# EE474 Introduction To Embedded Systems

Professor Dr Rania Hussein

Summer 2018

Student Jeff Josephsen

# Lab 2: General-Purpose Timers and Interrupts

Date 7-6-2018

Abstract:

In this lab the student gained familiarity with general purpose timers and software interrupts. Timers and interrupts were used to light LEDs and signal to systems when a button had been pressed. Appropriate registers were initialized and updated by way of pointers. A vector table, named cstartup\_M.c, was used to declare, define, and place interrupts in the correct bit location of the interrupt register. Interrupt Service Routines, or ISRs, were written and named Timer\_Handler or GPIO\_PORT\_X\_Handler. These ISR Handlers would then interrupt regular processes flow (like my children often do) to require the system to complete a divergent set of code. In the last part of the Lab it was found that the timer handler needed to be disabled and enabled via the timer clock enable bit for completion of other appropriate processes.

Introduction:

Specific documentation was provided to the student. This documentation included the following:

1. The EK-TM4C123 Launchpad (<http://www.ti.com/tool/EK-TM4C123GXL>)
2. TM4C123 data sheet ([https://canvas.uw.edu/courses/1205180/files/folder/Ek-](https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49165887)  [TM4C123GXL?preview=49165887](https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49165887))
3. IAR workbench or other IDE
4. LEDs (<https://learn.adafruit.com/all-about-leds/the-led-datasheet>)
5. Push buttons (<https://www.alps.com/prod/info/E/HTML/Tact/SnapIn/SKHH/SKHHAKA010.html>)
6. Debouncing (<https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49719211>)

Figure 1 shows an overview of the different parts of the Launchpad, refer to the user manual (posted on canvas) for more information on the board’s features.

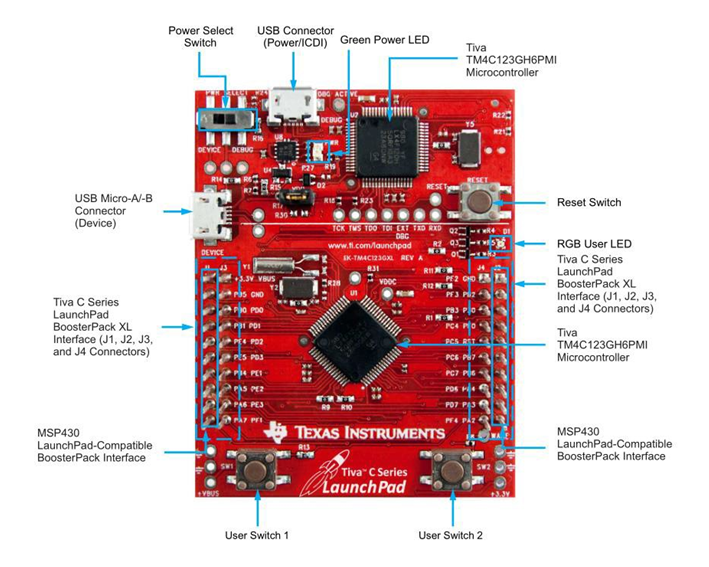


Figure 1 Tiva C Series LaunchPad Components

The RGB user LED consists of three LEDs Red, Green, and Blue and are connected to the processor through Port F (as shown in Figure 2 and Figure 3)

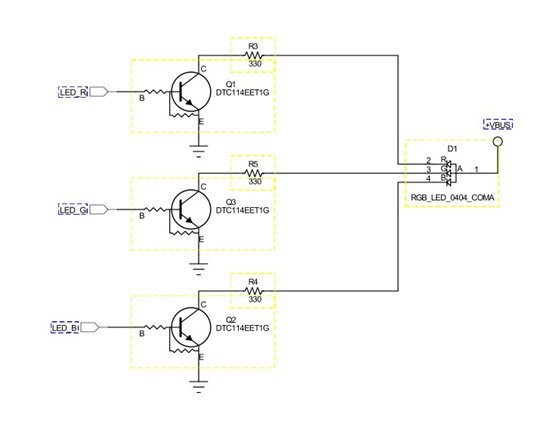


Figure 2 Onboard LED Components

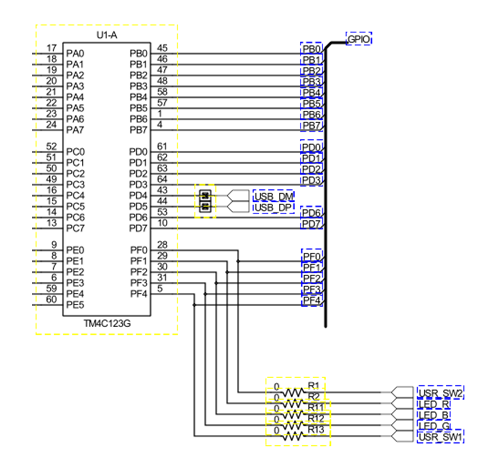


Figure 3 - Port F Pinout

Procedure:

Part A Timers and Timer Modes

It this first part of the lab the student is introduced to the General Purpose Timer Module or GPTM. This module can be used to count up or down to a preset number and then increment a bit in a the GPTM Raw Interrupt Status Register or GPTMRIS. This GPTMRIS Register has bits for each of the timers that may complete their count. The TIVA board has 24 different timers that can be used to mark increments of time.

Part A Initializations and Configurations

There were a number of registers that needed initialization for our purposes. For these initializations pointers were declared that pointed at the register locations, and were dereferenced for the updating of register values. (see Lab 1 documentation for more on pointers and dereferencing.) The following figure 4 shows the initializations of register values and the registers involved.

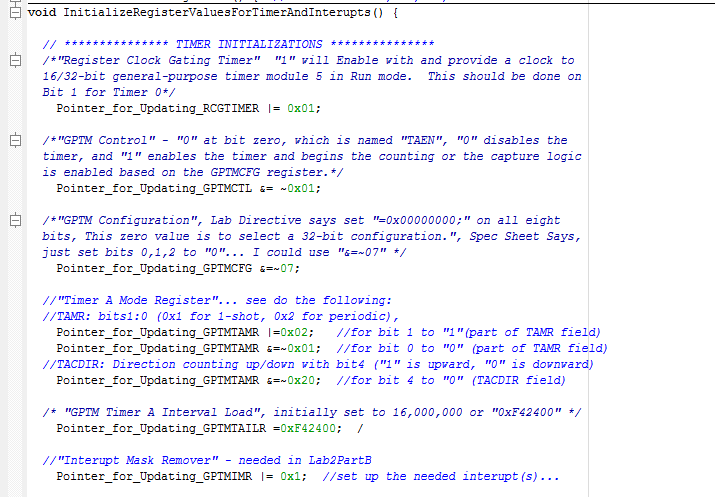


Figure Initialization of Timer Registers

Part A Required Tasks

In part A the student was asked to provide software line coding that would blink each of the on-board LEDs in order each 1 second period with the use of timer. The code written by the student for this is shown in the following figure 5.

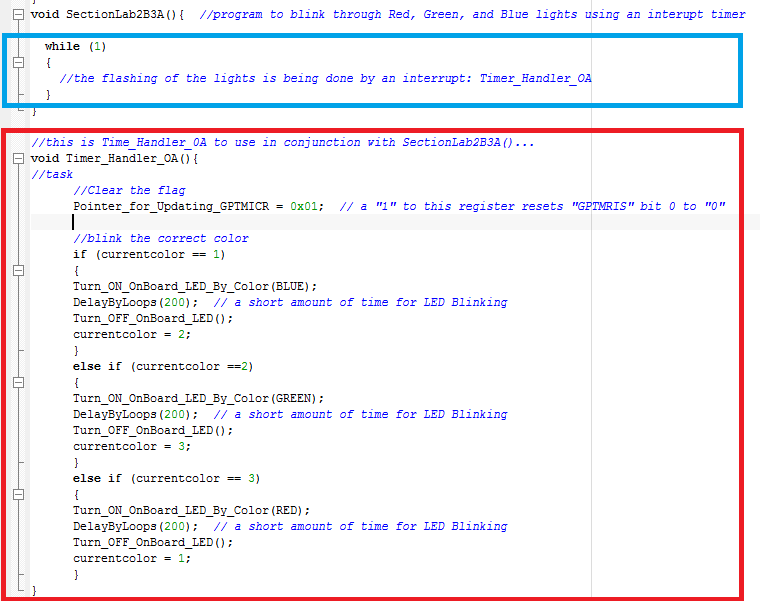


Figure Timer Handler process lighting LEDs

As you can see in the image above in the light blue box the infinite while loop is not doing much. This is because the tasks that are needed are being handled by "Timer\_Handler\_OA()" whenever the timer counts down from 16,000,000 to determine a 1 second time period. You may also notice in the code that the LEDs are turned on for a duration of 200 loops in "DelayByLoops()". This 200 loops allows for approximately a 0.1 second duration flash.

Also, in this part of this lab the student was asked to update the traffic light system from the previous lab to include Timer 0A. This timer was used to do two tasks; both to measure a 2-second push of the Start-stop button or Passenger Button and to measure a 5-second delay between state changes. Accomplishing as much meant utilizing variables to count the number of timer countdowns that were being accomplished whenever such a count was needed. The line code for this can be found at the end of the lab.

Figure 6 below shows the hardware design of the electrical circuit of the system.

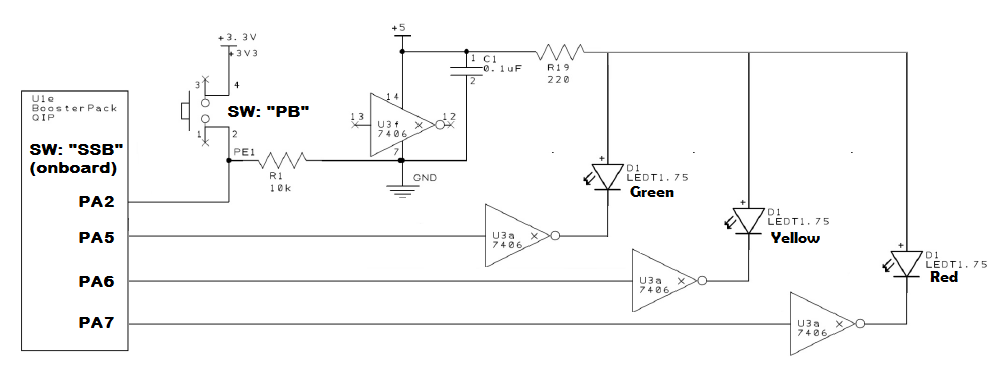


Figure 6 Circuitry for hardware implementation of system

Also, in Figure 7 below shows the finite state machine with the states and change causing events thereof.

