# EENG461, LAB2 report

Mark Clark and Jeremy Munson

**Objective** In this lab we begin exploring interrupts and the concept of task scheduling. We are to have two LEDS are to blink in a pattern, and another LED toggle with a button press, using debouncing.

**Introduction** Task scheduling is a necessary part of real-time embedded systems that allows for many disparate activities to be completed in the appropriate order and at the appropriate time. The Prof indicated that his requirements are "just get interrupts working", so we did not implement a general-purpose task scheduler, like in the example code. We did, however, use interrupts to service the different tasks asynchronously.

**Narrative** The first step to any embedded system project is to understand the application and create the coding-outline, finite state machines and/or pseudocode. The coding-outline, finite state machines, and pseudocode provide a guideline in starting the development and coding process.

The code for this project starts with the initialization step, which includes configuring the GPIO ports, the timer module, and the accompanying interrupt masks and NVIC enables. The code for this project utilizes direct register manipulations using logical operations, such as |= and &=. For instance,

```
SYSCTL_RCGCGPIO_R |= (1 << 5);
```

Sets bit 5 of the RCGCGPIO register to a 1, enabling GPIO Port F.

The main function consists of the finite state machine for the main routine, which starts with the blue LED being turned on, waiting for one second, turning on the red LED, waiting for two seconds, and then turning both the red and blue LEDs off for two seconds. The main routine has a 5-second period and loops infinitely. The main loop requires a timer to keep time so the intervals are accurate. To achieve the proper timing, a one-second, periodic timer was configured with an attached timeout interrupt that triggered an interrupt service routine to increment a second counter. Whenever the max period of five seconds is reached, the second counter resets to zero. In order to set the correct timer period, the number of cycles/ticks required was calculated using the formula:

```
$$cycles = (desiredSeconds)\cdot (clkFrequency)$$
```

The code for changing the LEDs is a simple switch statement that chooses the appropriate LED configuration based on what second of the loop it is currently on. The LEDs are toggled on and off using the register at the GPIO_PORTF_DATA_BITS_R address. The simplest way to alter the value of a bit was found to be using the syntax:

```
GPIO_PORTF_DATA_BITS_R[value] = value;
// value is the hex value corresposding to the bit/s you want to set. (i.e. bit #1
↪  is 0x01 or (1 << 1))
```

The syntax also supports bitwise or and bitwise and operations in the value location to mix known values together as follows:

```
GPIO_PORTF_DATA_BITS_R[value1 | value2] = value1 | value2;
```

In addition to the main routine, the project called for a switch to toggle the green LED without preempting or changing the timing on the main loop. The switch was configured with an interrupt that triggered a subroutine that reads the bit at the GPIO_PORTF_DATA_BITS_R address and inverts the state of the third bit, which corresponds to the green LED.

The debouncing is implemented a bit different than we were shown to do; Instead of starting a timer at the rising edge, then reading the switch level after 10ms, we use the same timer for both tasks.

When the first edge change occurs, a button push down is registered immediately with no delay. If more edge changes are detected within 10ms then they are ignored. In order to differentiate between bounces that may occur on button release, the "state" of the button is tracked as either pushed or released. A valid release must be detected before a push can be registered, and vice-versa. In tracking the time passed since the button press, several potential race conditions are detected/corrected by disabling interrupts in a critical section and checking if the timer rolled over during the read. This method can debounce any number of switches with only a single hardware timer acting as a system time clock. Because handling these button presses is not time-critical, the edge change interrupt is set to the lowest priority level.

**Concluding Remarks** The lab was completed with no major issues, and the goals met. We successfully used timers and interrupts to perform actions at specific times, as well as respond to events asynchronously.

