab2_part2.c as a starting point for your code, write a C program that polls the PIT's TFLG register to see when the timer expires. When it does, your program should toggle the red LED. Use an appropriate load value so that the LED changes state approximately once every second.

Part 3: Implementing Interrupts

In order to implement interrupts, you must change some settings in the PIT. The timers can be configured to generate an interrupt when they reach zero, which triggers the PIT_IRQHandler. The code that controls this (and tells you that PIT_IRQHandler is the function you have to define) is in one of the startup files that is included with your C code. Within the interrupt, make sure to clear the interrupt flag, or you'll end up always in the interrupt code.

TASK 2:

Using lab2_part3.c as a starting point, write a new C program that toggles the red LED about once a second using a `for` loop to create a delay. When a PIT interrupt occurs – again, once every second – flash the green LED for approximately a tenth of a second. During and after the interrupt, the red LED should continue to blink.

Deliverables

1. **lab2_part2.c**, containing the solution for task 1. We encourage you to use meaningful variable and function names, comments, etc. to enhance code comprehensibility.
2. **lab2_part3.c**, containing the solution for task 2. We encourage you to use meaningful variable and function names, comments, etc. to enhance code comprehensibility.
3. **lab2_writeup.pdf**, containing a description of your solution along with the rationale for any design choices you make. We require that you describe in detail, in a separate section titled "Work Distribution", the following:
    a. How did you carry out the project? In your own words, identify the major components that constituted the project and how you conducted the a) design, b) coding, c) code review, d) testing, and e) documenting/writing. If you followed well-documented techniques (e.g., peer programming), this is the place to describe it.
    b. How did you collaborate? In your own words, explain how you divided the work, how you communicated with each other, and whether/how everyone on the team had an opportunity to play an active role in all the major tasks. If you used any sort of tools to facilitate collaboration, describe them here.