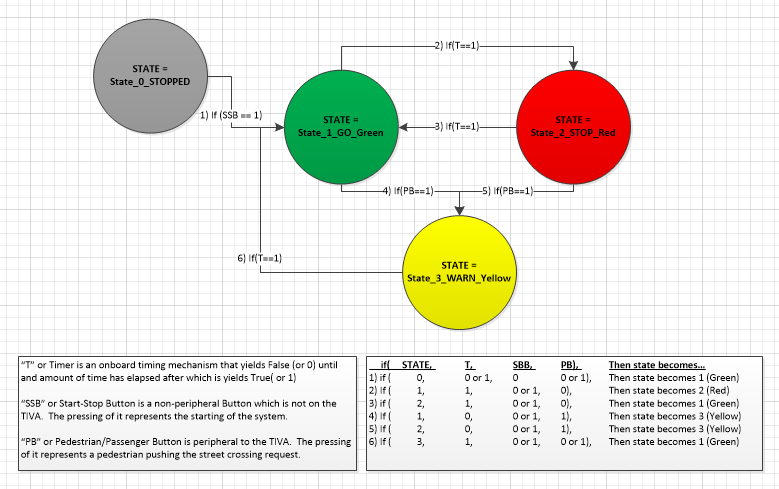


Figure Finite State Machine

Part B Interrupts and The Vector Table

In the second part of the Lab, Part B, the student was introduced to Interrupts and the Vector Table. The following Figure Number 8 is showing the portion of the code in a files named "cstartup\_M.c" where interupt handlers are placed in their location in the Interrupt Register. You can see in the red boxes that two handlers are named "GPIO\_PORT\_A\_Handler" and "Timer\_Handler\_OA."

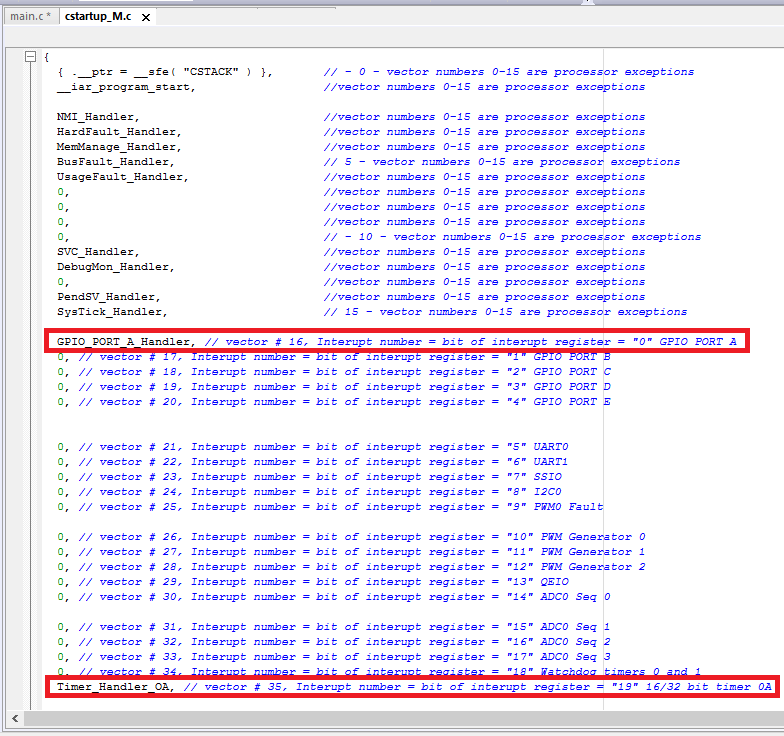


Figure 8 Vector Table Showing Handlers in Red

Also included in this file, but not shown, are the declarations of these type void external handler functions and their definitions. Here is an example of a declaration:

extern void GPIO\_PORT\_A\_Handler( void );

Also, here is an example of a handler function definition:

#pragma call\_graph\_root = "interrupt"  
\_\_weak void GPIO\_PORT\_F\_Handlerr( void ) { while (1) {} }

This cstartup\_M.c file was then included in the project where the handlers would be utilized.

Part B Implementing Interrupt Service Routine for Timer OA and Switches

At this point, with the vector table updated, the student returned to the main.c to implement Interrupt Service Routines or ISRs. These ISRs were written in their own void functions and were not called by the main() loop or subroutines. They were carried out whenever the timer would countdown to zero or the appropriate button connected to Port A was pressed.

Examples of the ISRs written by the student can be found in Figure 7.

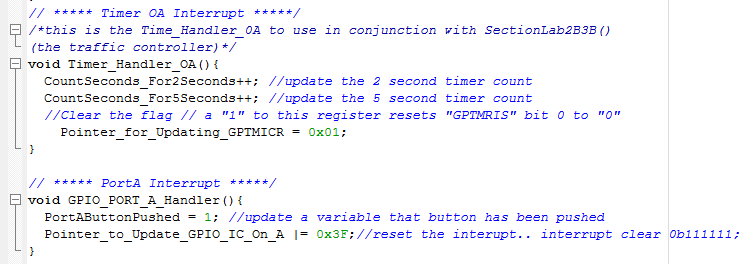


Figure Examples of Interrupt Service Routines

Timer Handlers and GPIO Port Handlers were used to improve the traffic light system. Whereas previously timers and the states of switches were continuously polled, now they were updated as soon as they were triggered with the use of interrupts.

Lastly, a new system was implemented in which a timer was utilized only when the one of the buttons had been pressed. In this system Timer\_Handler\_OA was enabled or disabled with the press of each onboard switch. Figure 8 below shows the code written by the student for this system.

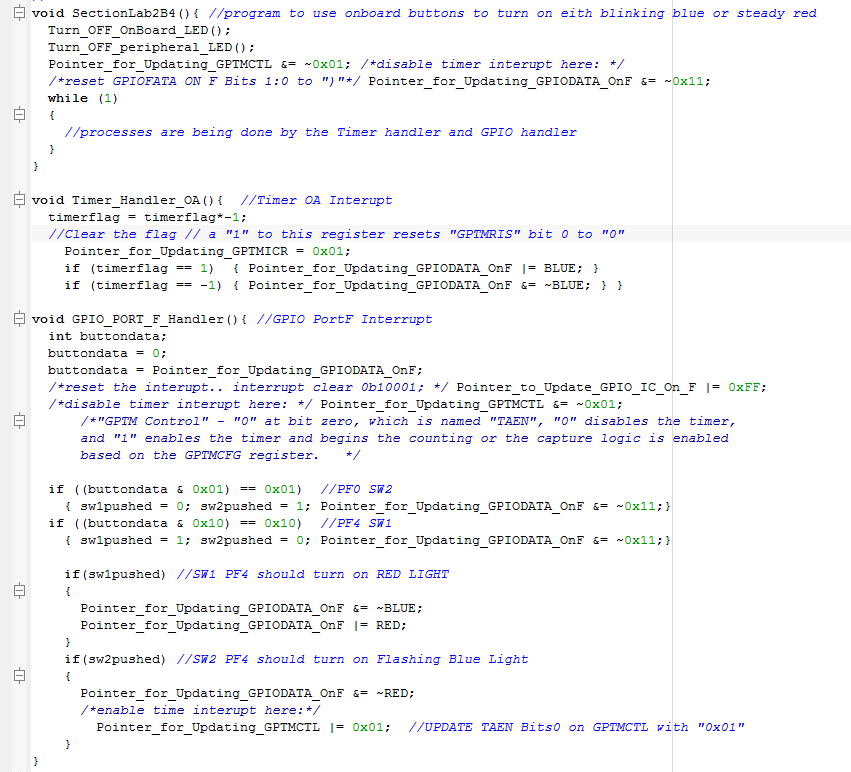


Figure Timer ISR Subject to Control of GPIO ISR

Results:

The required designs, and implementation in both software and hardware were completed by the student. The systems were able to be completed to operate as expected. There was some initial confusion as to which of the code initializations, declarations, definitions, instantiations, implementations of ISRs, and subordinations of ISR were to occur in which part of the code whether it be in the main, or cstartup\_M.c file, or ISRs. This seems to have been cleared up through the scouring a mixture of lab directives, lecture material, user manuals, other online material, and peer-to-peer communication. What went from feeling impossible suddenly started to make sense.

Conclusion:

The Student was able to come to an understanding of the use of timers and interrupts. Their usefulness became clear and the student was able to implement them in code. The ongoing updating of the traffic light design has been helpful in seeing why and how the Timer and interrupts could be utilized.

References*:*

*Tiva™ TM4C123GH6PM Microcontroller DATA SHEET. (2017). [ebook] Available at: https://canvas.uw.edu/courses/1205180/files/folder/Ek- TM4C123GXL?preview=49165887 [Accessed 29 Jun. 2018].*

Student Coded Source Files:

/\* This program is from Lab 2 Part A Required Task 1... "Update the program that

you used in Lab 1 Section A such that you blink one of the 3 LEDs every 3 second.\*/

//#include <tm4c123gh6pm.h>

#include <stdint.h>

#define RED 0x02 // "00010"

#define BLUE 0x04 // "00100"

#define GREEN 0x08 // "01000"

#define YELLOW 0x0A // "01010"

#define WHITE 0x0E // "01110"

#define PURPLE 0x06 // "00110"

#define BUTTON1 0x10 // "10000"

#define BUTTON2 0x01 // "00001"

// pointer for LAB1 Turning on an LED

#define Pointer\_for\_Updating\_RCGCGPIO (\*((volatile uint32\_t \*) 0x400FE608)) //Bit 5 needs to be turned on for Port F, Bit 1 needs to be on for Port A

#define Pointer\_for\_Updating\_GPIODEN (\*((volatile uint32\_t \*) 0x4002551C)) //Bit 7 needs to be turned on

#define Pointer\_for\_Updating\_GPIODIR (\*((volatile uint32\_t \*) 0x40025400)) //Bit 0 and 4 for the buttons need to be "0" for input, and Bits 1,2,3 need to be "1" for output

#define Pointer\_for\_Updating\_GPIODATA (\*((volatile uint32\_t \*) 0x400253FC)) //Bit 1,2,3 are used to turn on and off the LEDs, 1AND3 make Yellow, 1AND2AND3 make white light

#define Pointer\_for\_Updating\_GPIOLOCK (\*((volatile uint32\_t \*) 0x40025520)) //to program button PF0 value of 0x0 at bits 31:0 will unlock it so that it may be modified

#define Pointer\_for\_Updating\_GPIOCR (\*((volatile uint32\_t \*) 0x40025524)) //must be "1" on bits 7:0 to allow bits to be written to GPIOPUR

#define Pointer\_for\_Updating\_GPIOPUR (\*((volatile uint32\_t \*) 0x40025510)) //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up

//pointers for LAB2 TIMERS/INTERUPTS

#define Pointer\_for\_Updating\_RCGTIMER (\*((volatile uint32\_t \*) 0x400FE604)) //"Register Clock Gating Timer" "1" will Enable with and provide a clock to 16/32-bit general-purpose timer module 5 in Run mode. This should be done on Bit 1 for Timer 0

#define Pointer\_for\_Updating\_GPTMCTL (\*((volatile uint32\_t \*) 0x4003000C)) //"GPTM Control" - "0" at bit zero, which is named "TAEN", "0" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