**Appendix Code Outline and FSM** We prepared a FSM to describe the behaviour we were planning at the start. See Figure 1.

```
#pragma once
#ifndef EENG461_LAB_2_MAIN_H
#define EENG461_LAB_2_MAIN_H

int main (void);
void Disable_Interrupts(void);
void Enable_Interrupts(void);

#endif //EENG461_LAB_2_MAIN_H
```

**Appendix - main.h**

```
#include "main.h"
#include "setup.h"
#include "timers.h"
#include <stdint.h>
#include <common/tm4c123gh6pm.h>

volatile int sec_count;

int main (void) {

    setup();
```

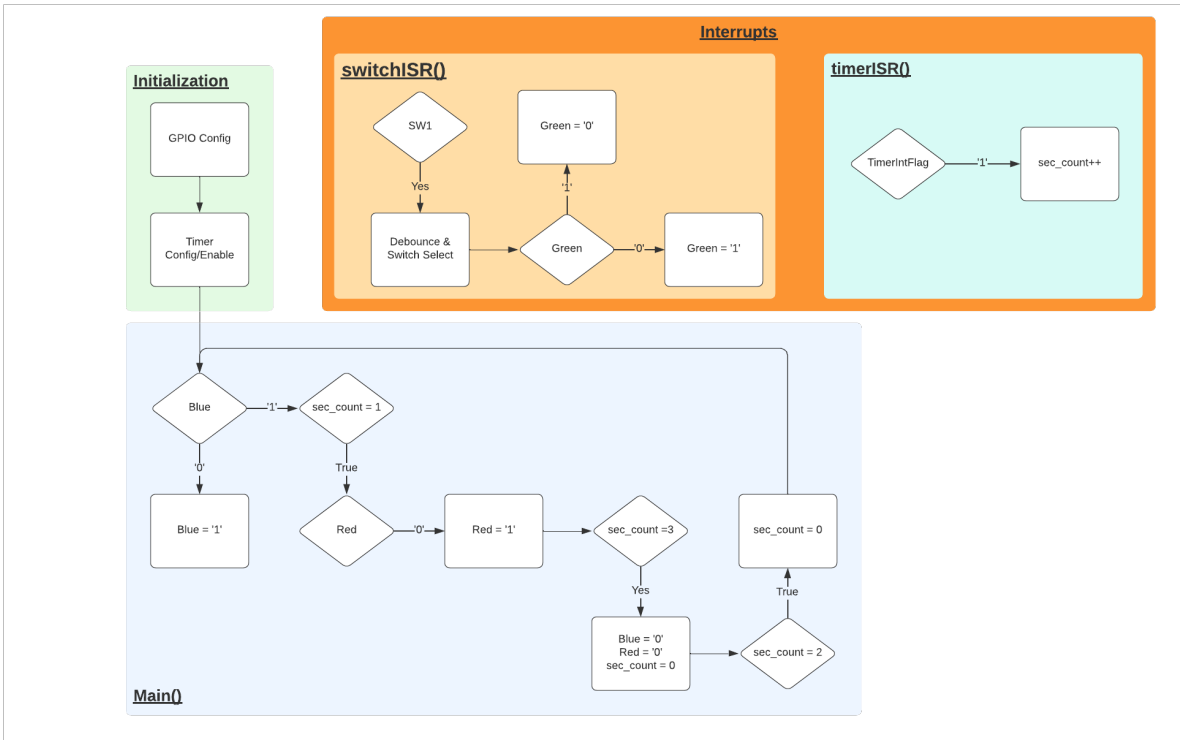


Figure 1: Coding Outline

```
    configureTimer();

    Enable_Interrupts(); //Enable Global Interrupts

    while (1) {

        switch (sec_count) {
            case 0:
                /*
                 * Time = 0 starts with blue led turning on
                 */
                GPIO_PORTF_DATA_BITS_R[BLUE_LED] = BLUE_LED;
                break;
            case 1:
                /*
                 * After 1 second of blue LED being on, turn on the red LED (leaving
↪   blue LED on)
                 */
                GPIO_PORTF_DATA_BITS_R[RED_LED] = RED_LED;
                break;
            case 3:
                /*
                 * After 2 seconds of red and blue LEDs being on, turn both of them
↪   off
```

```
                 */
                GPIO_PORTF_DATA_BITS_R[RED_LED | BLUE_LED] =  0;
                break;
        }
    }

    return (0);
}

/*
 * Taken from Lab Assignment
 */
void Disable_Interrupts(void) {
    __asm ("  CPSID    I\n");
}

void Enable_Interrupts(void) {
    __asm ("  CPSIE    I\n");
}
```