#define Pointer\_for\_Updating\_GPTMCFG (\*((volatile uint32\_t \*) 0x40030000)) //"GPTM Configuration", Lab Directive says set "=0x00000000;" on all eight bits, This zero value is to select a 32-bit configuration.", Spec Sheet Says, just set bits 0,1,2 to "0"... I could use "&=~07"

#define Pointer\_for\_Updating\_GPTMTAMR (\*((volatile uint32\_t \*) 0x40030004)) //"Timer A Mode Register"... see the following:

//TAMR: bits1:0 (0x1 for 1-shot, 0x2 for periodic),

//TACDIR: Direction counting up/down with bit4 ("1" is upward, "0" is downward)

#define Pointer\_for\_Updating\_GPTMTAILR (\*((volatile uint32\_t \*) 0x40030028)) // "GPTM Timer A Interval Load", initially set to 16,000,000 or "0xF42400"

#define Pointer\_for\_Updating\_GPTMIMR (\*((volatile uint32\_t \*) 0x40030018)) //"Interupt Mask Remover" - will be needed in Lab2PartB

#define Pointer\_for\_Updating\_GPTMRIS (\*((volatile uint32\_t \*) 0x4003001C)) // "GPTM Raw Interrupt Status"

#define Pointer\_for\_Updating\_GPTMICR (\*((volatile uint32\_t \*) 0x40030024)) // "GPTM Interupt Clear" - "GPTM Interrupt Clear, TATOCINT "GPTM Timer A Time-Out Raw Interrupt" on BIT 0 set to "1"

int color; //declare a variable for the "TurnOnColor" function

void Turn\_On\_Color(int color); //Declare the "TurnOnColor" function, Notice that iti s igiven a definition below main()

void Turn\_OFF\_ONBOARD\_LED();

int delaytime = 2000; //declare a variable for delay function;

void Delay(int delaytime); //Declare daley function //Notice that the funciton is defined below main()

int StateOfSwitch1 = 0;

int StateOfSwitch2 = 0;

int CheckStateOfSwitch(int switchnumber);

void InitializeRegisterValuesForTimerAndInterupts();

void SectionA3(); //program expanded to light up leds in order

void SectionA5(); //program to incorporate onboard switches

void SectionLab2A1(); //LAB 2 SECTION A1 program to blink an LED each 3 seconds with timer

int main()

{

//Uncomment the Selection to compile and upload to board

//SectionA3(); //LAB1 program expanded to light up all leds continuously

//SectionA5(); //LAB1 program to incorporate switches

SectionLab2A1(); //LAB2 program to blink an LED each 3 seconds with timer

return 0;

}

void SectionLab2A1(){ //LAB 2 SECTION A1 program to blink an LED each 3 seconds with timer

int timerflag = 0;

int currentcolor = 1; //1 for blue, 2 for green, 3 for red

InitializeRegisterValuesForTimerAndInterupts();

while (1)

{

timerflag = Pointer\_for\_Updating\_GPTMRIS; //monitor

if (timerflag)

{

timerflag = 0;

Pointer\_for\_Updating\_GPTMRIS = 0x00; //reading zero into this doesn't actually set it back to zero... instead do this...

Pointer\_for\_Updating\_GPTMICR = 0x01; // a "1" to this register resets "GPTMRIS" bit 0 to "0"

if (currentcolor == 1)

{

Turn\_On\_Color(BLUE);

Delay(200); // a short amount of time for LED Blinking

Turn\_OFF\_ONBOARD\_LED();

currentcolor = 2;

}

else if (currentcolor ==2)

{

Turn\_On\_Color(GREEN);

Delay(200); // a short amount of time for LED Blinking

Turn\_OFF\_ONBOARD\_LED();

currentcolor = 3;

}

else if (currentcolor == 3)

{

Turn\_On\_Color(RED);

Delay(200); // a short amount of time for LED Blinking

Turn\_OFF\_ONBOARD\_LED();

currentcolor = 1;

}

}

}

}

void Turn\_On\_Color(int color) {

Pointer\_for\_Updating\_RCGCGPIO = 0x20;

Pointer\_for\_Updating\_GPIODIR = color;

Pointer\_for\_Updating\_GPIODEN = color;

Pointer\_for\_Updating\_GPIODATA = 0x0;

Pointer\_for\_Updating\_GPIODATA = color;

}

void Turn\_OFF\_ONBOARD\_LED() {

Pointer\_for\_Updating\_RCGCGPIO = 0x20;

Pointer\_for\_Updating\_GPIODIR = color;

Pointer\_for\_Updating\_GPIODEN = color;

Pointer\_for\_Updating\_GPIODATA = 0x0;

}

void Delay(int delaytime) {

for (int i = 0; i <= delaytime; i++)

{

for (int j = 1; j <= 1000; j++)

{}

}

}

int CheckStateOfSwitch(int switchnumber) {

int state = 0;

Pointer\_for\_Updating\_RCGCGPIO = 0x20; //enable RCGCGPIO Clock gating at 0x400FE608 bit 5 as a "1" with "0x20" using a pointer // "100000"

Pointer\_for\_Updating\_GPIODEN = 0x11; //enable digital button pins on Port F at 0x4002551C // "10001"

Pointer\_for\_Updating\_GPIODIR = 0x0E; //set Port F pins 0 and 4 as "input" at 0x40025400 //"01110"

Pointer\_for\_Updating\_GPIOLOCK = 0x4C4F434B; //to program button PF0 a value of 0x4C4F434B must be set at 0x40025520 bits 31:0 will unlock it so that it may be modified //"00000000"

Pointer\_for\_Updating\_GPIOCR = 0x11; //must be "1" on bits 7:0 to allow bits to be written to GPIOPUR // "10000" or "00001"

Pointer\_for\_Updating\_GPIOPUR = 0x11; //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up // "10000" or "00001"

state = Pointer\_for\_Updating\_GPIODATA; // should be found at 0x400253FC

state = state & switchnumber;

if (state == 0x10)

{state = 0;}

else if (state == 0x01)

{state = 0;}

else if (state == 0x00)

{state = 1;}

return (state);

}

void InitializeRegisterValuesForTimerAndInterupts() {

Pointer\_for\_Updating\_RCGTIMER |= 0x01; //"Register Clock Gating Timer" "1" will Enable with and provide a clock to 16/32-bit general-purpose timer module 5 in Run mode. This should be done on Bit 1 for Timer 0

Pointer\_for\_Updating\_GPTMCTL &= ~0x01; //"GPTM Control" - "0" at bit zero, which is named "TAEN", ")" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

Pointer\_for\_Updating\_GPTMCFG &=~07; //"GPTM Configuration", Lab Directive says set "=0x00000000;" on all eight bits, This zero value is to select a 32-bit configuration.", Spec Sheet Says, just set bits 0,1,2 to "0"... I could use "&=~07"

//"Timer A Mode Register"... see do the following:

//TAMR: bits1:0 (0x1 for 1-shot, 0x2 for periodic),

Pointer\_for\_Updating\_GPTMTAMR |=0x02; //for bit 1 to "1"(part of TAMR field)

Pointer\_for\_Updating\_GPTMTAMR &=~0x01; //for bit 0 to "0" (part of TAMR field)

//TACDIR: Direction counting up/down with bit4 ("1" is upward, "0" is downward)

Pointer\_for\_Updating\_GPTMTAMR &=~0x20; //for bit 4 to "0" (TACDIR field)

Pointer\_for\_Updating\_GPTMTAILR =0xF42400; // "GPTM Timer A Interval Load", initially set to 16,000,000 or "0xF42400"

//"Interupt Mask Remover" - will be needed in Lab2PartB

//Pointer\_for\_Updating\_GPTMIMR = \_\_\_\_;

//UPDATE TAEN Bits0 on GPTMCTL with "0x01"

Pointer\_for\_Updating\_GPTMCTL |= 0x01; //"GPTM Control" - "0" at bit zero, which is named "TAEN", ")" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

//These will be used in the code later, but not set here now

//Pointer\_for\_Updating\_GPTMRIS = \_\_\_\_; // "GPTM Raw Interrupt Status"

//Pointer\_for\_Updating\_GPTMICR = \_\_\_\_; // "GPTM Interupt Clear" - "GPTM Interrupt Clear, TATOCINT "GPTM Timer A Time-Out Raw Interrupt" on BIT 0 set to "1"

}

void SectionA3() { //program expanded to light up all leds continuously

while (1)

{

Turn\_On\_Color(RED);

Delay(delaytime);

Turn\_On\_Color(YELLOW);

Delay(delaytime);

Turn\_On\_Color(GREEN);

Delay(delaytime);

Turn\_On\_Color(BLUE);

Delay(delaytime);

Turn\_On\_Color(PURPLE);

Delay(delaytime);

Turn\_On\_Color(WHITE);

Delay(delaytime);

}

}

void SectionA5(){ //program to incorporate switches

while (1)

{

StateOfSwitch1 = CheckStateOfSwitch(BUTTON1);

StateOfSwitch2 = CheckStateOfSwitch(BUTTON2);

if (StateOfSwitch1 == 1)

{

Turn\_On\_Color(RED);

}

else

{

Pointer\_for\_Updating\_GPIODATA = 0x0; // clear all Port F

}

if (StateOfSwitch2 == 1)

{

Turn\_On\_Color(BLUE);

}

else

{

Pointer\_for\_Updating\_GPIODATA = 0x0; // clear all Port F

}

}

}

// program for EE474 Lab2 Part A Task #2 - in which a portion of Lab1 B is rewritten and updated with timers and interrupts

// written by student Jeff Josephsen UWStudentID# 0860205

//#include <tm4c123gh6pm.h>

#include <stdint.h>

//delare Constants

#define RED 0x02 // "00010"

#define BLUE 0x04 // "00100"

#define GREEN 0x08 // "01000"

#define YELLOW 0x0A // "01010"

#define WHITE 0x0E // "01110"

#define PURPLE 0x06 // "00110"

#define BUTTON1 0x10 // "10000"

#define BUTTON2 0x01 // "00001"

#define PERIPHERAL\_BUTTON\_A 0x04 // 0x20 or "100000" when PA5 is HIGH, "010000" when PA4 is HIGH, "001000" when PA3 is HIGH, "000100" when PA2 is HIGH

//#define PERIPHERAL\_BUTTON\_B 0x01 //

//declare Pointers

// Pointers for All Ports...

#define Pointer\_for\_Updating\_RCGCGPIO\_OnPortsX (\*((volatile uint32\_t \*) 0x400FE608)) //Bit 5 needs to be turned on for Port F, Bit 1 needs to be on for Port A

// Pointers for Port F...