**Appendix - main.c**

```
#pragma once
#define GREEN_LED (1 << 3)
#define BLUE_LED (1 << 2)
#define RED_LED (1 << 1)
#define RGB_PINS GREEN_LED | BLUE_LED | RED_LED
#define SW1_PIN (1 << 4)

void setup(void);
```

**Appendix - setup.h**

```
#include "common/tm4c123gh6pm.h"
#include "setup.h"
#include <stdint.h>

void setup(void) {
    // Enable GPIO clock
    SYSCTL_RCGCGPIO_R |= (1 << 5);
    while(!(SYSCTL_PRGPIO_R & SYSCTL_PRGPIO_R5)) {};

    // Configure pins
    GPIO_PORTF_DEN_R |= RGB_PINS;
    GPIO_PORTF_DR8R_R |= RGB_PINS;

    // Set initial values
    GPIO_PORTF_DATA_R &= ~RGB_PINS; //All off to start
```

```
    // Set pin directions
    GPIO_PORTF_DIR_R |= RGB_PINS;

    //SW1 pullup
    GPIO_PORTF_PUR_R |= SW1_PIN;
    GPIO_PORTF_DEN_R |= SW1_PIN;

    //Enable interrupts on value of buttons
    GPIO_PORTF_IS_R &= ~SW1_PIN; //Edge triggered
    GPIO_PORTF_IBE_R |= SW1_PIN; //Both Edges
    GPIO_PORTF_IM_R |= SW1_PIN;  //Unmask the pin

    NVIC_PRI7_R = (NVIC_PRI7_R & ~NVIC_PRI7_INT30_M) | (0x7 << NVIC_PRI7_INT30_S);
↪   //Set the PORTF interrupt to the lowest priority.

    NVIC_EN0_R |= (1 << 30);     // Enable Port F interrupts in nvic
}
```

**Appendix - setup.c**

```
#pragma once
void PORTF_int_handler(void);
```

**Appendix - sw1_int.h**

```


#include "sw1_int.h"

#include <stdbool.h>
#include <stdint.h>

#include "common/tm4c123gh6pm.h"
#include "setup.h"
#include "timers.h"

#define MIN_CLOCKS_DEBOUNCE (int32_t)((int32_t)CYCLES_PER_SEC/100)

#define TIMER_ISR_IS_PENDING (TIMER0_MIS_R & TIMER_ICR_TATOCINT)

static void sw1_debounce(void);

void PORTF_int_handler(void){
    // ASSUMPTION: ONLY SW1 ON PORTF is toggled.
    // If more pins were being toggled, then we would need to dispatch a different
    ↪  handeler for each.
    // Such code might look like this.

    if(GPIO_PORTF_MIS_R & SW1_PIN) {
        GPIO_PORTF_IM_R &= ~SW1_PIN; //Disable interrupt
```

```
        GPIO_PORTF_ICR_R |= SW1_PIN; //Clear interrupt
        sw1_debounce();
        GPIO_PORTF_IM_R |= SW1_PIN; //Re-enable interrupt
    }
}

//Performs debounce by disallowing further input after the first edge transition
↪  until MIN_CLOCKS_DEBOUNCE has passed.
static void sw1_debounce(void){
    static int32_t uptime_last = 0;
    static int32_t cycles_last = 0;
    static enum { RELEASED, PRESSED } button_state = RELEASED;

    const enum { FALLING, RISING } edge_type = (GPIO_PORTF_DATA_BITS_R[SW1_PIN] ==
    ↪  SW1_PIN);

    int32_t uptime_now;
    int32_t cycles_now;

    // Early exit for don't-care state combinations
    if( ((button_state == PRESSED)  && (edge_type == FALLING )) ||
        ((button_state == RELEASED) && (edge_type == RISING)) ) {
        return;
    }

    //Critical Section: read of the current time
    __asm("CPSID I\n"); //Disable interrupt handling
    do {
        if(TIMER_ISR_IS_PENDING) timerISR();

        uptime_now = uptime_seconds;
        cycles_now = CYCLES_PER_SEC-TIMER0_TAR_R;

    // If the counter overflowed during this code block, then our reads of uptime
    ↪  and cycles are invalid. Re-do them.
    } while (TIMER_ISR_IS_PENDING);
    __asm("CPSIE I\n"); //Re-enable interrupt handeling

    int32_t seconds_passed = (uptime_now - uptime_last);
    if(seconds_passed > 2) seconds_passed = 2; // Prevent overflow
    int32_t cycles_passed  = (cycles_now - cycles_last) +
    ↪  seconds_passed*CYCLES_PER_SEC;

    if(cycles_passed > MIN_CLOCKS_DEBOUNCE) {
        uptime_last = uptime_now;
        cycles_last = cycles_now;

        switch(button_state) {
            case PRESSED:
                button_state = RELEASED;
                break;
```

```
            case RELEASED:
                button_state = PRESSED;
                GPIO_PORTF_DATA_BITS_R[GREEN_LED] ^= GREEN_LED;
        }
    }
}
```

**Appendix - sw1_int.c**

```
#include "stdint.h"

#ifndef EENG461_LAB_2_TIMERS_H
#define EENG461_LAB_2_TIMERS_H

extern volatile int sec_count;
extern volatile int32_t uptime_seconds;

#define CYCLES_PER_SEC 16000000

void configureTimer (void);
void timerISR (void);

#endif //EENG461_LAB_2_TIMERS_H
```

**Appendix - timers.h**

```
#include <stdint.h>
#include "timers.h"
#include "main.h"
#include "setup.h"
#include "common/tm4c123gh6pm.h"

volatile int32_t uptime_seconds;

void configureTimer (void) {

    SYSCTL_RCGCTIMER_R |= SYSCTL_RCGCTIMER_R0; //Enable Run Mode Clock Gating
↪   Control for Timer 0

    while (!(SYSCTL_PRTIMER_R & SYSCTL_RCGCTIMER_R0)) {}

    TIMER0_CTL_R &= ~TIMER_CTL_TAEN; //Disable Timer
    TIMER0_CFG_R = TIMER_CFG_32_BIT_TIMER;
    TIMER0_TAMR_R |= TIMER_TAMR_TAMR_PERIOD; //Set Timer to count down periodically
    TIMER0_TAILR_R = CYCLES_PER_SEC - 1;
    TIMER0_TAPR_R = 0;
    TIMER0_ICR_R |= TIMER_ICR_TATOCINT; //Clear Interrupt
    TIMER0_IMR_R |= TIMER_IMR_TATOIM; //Enable Interrupt as Timeout
```

```
    NVIC_EN0_R = 1 << (INT_TIMER0A - 16);
    TIMER0_CTL_R |= TIMER_CTL_TAEN; //Enable Timer

}

void timerISR (void) {

    TIMER0_IMR_R &= ~TIMER_IMR_TATOIM; //Disable Interrupt
    TIMER0_ICR_R |= TIMER_ICR_TATOCINT; //Clear Interrupt

    if (sec_count < 4) {
        sec_count++; //Increment second counter
    } else {
        sec_count = 0;
    }
    uptime_seconds++;

    TIMER0_IMR_R |= TIMER_IMR_TATOIM; //Enable Interrupt
}
```