#define Pointer\_for\_Updating\_GPIODEN\_OnF (\*((volatile uint32\_t \*) 0x4002551C)) //Bit 7 needs to be turned on

#define Pointer\_for\_Updating\_GPIODIR\_OnF (\*((volatile uint32\_t \*) 0x40025400)) //Bit 0 and 4 for the buttons need to be "0" for input, and Bits 1,2,3 need to be "1" for output

#define Pointer\_for\_Updating\_GPIODATA\_OnF (\*((volatile uint32\_t \*) 0x400253FC)) //Bit 1,2,3 are used to turn on and off the LEDs, 1AND3 make Yellow, 1AND2AND3 make white light

#define Pointer\_for\_Updating\_GPIOLOCK\_OnF (\*((volatile uint32\_t \*) 0x40025520)) //to program button PF0 value of 0x0 at bits 31:0 will unlock it so that it may be modified

#define Pointer\_for\_Updating\_GPIOCR\_OnF (\*((volatile uint32\_t \*) 0x40025524)) //must be "1" on bits 7:0 to allow bits to be written to GPIOPUR

#define Pointer\_for\_Updating\_GPIOPUR\_OnF (\*((volatile uint32\_t \*) 0x40025510)) //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up

// Pointers for Port A...

#define Pointer\_for\_Updating\_GPIOAMSEL\_R\_OnA (\*((volatile uint32\_t \*) 0x40004528)) //used to enable/isolate analog ciruits/functions from the port/pins and "0" =isolated, "1" = capable of analog functioning

#define Pointer\_for\_Updating\_GPIOPCTL\_R\_OnA (\*((volatile uint32\_t \*) 0x4000452C)) //GPIO Port Control is in conjunction with GPIOAFSEL

#define Pointer\_for\_Updating\_GPIODIR\_OnA (\*((volatile uint32\_t \*) 0x40004400)) //Used for setting pin directions (in "0"/out "1") on Port A

#define Pointer\_for\_Updating\_GPIOAFSEL\_OnA (\*((volatile uint32\_t \*) 0x40004420)) //

#define Pointer\_for\_Updating\_GPIODEN\_OnA (\*((volatile uint32\_t \*) 0x4000451C)) //This is used to enable pins on prt as Digital

#define Pointer\_for\_Updating\_GPIODATA\_OnA (\*((volatile uint32\_t \*) 0x400043FC)) // vs "..."080" //this is for grabbing the value of Data on Port A

#define Pointer\_for\_Updating\_GPIOPUR\_OnA (\*((volatile uint32\_t \*) 0x40004510)) //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up

#define Pointer\_for\_BUTTONDATA (\*((volatile uint32\_t \*) 0x40004080))

//pointers for LAB2 TIMERS/INTERUPTS

#define Pointer\_for\_Updating\_RCGTIMER (\*((volatile uint32\_t \*) 0x400FE604)) //"Register Clock Gating Timer" "1" will Enable with and provide a clock to 16/32-bit general-purpose timer module 5 in Run mode. This should be done on Bit 1 for Timer 0

#define Pointer\_for\_Updating\_GPTMCTL (\*((volatile uint32\_t \*) 0x4003000C)) //"GPTM Control" - "0" at bit zero, which is named "TAEN", "0" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

#define Pointer\_for\_Updating\_GPTMCFG (\*((volatile uint32\_t \*) 0x40030000)) //"GPTM Configuration", Lab Directive says set "=0x00000000;" on all eight bits, This zero value is to select a 32-bit configuration.", Spec Sheet Says, just set bits 0,1,2 to "0"... I could use "&=~07"

#define Pointer\_for\_Updating\_GPTMTAMR (\*((volatile uint32\_t \*) 0x40030004)) //"Timer A Mode Register"... see the following:

//TAMR: bits1:0 (0x1 for 1-shot, 0x2 for periodic),

//TACDIR: Direction counting up/down with bit4 ("1" is upward, "0" is downward)

#define Pointer\_for\_Updating\_GPTMTAILR (\*((volatile uint32\_t \*) 0x40030028)) // "GPTM Timer A Interval Load", initially set to 16,000,000 or "0xF42400"

#define Pointer\_for\_Updating\_GPTMIMR (\*((volatile uint32\_t \*) 0x40030018)) // "Interupt Mask Remover" - will be needed in Lab2PartB

#define Pointer\_for\_Updating\_GPTMRIS (\*((volatile uint32\_t \*) 0x4003001C)) // "GPTM Raw Interrupt Status"

#define Pointer\_for\_Updating\_GPTMICR (\*((volatile uint32\_t \*) 0x40030024)) // "GPTM Interupt Clear" - "GPTM Interrupt Clear, TATOCINT "GPTM Timer A Time-Out Raw Interrupt" on BIT 0 set to "1"

// declare functions

//To Enable (or Disable) specific registers

void EnablePortFRegisters(); //setting register values for onboard ports, buttons, and LEDs

void EnablePortARegisters(); //setting register values for peripherals

void InitializeRegisterValuesForTimerAndInterupts();

int delayloops = 6000;

void DelayByLoops(int delayloops); // delaytime set to 5000 allots about 1 second

int CheckStateOfOnboardSwitch(int switchnumber); // input is in form of this example "BUTTON1" or "0x10"

int CheckStateOfPERIPHERALSwitch(int switchnumber); // input is in form of this example "BUTTON1" or "0x10"

void Turn\_ON\_OnBoard\_LED\_By\_Color(int color);

void Turn\_OFF\_OnBoard\_LED();

void Turn\_ON\_peripheral\_LED\_By\_Color(int color);

void Turn\_OFF\_peripheral\_LED();

enum system\_state\_data\_type{State\_0\_FSM\_STOPPED, State\_1\_STOP\_Red, State\_2\_GO\_Green, State\_3\_WARN\_Yellow};

void DelayByTimerOneSecond();

int DelaySeconds = 2;

int DelayByTimerXSeconds(int DelaySeconds); //this function calls DelayByTimerOneSecond() in 1 loop per second.

void SectionB1(); //program to interface peripheral switches

void SectionB2(); //program for button debouncing

void SectionB3(); //program for lighting LED

int main() {

EnablePortFRegisters(); //setting register values for onboard ports, buttons, and LEDs

EnablePortARegisters(); //setting register values for peripherals

InitializeRegisterValuesForTimerAndInterupts(); //setting register values for Timers and Interupts

// Uncomment the Selection to compile and upload to board

//SectionB1(); //program to interface a peripheral button

//SectionB2(); //program which lights eternal LED via external button

SectionB3(); //program for lighting LED like traffic controller

//return 0;

}

void SectionB1() {

while (1)

{

if (CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A))

{

Turn\_ON\_OnBoard\_LED\_By\_Color(RED);

}

else

{

Turn\_ON\_OnBoard\_LED\_By\_Color(BLUE);

}

}

}

void SectionB2() {

while (1)

{

Turn\_ON\_peripheral\_LED\_By\_Color(RED);

Turn\_OFF\_peripheral\_LED();

// SysCtlDelay(7000000); //uncomment if you need debouncing

}

}

void SectionB3() { //This is Section B3 of Lab1 JJosephsen UW#0860205

Turn\_OFF\_peripheral\_LED();

enum system\_state\_data\_type CurrentState = 0;

int ButtonStateA = 0;

int ButtonStateB = 0;

int CountToTWOSeconds = 0;

int CountToFIVESeconds = 0;

while(1)

{

while (CurrentState ==0) //\_\_\_\_\_State\_0\_FSM\_STOPPED/OFF\_\_\_\_\_

{

ButtonStateA = CheckStateOfOnboardSwitch(BUTTON1);

DelayByTimerOneSecond();

ButtonStateB = CheckStateOfOnboardSwitch(BUTTON1);

if ((ButtonStateA == ButtonStateB) && (ButtonStateA == 1)) //if button state is the same after 1 second

{

CountToTWOSeconds++; //increment button state count of seconds

if(CountToTWOSeconds >= 1) //if button has been pressed for 2 seconds

{

CurrentState = 1; //becomes red state

CountToTWOSeconds = 0; //reset

}

}

else

{

CountToTWOSeconds = 0; //reset

}

}

while (CurrentState ==1) //\_\_\_\_\_State\_1\_STOP\_Red\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(RED);

ButtonStateA = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

DelayByTimerOneSecond();

ButtonStateB = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

if ((ButtonStateA == ButtonStateB)&&(ButtonStateA ==1)) //if button state is the same after 1 second

{

CountToTWOSeconds++; //increment button state count of seconds

if(CountToTWOSeconds >= 1) //if button has been pressed for 2 seconds

{

CurrentState = 3; //becomes yellow state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

CountToFIVESeconds++;

if (CountToFIVESeconds >= 4)

{

CurrentState = 2; //becomes green state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

while (CurrentState ==2) //\_\_\_\_\_State\_2\_GO\_Green\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(GREEN);

ButtonStateA = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

DelayByTimerOneSecond();

ButtonStateB = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

if((ButtonStateA == ButtonStateB)&&(ButtonStateA==1)) //if button state is the same after 1 second

{

CountToTWOSeconds++; //increment button state count of seconds

if(CountToTWOSeconds >= 1) //if button has been pressed for 2 seconds

{

CurrentState = 3; //becomes yellow state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

CountToFIVESeconds++;

if (CountToFIVESeconds >= 4)

{

CurrentState = 1; //becomes red state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

while (CurrentState ==3) //\_\_\_\_\_State\_3\_WARN\_Yellow\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(YELLOW);

DelayByTimerXSeconds(5); //use timer to delay 5 seconds

CurrentState = 2; //becomes green state

}

}

}

int CheckStateOfOnboardSwitch(int switchnumber){ // input is in form of this example "BUTTON1" or "0x10"

int ButtonState = 0;

ButtonState = Pointer\_for\_Updating\_GPIODATA\_OnF; // can be checked at memory space: 0x400253FC

ButtonState = ButtonState & switchnumber;

if (ButtonState == 0x10)

{ButtonState = 0;}

else if (ButtonState == 0x01)

{ButtonState = 0;}

else if (ButtonState == 0x00)

{ButtonState = 1;}

return (ButtonState);

}

int CheckStateOfPERIPHERALSwitch(int switchnumber){ // input is in form of this example "PERIPHERAL\_BUTTON\_A" or "0x02" (for PA2)

int ButtonState = 0;

ButtonState = Pointer\_for\_Updating\_GPIODATA\_OnA;

/\*when button into PA5 is pressed or high, then bit 5 is ON or HIGH \*/

ButtonState = ButtonState & switchnumber; //apply mask of switch in question

if (ButtonState == switchnumber)

{ButtonState = 1;}

else if (ButtonState == 0x00)

{ButtonState = 0;}

return (ButtonState);

}

void Turn\_ON\_OnBoard\_LED\_By\_Color(int color){

Pointer\_for\_Updating\_GPIODATA\_OnF = 0x0;

Pointer\_for\_Updating\_GPIODATA\_OnF = color;

}

void Turn\_OFF\_OnBoard\_LED(){

Pointer\_for\_Updating\_GPIODATA\_OnF = 0x0;

}

void Turn\_ON\_peripheral\_LED\_By\_Color(int color) {

if (color == RED)

{

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x80; //Turn on PA7 (bit 7)

}

else if (color == YELLOW)

{

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x40; //Turn on PA6 (bit 6)

}

else if (color == GREEN)

{

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x10; //Turn on PA5 (bit 5)

}

}

void Turn\_OFF\_peripheral\_LED(){

Pointer\_for\_Updating\_GPIODATA\_OnA &= 0x0; //Turn OFF PA5, PA6, PA7 - why does the green light pop on?

}

void DelayByLoops(int delayloops){

for (int i = 0; i <= delayloops; i++) {

for (int j = 1; j <= 1000; j++) {

}

}

}

void DelayByTimerOneSecond(){

int timerflag = 0;

do{

timerflag = Pointer\_for\_Updating\_GPTMRIS; //monitor/poll the flag

}while (timerflag ==0);

Pointer\_for\_Updating\_GPTMRIS = 0x00; //reading zero into this doesn't actually set it back to zero... instead do this...

Pointer\_for\_Updating\_GPTMICR = 0x01; // a "1" to this register resets "GPTMRIS" bit 0 to "0"

}

int DelayByTimerXSeconds(int DelaySeconds){ //input is limited to positive intergers

for (int i=0; i<=DelaySeconds; i++)

{

DelayByTimerOneSecond();

}

return (1);

}

void EnablePortFRegisters() {

// General...

Pointer\_for\_Updating\_RCGCGPIO\_OnPortsX = 0x21; //enable RCGCGPIO Clock gating at 0x400FE608 bit 5 and 0 turned to "1" with "0x21" or ("100001") ("FEDCBA") using a pointer

//Port F Onboard lights and switches...

Pointer\_for\_Updating\_GPIODEN\_OnF = 0x1F; //enable digital Pins on Port F at 0x4002551C // "11111"

Pointer\_for\_Updating\_GPIODIR\_OnF = 0x0E; //set Port F pins 0 and 4 as "input" at 0x40025400 //"01110"

Pointer\_for\_Updating\_GPIOLOCK\_OnF = 0x4C4F434B; //to program button PF0 a value of 0x4C4F434B must be set at 0x40025520 bits 31:0 will unlock it so that it may be modified //"00000000"

Pointer\_for\_Updating\_GPIOCR\_OnF = 0x11; //must be "1" on bits 7:0 to allow bits to be written to GPIOPUR // "10001"

Pointer\_for\_Updating\_GPIOPUR\_OnF = 0x11; //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up // "10001"

Pointer\_for\_Updating\_GPIODATA\_OnF = 0x00; //reset to zero at start

}

void EnablePortARegisters() { //LEDs are on A5, A6, A7, and Button is on A2

//Port A I/O for Peripheral Buttons and Switches...

Pointer\_for\_Updating\_GPIOAMSEL\_R\_OnA = 0x00; // 0x04 ? // Analog mode select ("0" -> digital, "1"-> Analog)

Pointer\_for\_Updating\_GPIODEN\_OnA = 0xFF; //enable as digital with "1" Enable PortPin A2, A5, A6, A7

Pointer\_for\_Updating\_GPIODIR\_OnA = 0xF0; //"DIR is DIRECTION (in="0"/out="1") [oxFO = "1111 0000"

Pointer\_for\_Updating\_GPIOPCTL\_R\_OnA = ~0xFF0F0F00; // This is PortControl on a MUX Pins 76\_4\_\_2\_\_ are enabled via "0"

Pointer\_for\_Updating\_GPIOAFSEL\_OnA &= 0x00; //"Alternative Function Select"… "0" means set to Digital function and not an alternative function

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x00; //reset to zero at start

}

void InitializeRegisterValuesForTimerAndInterupts() {

Pointer\_for\_Updating\_RCGTIMER |= 0x01; //"Register Clock Gating Timer" "1" will Enable with and provide a clock to 16/32-bit general-purpose timer module 5 in Run mode. This should be done on Bit 1 for Timer 0

Pointer\_for\_Updating\_GPTMCTL &= ~0x01; //"GPTM Control" - "0" at bit zero, which is named "TAEN", ")" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

Pointer\_for\_Updating\_GPTMCFG &=~07; //"GPTM Configuration", Lab Directive says set "=0x00000000;" on all eight bits, This zero value is to select a 32-bit configuration.", Spec Sheet Says, just set bits 0,1,2 to "0"... I could use "&=~07"

//"Timer A Mode Register"... see do the following:

//TAMR: bits1:0 (0x1 for 1-shot, 0x2 for periodic),

Pointer\_for\_Updating\_GPTMTAMR |=0x02; //for bit 1 to "1"(part of TAMR field)

Pointer\_for\_Updating\_GPTMTAMR &=~0x01; //for bit 0 to "0" (part of TAMR field)

//TACDIR: Direction counting up/down with bit4 ("1" is upward, "0" is downward)

Pointer\_for\_Updating\_GPTMTAMR &=~0x20; //for bit 4 to "0" (TACDIR field)

Pointer\_for\_Updating\_GPTMTAILR =0xF42400; // "GPTM Timer A Interval Load", initially set to 16,000,000 or "0xF42400"

//"Interupt Mask Remover" - will be needed in Lab2PartB

//Pointer\_for\_Updating\_GPTMIMR = \_\_\_\_;

//UPDATE TAEN Bits0 on GPTMCTL with "0x01"

Pointer\_for\_Updating\_GPTMCTL |= 0x01; //"GPTM Control" - "0" at bit zero, which is named "TAEN", ")" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

//These will be used in the code later, but not set here now

//Pointer\_for\_Updating\_GPTMRIS = \_\_\_\_; // "GPTM Raw Interrupt Status"

//Pointer\_for\_Updating\_GPTMICR = \_\_\_\_; // "GPTM Interupt Clear" - "GPTM Interrupt Clear, TATOCINT "GPTM Timer A Time-Out Raw Interrupt" on BIT 0 set to "1"

}

// program for EE474 Lab2 Part B - in which a portion of Lab1 B is rewritten and updated with timers and interupts

// written by student Jeff Josephsen UWStudentID# 0860205

//#include <tm4c123gh6pm.h>

#include <stdint.h>

//delare Constants

#define RED 0x02 // "00010"

#define BLUE 0x04 // "00100"

#define GREEN 0x08 // "01000"

#define YELLOW 0x0A // "01010"

#define WHITE 0x0E // "01110"

#define PURPLE 0x06 // "00110"

#define BUTTON1 0x10 // "10000"

#define BUTTON2 0x01 // "00001"

#define PERIPHERAL\_BUTTON\_A 0x04 // 0x20 or "100000" when PA5 is HIGH, "010000" when PA4 is HIGH, "001000" when PA3 is HIGH, "000100" when PA2 is HIGH

#define PERIPHERAL\_BUTTON\_B 0x01 //

//declare Pointers

// Pointers for All Ports...

#define Pointer\_for\_Updating\_RCGCGPIO\_OnPortsX (\*((volatile uint32\_t \*) 0x400FE608)) //Bit 5 needs to be turned on for Port F, Bit 1 needs to be on for Port A

// Pointers for Port F...

#define Pointer\_for\_Updating\_GPIODEN\_OnF (\*((volatile uint32\_t \*) 0x4002551C)) //Bit 7 needs to be turned on

#define Pointer\_for\_Updating\_GPIODIR\_OnF (\*((volatile uint32\_t \*) 0x40025400)) //Bit 0 and 4 for the buttons need to be "0" for input, and Bits 1,2,3 need to be "1" for output

#define Pointer\_for\_Updating\_GPIODATA\_OnF (\*((volatile uint32\_t \*) 0x400253FC)) //Bit 1,2,3 are used to turn on and off the LEDs, 1AND3 make Yellow, 1AND2AND3 make white light

#define Pointer\_for\_Updating\_GPIOLOCK\_OnF (\*((volatile uint32\_t \*) 0x40025520)) //to program button PF0 value of 0x0 at bits 31:0 will unlock it so that it may be modified

#define Pointer\_for\_Updating\_GPIOCR\_OnF (\*((volatile uint32\_t \*) 0x40025524)) //must be "1" on bits 7:0 to allow bits to be written to GPIOPUR

#define Pointer\_for\_Updating\_GPIOPUR\_OnF (\*((volatile uint32\_t \*) 0x40025510)) //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up

// Pointers for Port A...

#define Pointer\_for\_Updating\_GPIOAMSEL\_R\_OnA (\*((volatile uint32\_t \*) 0x40004528)) //used to enable/isolate analog ciruits/functions from the port/pins and "0" =isolated, "1" = capable of analog functioning

#define Pointer\_for\_Updating\_GPIOPCTL\_R\_OnA (\*((volatile uint32\_t \*) 0x4000452C)) //GPIO Port Control is in conjunction with GPIOAFSEL

#define Pointer\_for\_Updating\_GPIODIR\_OnA (\*((volatile uint32\_t \*) 0x40004400)) //Used for setting pin directions (in "0"/out "1") on Port A

#define Pointer\_for\_Updating\_GPIOAFSEL\_OnA (\*((volatile uint32\_t \*) 0x40004420)) //

#define Pointer\_for\_Updating\_GPIODEN\_OnA (\*((volatile uint32\_t \*) 0x4000451C)) //This is used to enable pins on prt as Digital

#define Pointer\_for\_Updating\_GPIODATA\_OnA (\*((volatile uint32\_t \*) 0x400043FC)) // vs "..."080" //this is for grabbing the value of Data on Port A

#define Pointer\_for\_Updating\_GPIOPUR\_OnA (\*((volatile uint32\_t \*) 0x40004510)) //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up

#define Pointer\_for\_BUTTONDATA (\*((volatile uint32\_t \*) 0x40004080))

//pointers for LAB2 TIMERS/INTERUPTS

#define Pointer\_for\_Updating\_RCGTIMER (\*((volatile uint32\_t \*) 0x400FE604)) //"Register Clock Gating Timer" "1" will Enable with and provide a clock to 16/32-bit general-purpose timer module 5 in Run mode. This should be done on Bit 1 for Timer 0

#define Pointer\_for\_Updating\_GPTMCTL (\*((volatile uint32\_t \*) 0x4003000C)) //"GPTM Control" - "0" at bit zero, which is named "TAEN", "0" disables the timer, and "1" enables the timer and begins the counting or the capture logic is enabled based on the GPTMCFG register.