**Appendix - timers.h**

```
CC      = arm-none-eabi-gcc
LD      = arm-none-eabi-ld
AR      = arm-none-eabi-ar
AS      = arm-none-eabi-as
OBJSIZE = arm-none-eabi-size
OBJDUMP = arm-none-eabi-objdump
OBJCOPY = arm-none-eabi-objcopy
CPU     = -mcpu=cortex-m4
FPU     = -mfpu=fpv4-sp-d16 -mfloat-abi=softfp

AFLAGS  = -mthumb ${CPU} ${FPU} -MD

CFLAGS  = -mthumb ${CPU} ${FPU} -Og -ffunction-sections -fno-builtin -fdata-sections
↪  -MD -std=c17 -Wall -Wextra -pedantic -c -Dgcc -g -I driver -I src -I sys

CFLAGS  += -DPART_TM4C123GH6PM -DTARGET_IS_BLIZZARD_RA1

LDFLAGS=--gc-sections
#-L /usr/lib/arm-none-eabi/newlib -lc

AFLAGS+=${patsubst %,-I%,${subst :, ,${IPATH}}}
CFLAGS+=${patsubst %,-I%,${subst :, ,${IPATH}}}

export CC LD AR AS OBJCOPY
export CFLAGS AFLAGS LDFLAGS

BUILD_DIR  := objs
```

```
SRCS       := $(wildcard src/*.c) $(wildcard sys/*.c)
HEADERS    := $(wildcard src/*.h) $(wildcard src/*.h) Makefile
OBJECTS    := $(addprefix $(BUILD_DIR)/,$(subst .c,.o, $(SRCS)))
TARGET     := lab2binary

all: credir $(BUILD_DIR)/$(TARGET).axf

# Compile
$(BUILD_DIR)/%.o: %.c $(HEADERS)
    $(CC) $(CFLAGS) -o $(@) -MD -MF $(addprefix $(BUILD_DIR)/,$(subst .c,.d,$<))
↪   $(firstword $^)

# Linking
$(BUILD_DIR)/$(TARGET).axf: $(OBJECTS)
    $(LD) $(OBJECTS) -T sys/tm4c123.ld --entry ResetISR $(LDFLAGS) -o $(@)
↪   -Map=$(BUILD_DIR)/$(TARGET).map
    $(OBJCOPY) -O binary ${@} ${@:.axf=.bin}
    $(OBJDUMP) -Sd -W $(@) > ${BUILD_DIR}/$(TARGET).lss
    $(OBJSIZE) ${@}

# Load binary to device
flash:
    sudo lm4flash $(BUILD_DIR)/$(TARGET).bin

clean:
    rm -rf $(BUILD_DIR)/*
    rm lab2_report.pdf

credir:
    mkdir -p $(BUILD_DIR)
    mkdir -p $(BUILD_DIR)/src
    mkdir -p $(BUILD_DIR)/sys

app_info: $(BUILD_DIR)/${TARGET}.axf
    arm-none-eabi-readelf -a $(^)

%.pdf: docs/src/%.md Makefile
    pandoc $< -o $@ --highlight-style tango --pdf-engine=xelatex

report: lab2_report.pdf

.PHONY: all clean flash compiling app_info report
```

**Appendix - Makefile**

```
#!/bin/bash
sudo echo ""
sudo openocd -f ti_ek-tm4c123gxl.cfg &> /dev/null &
openocd_pid=$!
```

```
gdb-multiarch -ex "file objs/lab2binary.axf" -ex "target extended-remote
↪  localhost:3333" -ex "b main" -ex "layout split" -ex "mon reset halt" -ex "ni"
↪  -tui

kill $openocd_pid
```

**Appendix - debug.sh**

```

source [find interface/ti-icdi.cfg]

transport select hla_jtag

set WORKAREASIZE 0x8000
set CHIPNAME tm4c123gh6pm
source [find target/stellaris.cfg]
```

**Appendix - debug.sh**