#define Pointer\_for\_Updating\_GPTMCFG (\*((volatile uint32\_t \*) 0x40030000)) //"GPTM Configuration", Lab Directive says set "=0x00000000;" on all eight bits, This zero value is to select a 32-bit configuration.", Spec Sheet Says, just set bits 0,1,2 to "0"... I could use "&=~07"

#define Pointer\_for\_Updating\_GPTMTAMR (\*((volatile uint32\_t \*) 0x40030004)) //"Timer A Mode Register"... see the following:

//TAMR: bits1:0 (0x1 for 1-shot, 0x2 for periodic),

//TACDIR: Direction counting up/down with bit4 ("1" is upward, "0" is downward)

#define Pointer\_for\_Updating\_GPTMTAILR (\*((volatile uint32\_t \*) 0x40030028)) // "GPTM Timer A Interval Load", initially set to 16,000,000 or "0xF42400"

//pointers for part LAB2 Part B Interupts

#define Pointer\_for\_Updating\_GPTMIMR (\*((volatile uint32\_t \*) 0x40030018)) // "Interupt Mask Remover" - will be needed in Lab2PartB... set bit0 to "1"

#define Pointer\_for\_Updating\_GPTMRIS (\*((volatile uint32\_t \*) 0x4003001C)) // "GPTM Raw Interrupt Status"

#define Pointer\_for\_Updating\_GPTMICR (\*((volatile uint32\_t \*) 0x40030024)) // "GPTM Interupt Clear" - p745, TATOCINT "GPTM Timer A Time-Out Raw Interrupt" on BIT 0 set to "1"

#define Pointer\_for\_Updating\_GPIOIS\_On\_A (\*((volatile uint32\_t \*) 0x40004404)) //"GPIO Interupt Sense" "0" for edge, "1" for level Bits 7:0 should be "0" on needed pin

#define Pointer\_for\_Updating\_GPIOIS\_On\_F (\*((volatile uint32\_t \*) 0x40025404)) //"GPIO Interupt Sense" "0" for edge, "1" for level Bits 7:0 should be "0" on needed pin

#define Pointer\_for\_Updating\_GPIOIBE\_On\_A (\*((volatile uint32\_t \*) 0x40004408)) //"GPIO Interupt Both Edges" should be "0" to allow interupt generation to be controlled by GPIOIEV. Should be "1" to allow interupt on both edges of the coresponding pin to trigger input.

#define Pointer\_for\_Updating\_GPIOIBE\_On\_F (\*((volatile uint32\_t \*) 0x40025408)) //"GPIO Interupt Both Edges" should be "0" to allow interupt generation to be controlled by GPIOIEV. Should be "1" to allow interupt on both edges of the coresponding pin to trigger input.

#define Pointer\_for\_Updating\_GPIOIEV\_On\_A (\*((volatile uint32\_t \*) 0x4000440C)) //"GPIO Interput Event" "0" for the falling edge or low leve on a coorespinding pin. "1" for the rising or highlevel on the corresponding pin... to trigger an interupt.

#define Pointer\_for\_Updating\_GPIOIEV\_On\_F (\*((volatile uint32\_t \*) 0x4002540C)) //"GPIO Interput Event" "0" for the falling edge or low leve on a coorespinding pin. "1" for the rising or highlevel on the corresponding pin... to trigger an interupt.

#define Pointer\_for\_Updating\_GPIOIM\_On\_A (\*((volatile uint32\_t \*) 0x40004410)) //"GPIO Interrupt Mask" "0" means that the interrutp from the corresponding mpin is masked, "1" means that the interrupt from the corresponding pin is sent to the interrupt controller.

#define Pointer\_for\_Updating\_GPIOIM\_On\_F (\*((volatile uint32\_t \*) 0x40025410)) //"GPIO Interrupt Mask" "0" means that the interrutp from the corresponding mpin is masked, "1" means that the interrupt from the corresponding pin is sent to the interrupt controller.

#define Pointer\_for\_Updating\_GPIOMIS\_OnA (\*((volatile uint32\_t \*) 0x40000418)) // "GPIO MASKED Interrupt Status"

#define Pointer\_for\_Updating\_GPIOMIS\_OnF (\*((volatile uint32\_t \*) 0x40025418)) // "GPIO MASKED Interrupt Status"

#define Pointer\_for\_Updating\_NVIC (\*((volatile uint32\_t \*) 0xE000E100)) //enable interupt at NVIC Page 141-142 pf user manual Base 0xE000E000 with offeset of 0x100 for EN0 = 0xE000E1000 = >bit 19 turned on "0" means that interupt is disabled, and "1" means that intereupt is enabled. Note: to disable interupts you reset them at memory location 0xE000.E000 offset pin 0x180

#define Pointer\_to\_disable\_interupts (\*((volatile uint32\_t \*) 0xE000E180)) //to disable interupts you reset them at memory location 0xE000.E000 offset 0x180 and for this purpose pin 19.

#define Pointer\_to\_Update\_Priority0\_Int0\_3 (\*((volatile uint32\_t \*) 0xE000E400)) //Priority level: interupt level must be less than BASEPRI. //BASE 0xE000.E000 offset 0x400; interupt priority 19 is at 0x410 and should be set to: less than BASEPRI

#define Pointer\_to\_Update\_Priority4\_Int16\_19 (\*((volatile uint32\_t \*) 0xE000E410)) //Priority level: interupt level must be less than BASEPRI. //BASE 0xE000.E000 offset 0x400; interupt priority 19 is at 0x410 and should be set to: less than BASEPRI

#define Pointer\_to\_Update\_Priority7\_Int28\_31 (\*((volatile uint32\_t \*) 0xE000E41C)) //Priority level: interupt level must be less than BASEPRI. //BASE 0xE000.E000 offset 0x400; interupt priority 19 is at 0x410 and should be set to: less than BASEPRI

#define Pointer\_to\_Update\_PRIMASK (\*((volatile uint32\_t \*) 0x00000000)) //enable interupts globally on I-bit of the PRIMASK (Priority mask); BASE AT 0x0000.0000 bit 0 should be a "1" to prevent the activation of all exceptions with configurable priority. "0" for no effect. This register is only accessible in "privledged mode."

#define Pointer\_to\_Update\_GPIO\_IC\_On\_A (\*((volatile uint32\_t \*) 0x4000441C)) //Interrupt Clear on PortA

#define Pointer\_to\_Update\_GPIO\_IC\_On\_F (\*((volatile uint32\_t \*) 0x4002541C)) //Interrupt Clear on PortF

// declare functions

//To Enable (or Disable) specific registers

void InitializePortFRegisters(); //setting register values for onboard ports, buttons, and LEDs

void InitializePortARegisters(); //setting register values for peripherals

void InitializeRegisterValuesForTimerAndInterupts();

int delayloops = 6000;

void DelayByLoops(int delayloops); // delaytime set to 5000 allots about 1 second

int CheckStateOfOnboardSwitch(int switchnumber); // input is in form of this example "BUTTON1" or "0x10"

int CheckStateOfPERIPHERALSwitch(int switchnumber); // input is in form of this example "BUTTON1" or "0x10"

void Turn\_ON\_OnBoard\_LED\_By\_Color(int color);

void Turn\_OFF\_OnBoard\_LED();

void Turn\_ON\_peripheral\_LED\_By\_Color(int color);

void Turn\_OFF\_peripheral\_LED();

enum system\_state\_data\_type{State\_0\_FSM\_STOPPED, State\_1\_STOP\_Red, State\_2\_GO\_Green, State\_3\_WARN\_Yellow};

void DelayByTimerOneSecond();

int DelaySeconds = 2;

int DelayByTimerXSeconds(int DelaySeconds); //this function calls DelayByTimerOneSecond() in 1 loop per second.

int currentcolor = 1; //1 for blue, 2 for green, 3 for red //this variable needs to be global for the interrupt in

int CountSeconds\_For2Seconds = 0;

int CountSeconds\_For5Seconds = 0;

int PortAButtonPushed = 0;

int PortFButtonPushed = 0;

int sw1pushed = 0;

int sw2pushed = 0;

int timerflag = -1; //"1" to go "ON", "-1" to go "OFF" flipped with timerflag = timerflag \* -1;

void SectionLab1B1(); //program to interface peripheral switches

void SectionLab1B2(); //program for button debouncing

void SectionLab1B3(); //program for lighting LED like traffic controller

void SectionLab2B3A(); //program to blink through Red, Green, and Blue lights using an interupt timer

void SectionLab2B3B(); //program to update traffic controller with timer interupts and button interupts

void SectionLab2B4(); //program to use onboard buttons to turn on eith blinking blue or steady red

int main() {

InitializePortFRegisters(); //setting register values for onboard ports, buttons, and LEDs

InitializePortARegisters(); //setting register values for peripherals

InitializeRegisterValuesForTimerAndInterupts(); //setting register values for Timers and Interupts

// Uncomment the Selection to compile and upload to board

//SectionLab1B1(); //program to interface a peripheral button

//SectionLab1B2(); //program which lights eternal LED via external button

//SectionLab2B3A(); //program to blink through Red, Green, and Blue lights using an interupt timer

//SectionLab2B3B(); //program to update traffic controller with timer interupts and button interupts

SectionLab2B4(); //program to use onboard buttons to turn on eith blinking blue or steady red

//return 0;

}

void SectionLab1B1(){ //program to interface peripheral switches

while (1)

{

Turn\_ON\_peripheral\_LED\_By\_Color(RED);

Turn\_OFF\_peripheral\_LED();

}

}

void SectionLab1B2(){ //program for button debouncing

// SysCtlDelay(7000000); //uncomment and place in appropriate location of ocode if you need debouncing

while (1)

{

if (CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A))

{

Turn\_ON\_OnBoard\_LED\_By\_Color(RED);

}

else

{

Turn\_ON\_OnBoard\_LED\_By\_Color(BLUE);

}

}

}

void SectionLab1B3(){ //program for lighting LED like traffic controller

Turn\_OFF\_peripheral\_LED();

enum system\_state\_data\_type CurrentState = 0;

int ButtonStateA = 0;

int ButtonStateB = 0;

int CountToTWOSeconds = 0;

int CountToFIVESeconds = 0;

while(1)

{

while (CurrentState ==0) //\_\_\_\_\_State\_0\_FSM\_STOPPED/OFF\_\_\_\_\_

{

ButtonStateA = CheckStateOfOnboardSwitch(BUTTON1);

DelayByTimerOneSecond();

ButtonStateB = CheckStateOfOnboardSwitch(BUTTON1);

if ((ButtonStateA == ButtonStateB) && (ButtonStateA == 1)) //if button state is the same after 1 second

{

CountToTWOSeconds++; //increment button state count of seconds

if(CountToTWOSeconds >= 1) //if button has been pressed for 2 seconds

{

CurrentState = 1; //becomes red state

CountToTWOSeconds = 0; //reset

}

}

else

{

CountToTWOSeconds = 0; //reset

}

}

while (CurrentState ==1) //\_\_\_\_\_State\_1\_STOP\_Red\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(RED);

ButtonStateA = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

DelayByTimerOneSecond();

ButtonStateB = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

if ((ButtonStateA == ButtonStateB)&&(ButtonStateA ==1)) //if button state is the same after 1 second

{

CountToTWOSeconds++; //increment button state count of seconds

if(CountToTWOSeconds >= 1) //if button has been pressed for 2 seconds

{

CurrentState = 3; //becomes yellow state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

CountToFIVESeconds++;

if (CountToFIVESeconds >= 4)

{

CurrentState = 2; //becomes green state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

while (CurrentState ==2) //\_\_\_\_\_State\_2\_GO\_Green\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(GREEN);

ButtonStateA = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

DelayByTimerOneSecond();

ButtonStateB = CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A);

if((ButtonStateA == ButtonStateB)&&(ButtonStateA==1)) //if button state is the same after 1 second

{

CountToTWOSeconds++; //increment button state count of seconds

if(CountToTWOSeconds >= 1) //if button has been pressed for 2 seconds

{

CurrentState = 3; //becomes yellow state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

CountToFIVESeconds++;

if (CountToFIVESeconds >= 4)

{

CurrentState = 1; //becomes red state

CountToTWOSeconds = 0; //reset

CountToFIVESeconds = 0; //reset

}

}

while (CurrentState ==3) //\_\_\_\_\_State\_3\_WARN\_Yellow\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(YELLOW);

DelayByTimerXSeconds(5); //use timer to delay 5 seconds

CurrentState = 2; //becomes green state

}

}

}

void SectionLab2B3A(){ //program to blink through Red, Green, and Blue lights using an interupt timer

while (1)

{

//the flashing of the lights is being done by an interrupt: Timer\_Handler\_OA

}

}

//this is Time\_Handler\_0A to use in conjunction with SectionLab2B3A()...

void Timer\_Handler\_OA(){

//task

//Clear the flag

Pointer\_for\_Updating\_GPTMICR = 0x01; // a "1" to this register resets "GPTMRIS" bit 0 to "0"

//blink the correct color

if (currentcolor == 1)

{

Turn\_ON\_OnBoard\_LED\_By\_Color(BLUE);

DelayByLoops(200); // a short amount of time for LED Blinking

Turn\_OFF\_OnBoard\_LED();

currentcolor = 2;

}

else if (currentcolor ==2)

{

Turn\_ON\_OnBoard\_LED\_By\_Color(GREEN);

DelayByLoops(200); // a short amount of time for LED Blinking

Turn\_OFF\_OnBoard\_LED();

currentcolor = 3;

}

else if (currentcolor == 3)

{

Turn\_ON\_OnBoard\_LED\_By\_Color(RED);

DelayByLoops(200); // a short amount of time for LED Blinking

Turn\_OFF\_OnBoard\_LED();

currentcolor = 1;

}

}

void SectionLab2B3B(){ //program to update traffic controller with timer interupts and button interupts

Turn\_OFF\_peripheral\_LED();

Turn\_OFF\_OnBoard\_LED();

enum system\_state\_data\_type CurrentState = 0;

while(1)

{

//all lights should be off

while (CurrentState ==0) //\_\_\_\_\_State\_0\_FSM\_STOPPED/OFF\_\_\_\_\_

{

PortFButtonPushed = PortFButtonPushed;

if (PortFButtonPushed && CheckStateOfOnboardSwitch(BUTTON1)) //if button state is the same after 2 second

{

CurrentState = 1; //becomes red state

CountSeconds\_For2Seconds = 0; //reset

PortFButtonPushed = 0;

}

}

while (CurrentState ==1) //\_\_\_\_\_State\_1\_STOP\_Red\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(RED);

if (PortAButtonPushed && CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A)) //if button state is the same after 2 second

{

CurrentState = 3; //becomes yellow state

CountSeconds\_For2Seconds = 0; //reset

CountSeconds\_For5Seconds = 0; //reset

PortAButtonPushed = 0;

}

if (CountSeconds\_For5Seconds >= 5)

{

CurrentState = 2; //becomes green state

CountSeconds\_For2Seconds = 0; //reset

CountSeconds\_For5Seconds = 0; //reset

PortAButtonPushed = 0;

}

}

while (CurrentState ==2) //\_\_\_\_\_State\_2\_GO\_Green\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(GREEN);

if(PortAButtonPushed && CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A)) //if button state is the same after 2 second

{

CurrentState = 3; //becomes yellow state

CountSeconds\_For2Seconds = 0; //reset

CountSeconds\_For5Seconds = 0; //reset

PortAButtonPushed = 0;

}

if (CountSeconds\_For5Seconds >= 5)

{

CurrentState = 1; //becomes red state

CountSeconds\_For2Seconds = 0; //reset

CountSeconds\_For5Seconds = 0; //reset

PortAButtonPushed = 0;

}

}

while (CurrentState ==3) //\_\_\_\_\_State\_3\_WARN\_Yellow\_\_\_\_\_

{

Turn\_ON\_peripheral\_LED\_By\_Color(YELLOW);

if (CountSeconds\_For5Seconds >= 5)

{

CurrentState = 2; //becomes green state

CountSeconds\_For2Seconds = 0; //reset

CountSeconds\_For5Seconds = 0; //reset

PortAButtonPushed = 0;

}

}

}

}

//Timer OA Interupt //this is the Time\_Handler\_0A to use in conjunction with SectionLab2B3B() (the traffic controller)... and should be commented out otherwise

//void Timer\_Handler\_OA(){

// CountSeconds\_For2Seconds++; //update the 2 second timer count

// CountSeconds\_For5Seconds++; //update the 5 second timer count

// Pointer\_for\_Updating\_GPTMICR = 0x01; //Clear the flag // a "1" to this register resets "GPTMRIS" bit 0 to "0"

//}

////PortA Interrupt

//void GPIO\_PORT\_A\_Handler(){

// PortAButtonPushed = 1; //update a variable that button has been pushed

// Pointer\_to\_Update\_GPIO\_IC\_On\_A |= 0x3F;//reset the interupt.. interrupt clear 0b111111;

//}

////PortF Interrupt

//void GPIO\_PORT\_F\_Handler(){

// PortFButtonPushed = 1; //update a variable that button has been pushed

// Pointer\_to\_Update\_GPIO\_IC\_On\_F |= 0x3F;//reset the interupt.. interrupt clear 0b111111;

//}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void SectionLab2B4(){ //program to use onboard buttons to turn on eith blinking blue or steady red

Turn\_OFF\_OnBoard\_LED();

Turn\_OFF\_peripheral\_LED();

//Pointer\_for\_Updating\_GPTMCTL |= 0x01; //UPDATE TAEN Bits0 on GPTMCTL with "0x01" and thereby ENABLE timer

Pointer\_for\_Updating\_GPTMCTL &= ~0x01; /\*disable timer interupt here: \*/

/\*reset GPIOFATA ON F Bits 1:0 to ")"\*/ Pointer\_for\_Updating\_GPIODATA\_OnF &= ~0x11;

while (1)

{

//processes are being done by the Timer handler and GPIO handler

}

}

void Timer\_Handler\_OA(){ //Timer OA Interupt //this is the Time\_Handler\_0A to use in conjunction with SectionLab2B4() and should be commented out otherwise

timerflag = timerflag\*-1;

Pointer\_for\_Updating\_GPTMICR = 0x01; //Clear the flag // a "1" to this register resets "GPTMRIS" bit 0 to "0"

if (timerflag == 1) { Pointer\_for\_Updating\_GPIODATA\_OnF |= BLUE; }

if (timerflag == -1) { Pointer\_for\_Updating\_GPIODATA\_OnF &= ~BLUE; } }

void GPIO\_PORT\_F\_Handler(){ //GPIO PortF Interrupt

int buttondata;

buttondata = 0;

buttondata = Pointer\_for\_Updating\_GPIODATA\_OnF;

/\*reset the interupt.. interrupt clear 0b10001; \*/ Pointer\_to\_Update\_GPIO\_IC\_On\_F |= 0xFF;

/\*disable timer interupt here: \*/ Pointer\_for\_Updating\_GPTMCTL &= ~0x01;

/\*"GPTM Control" - "0" at bit zero, which is named "TAEN", "0" disables the timer,

and "1" enables the timer and begins the counting or the capture logic is enabled

based on the GPTMCFG register. \*/

if ((buttondata & 0x01) == 0x01) //PF0 SW2

{ sw1pushed = 0; sw2pushed = 1; Pointer\_for\_Updating\_GPIODATA\_OnF &= ~0x11;}

if ((buttondata & 0x10) == 0x10) //PF4 SW1

{ sw1pushed = 1; sw2pushed = 0; Pointer\_for\_Updating\_GPIODATA\_OnF &= ~0x11;}

if(sw1pushed) //SW1 PF4 should turn on RED LIGHT

{

Pointer\_for\_Updating\_GPIODATA\_OnF &= ~BLUE;

Pointer\_for\_Updating\_GPIODATA\_OnF |= RED;

}

if(sw2pushed) //SW2 PF4 should turn on Flashing Blue Light

{

Pointer\_for\_Updating\_GPIODATA\_OnF &= ~RED;

/\*enable time interupt here:\*/ Pointer\_for\_Updating\_GPTMCTL |= 0x01; //UPDATE TAEN Bits0 on GPTMCTL with "0x01"

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\* callable functions and initializations \*\*\*\*//

int CheckStateOfOnboardSwitch(int switchnumber){ // input is in form of this example "BUTTON1" or "0x10"

int ButtonStateA = 0;

int ButtonStateB = 0;

ButtonStateA = Pointer\_for\_Updating\_GPIODATA\_OnF; // can be checked at memory space: 0x400253FC

ButtonStateA = ButtonStateA & switchnumber;

if (ButtonStateA == 0x10)

{ButtonStateA = 0;}

else if (ButtonStateA == 0x01)

{ButtonStateA = 0;}

else if (ButtonStateA == 0x00)

{ButtonStateA = 1;}

if(CountSeconds\_For2Seconds >= 2){ //if button has been pressed for 2 seconds

ButtonStateB = Pointer\_for\_Updating\_GPIODATA\_OnF; // can be checked at memory space: 0x400253FC

ButtonStateB = ButtonStateB & switchnumber;

if (ButtonStateB == 0x10)

{ButtonStateB = 0;}

else if (ButtonStateB == 0x01)

{ButtonStateB = 0;}

else if (ButtonStateB == 0x00)

{ButtonStateB = 1;}

}

return ((ButtonStateA == ButtonStateB)&&(ButtonStateA ==1));

}

int CheckStateOfPERIPHERALSwitch(int switchnumber){ // input is in form of this example "PERIPHERAL\_BUTTON\_A" or "0x02" (for PA2)

int ButtonStateA = 0;

int ButtonStateB = 0;

ButtonStateA = Pointer\_for\_Updating\_GPIODATA\_OnA; // can be checked at memory space: 0x400253FC

ButtonStateA = ButtonStateA & switchnumber;

if (ButtonStateA == 0x10)

{ButtonStateA = 0;}

else if (ButtonStateA == 0x01)

{ButtonStateA = 0;}

else if (ButtonStateA == 0x00)

{ButtonStateA = 1;}

if(CountSeconds\_For2Seconds >= 2) //if button has been pressed for 2 seconds

{

ButtonStateB = Pointer\_for\_Updating\_GPIODATA\_OnA; // can be checked at memory space: 0x400253FC

ButtonStateB = ButtonStateB & switchnumber;

if (ButtonStateB == 0x10)

{ButtonStateB = 0;}

else if (ButtonStateB == 0x01)

{ButtonStateB = 0;}

else if (ButtonStateB == 0x00)

{ButtonStateB = 1;}

}

return ((ButtonStateA == ButtonStateB)&&(ButtonStateA ==1));

}

void Turn\_ON\_OnBoard\_LED\_By\_Color(int color){

//Pointer\_for\_Updating\_GPIODATA\_OnF = 0x0;

Pointer\_for\_Updating\_GPIODATA\_OnF = color;

}

void Turn\_OFF\_OnBoard\_LED(){

Pointer\_for\_Updating\_GPIODATA\_OnF = 0x0;

}

void Turn\_ON\_peripheral\_LED\_By\_Color(int color) {

if (color == RED)

{

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x80; //Turn on PA7 (bit 7)

}

else if (color == YELLOW)

{

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x40; //Turn on PA6 (bit 6)

}

else if (color == GREEN)

{

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x10; //Turn on PA4 (bit 4)

}

}

void Turn\_OFF\_peripheral\_LED(){

Pointer\_for\_Updating\_GPIODATA\_OnA &= 0x0; //Turn OFF PA4, PA6, PA7

}

void DelayByLoops(int delayloops){

for (int i = 0; i <= delayloops; i++) {

for (int j = 1; j <= 1000; j++) {

}

}

}

void DelayByTimerOneSecond(){

int timerflag = 0;

do{

timerflag = Pointer\_for\_Updating\_GPTMRIS; //monitor/poll the flag

}while (timerflag ==0);

Pointer\_for\_Updating\_GPTMRIS = 0x00; //reading zero into this doesn't actually set it back to zero... instead do this...

Pointer\_for\_Updating\_GPTMICR = 0x01; // a "1" to this register resets "GPTMRIS" bit 0 to "0"

}

int DelayByTimerXSeconds(int DelaySeconds){ //input is limited to positive intergers

for (int i=0; i<=DelaySeconds; i++)

{

DelayByTimerOneSecond();

}

return (1);

}

void InitializePortFRegisters() {

// General...

Pointer\_for\_Updating\_RCGCGPIO\_OnPortsX = 0x21; //enable RCGCGPIO Clock gating at 0x400FE608 bit 5 and 0 turned to "1" with "0x21" or ("100001") ("FEDCBA") using a pointer

//Port F Onboard lights and switches...

Pointer\_for\_Updating\_GPIODEN\_OnF = 0x1F; //enable digital Pins on Port F at 0x4002551C // "11111"

Pointer\_for\_Updating\_GPIODIR\_OnF = 0x0E; //set Port F pins 0 and 4 as "input" at 0x40025400 //"01110"

Pointer\_for\_Updating\_GPIOLOCK\_OnF = 0x4C4F434B; //to program button PF0 a value of 0x4C4F434B must be set at 0x40025520 bits 31:0 will unlock it so that it may be modified //"00000000"

Pointer\_for\_Updating\_GPIOCR\_OnF = 0x11; //must be "1" on bits 7:0 to allow bits to be written to GPIOPUR // "10001"

Pointer\_for\_Updating\_GPIOPUR\_OnF = 0x11; //This is the Pull-up register, and bits 7:0 can be on with "1" to enable pull-up // "10001"

Pointer\_for\_Updating\_GPIODATA\_OnF = 0x00; //reset to zero at start

}

void InitializePortARegisters() { //LEDs are on A5, A6, A7, and Button is on A2

//Port A I/O for Peripheral Buttons and Switches...

Pointer\_for\_Updating\_GPIOAMSEL\_R\_OnA = 0x00; // 0x04 ? // Analog mode select ("0" -> digital, "1"-> Analog)

Pointer\_for\_Updating\_GPIODEN\_OnA = 0xFF; //enable as digital with "1" Enable PortPin A2, A5, A6, A7

Pointer\_for\_Updating\_GPIODIR\_OnA = 0xF0; //"DIR is DIRECTION (in="0"/out="1") [oxFO = "1111 0000"

Pointer\_for\_Updating\_GPIOPCTL\_R\_OnA = ~0xFF0F0F00; // This is PortControl on a MUX Pins 76\_4\_\_2\_\_ are enabled via "0"

Pointer\_for\_Updating\_GPIOAFSEL\_OnA &= 0x00; //"Alternative Function Select"… "0" means set to Digital function and not an alternative function

Pointer\_for\_Updating\_GPIODATA\_OnA = 0x00; //reset to zero at start

}

void InitializeRegisterValuesForTimerAndInterupts() {

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TIMER INITIALIZATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*"Register Clock Gating Timer" "1" will Enable with and provide a clock to

16/32-bit general-purpose timer module 5 in Run mode. This should be done on

Bit 1 for Timer 0\*/

Pointer\_for\_Updating\_RCGTIMER |= 0x01;

/\*"GPTM Control" - "0" at bit zero, which is named "TAEN", "0" disables the

timer, and "1" enables the timer and begins the counting or the capture logic

is enabled based on the GPTMCFG register.\*/

Pointer\_for\_Updating\_GPTMCTL &= ~0x01;

/\*"GPTM Configuration", Lab Directive says set "=0x00000000;" on all eight

bits, This zero value is to select a 32-bit configuration.", Spec Sheet Says,

just set bits 0,1,2 to "0"... I could use "&=~07" \*/

Pointer\_for\_Updating\_GPTMCFG &=~07;

//"Timer A Mode Register"... see do the following:

//TAMR: bits1:0 (0x1 for 1-shot, 0x2 for periodic),

Pointer\_for\_Updating\_GPTMTAMR |=0x02; //for bit 1 to "1"(part of TAMR field)

Pointer\_for\_Updating\_GPTMTAMR &=~0x01; //for bit 0 to "0" (part of TAMR field)

//TACDIR: Direction counting up/down with bit4 ("1" is upward, "0" is downward)

Pointer\_for\_Updating\_GPTMTAMR &=~0x20; //for bit 4 to "0" (TACDIR field)

/\* "GPTM Timer A Interval Load", initially set to 16,000,000 or "0xF42400" \*/

Pointer\_for\_Updating\_GPTMTAILR =0xF42400; /

//"Interupt Mask Remover" - needed in Lab2PartB

Pointer\_for\_Updating\_GPTMIMR |= 0x1; //set up the needed interupt(s)...

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INTERRUPT INITIALIZATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*"GPIO Interupt Sense" "0" for edge, "1" for level Bits 7:0 should be "0" on

needed pin \*/

Pointer\_for\_Updating\_GPIOIS\_On\_F &= ~0x11; // bits PF0 and PF4... |=0b01110: |= 0x11;

Pointer\_for\_Updating\_GPIOIS\_On\_A &= ~0x0C; //bits PA2 and PA3... |= 0b10011; |=0x0C;

/\*"GPIO Interupt Both Edges" should be "0" to allow interupt generation to be

controlled by GPIOIEV. Should be "1" to allow interupt on both edges of the

coresponding pin to trigger input. \*/

Pointer\_for\_Updating\_GPIOIBE\_On\_F &=~0x11; // "0" on bits PF0 and PF4

Pointer\_for\_Updating\_GPIOIBE\_On\_A &=~0x0C; // "0" on bits PA2 and PA3

/\*"GPIO Interput Event" "0" for the falling edge or low level on a cooresponding

pin. "1" for the rising or highlevel on the corresponding pin... to trigger

an interupt. \*/

Pointer\_for\_Updating\_GPIOIEV\_On\_F &= ~0x11; // "1" on bits PF0 and PF4

Pointer\_for\_Updating\_GPIOIEV\_On\_A &= ~0x0C; // "1" on bits PA2 and PA3

/\*"GPIO Interrupt Mask" "0" means that the interrupt from the corresponding pin is masked,

"1" means that the interrupt from the corresponding pin is sent to the interrupt controller. \*/

Pointer\_for\_Updating\_GPIOIM\_On\_F |= 0x11; // bits PF0 and PF4

Pointer\_for\_Updating\_GPIOIM\_On\_A |= 0x0C; // bits PA2 and PA3

/\*to disable interupts you reset them at memory location 0xE000.E000 offset 0x180

and for this purpose pin 19.\*/

// Pointer\_to\_disable\_interupts =\_\_\_\_; //not needed now

/\* reset interrupt flags \*/

Pointer\_for\_Updating\_GPTMICR = 0x01; //a "1" to this register resets "GPTMRIS" bit 0 to "0"

Pointer\_to\_Update\_GPIO\_IC\_On\_A |= 0x3F;

Pointer\_to\_Update\_GPIO\_IC\_On\_F |= 0x3F;

/\*Priority level: interupt level must be less than BASEPRI. //BASE 0xE000.E000 offset 0x400, 0x410, 0x41C;

interupt priority 19 is at 0x410 and should be set to: less than BASEPRI\*/

/\*#0 priority of onboard buttons on PortF //sitting on vector table at

interupt #0, and bits 7:5 here "0b100000"\*/

Pointer\_to\_Update\_Priority0\_Int0\_3 |= (3<<5);

/\*#19 priority of timer\_handler //sitting on vector table at

interupt #19, and bits 31:29 here\*/

Pointer\_to\_Update\_Priority4\_Int16\_19 |= (3<<29);

/\*#30 priority of peripheral buttons on PortA //sitting on vector table at

interupt #30, and bits 23:21 here\*/

Pointer\_to\_Update\_Priority7\_Int28\_31 |= (3<<21);

/\*enable interupts globally on I-bit of the PRIMASK (Priority mask); BASE AT 0x0000.0000 bit 0

should be a "1" to prevent the activation of all exceptions with configurable priority. "0" for

no effect. This register is only accessible in "privledged mode." \*/

//Pointer\_to\_Update\_PRIMASK =\_\_\_\_;

//pointers for part LAB2 Part B Interupts

/\*enable interupt at NVIC Page 141-142 pf user manual

Base 0xE000E000 with offeset of 0x100 for

EN0 = 0xE000E1000 = >bit 19 turned on "0" means that interupt is disabled, and "1" means that

interupt is enabled. Note: to disable interupts you reset them at memory location 0xE000.E000

offset pin 0x180\*/

Pointer\_for\_Updating\_NVIC = 0x0;

Pointer\_for\_Updating\_NVIC |=(1<<19);

Pointer\_for\_Updating\_NVIC |= (1<<0); // for PORTA /\* set the 1st bit to "1"

Pointer\_for\_Updating\_NVIC |= (1<<30);// for PORTF "1" followed by 30 zeros

/\*"GPTM Control" - "0" at bit zero, which is named "TAEN", "0" disables the timer,

and "1" enables the timer and begins the counting or the capture logic is enabled

based on the GPTMCFG register.\*/

Pointer\_for\_Updating\_GPTMCTL |= 0x01; //UPDATE TAEN Bits0 on GPTMCTL with "0x01"

}