# EE474 Introduction To Embedded Systems

Professor Dr Rania Hussein

Summer 2018

Student Jeff Josephsen

# Lab 1: Input/Output Interfacing and Traffic Light Controller, FSMs

Date 6-27-2018

Abstract:

Student gained familiarity with IAR workbench software development environment useful in programming the EK-TM4C123 board. While learning basics of general-purpose input and output (GPIO) he performing simple digital input (interfacing push button) and output (interfacing LED). The student gained an understanding the power of pointers in C and how to use them in addressing GPIO registers. Perusal of specifications sheets and datasheet to extract information was necessary for addressing the LaunchPad components. Lastly, a design and implementation of a finite state machine (FSM) traffic light controller was completed.

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. IDE installation guide ([https://canvas.uw.edu/courses/1205180/files/folder/Ek-](https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49716060)  [TM4C123GXL?preview=49716060](https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49716060))
5. Tutorial video (<https://www.youtube.com/watch?v=cLbaSFKdAho>)

Since we use different versions of the IDE in this video, please consult the above IDE installation guide if you find any differences.

1. Oscilloscope and some cables
2. LEDs (<https://learn.adafruit.com/all-about-leds/the-led-datasheet>)
3. Push buttons (<https://www.alps.com/prod/info/E/HTML/Tact/SnapIn/SKHH/SKHHAKA010.html>)
4. Debouncing (<https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49719211>)
5. Lab write up specifications (<https://canvas.uw.edu/courses/1205180/files/folder/labs?preview=49165850>)

Software program line coding was implemented within and IDE suite. The Information provided to the student included the following:

1. IAR workbench is installed on the lab machines in the lab and the installation instructions (<https://canvas.uw.edu/courses/1205180/files/folder/Ek-TM4C123GXL?preview=49716060>) are posted on Canvas. If you want to download the software on your laptop please note that IAR works only on Windows OS so if your computer runs Linux or Mac OS, you will need to install a virtual machine. You are free to use other IDEs such as Keil uVision or Code Composer but make sure that the TM4C123 is supported (the board is supported by Keil 4.7 but not Keil 5).
2. The free version of IAR limits your data and code to 32 KB, which is not an issue for the lab assignments in this course.

Completed and complied software programs were uploaded, or embedded, to a TM4C123GXL board. The following information, including figures 1,2 and 3 below were presented to the student in preparation of software line coding.

[Figure 1](#_bookmark0) 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.

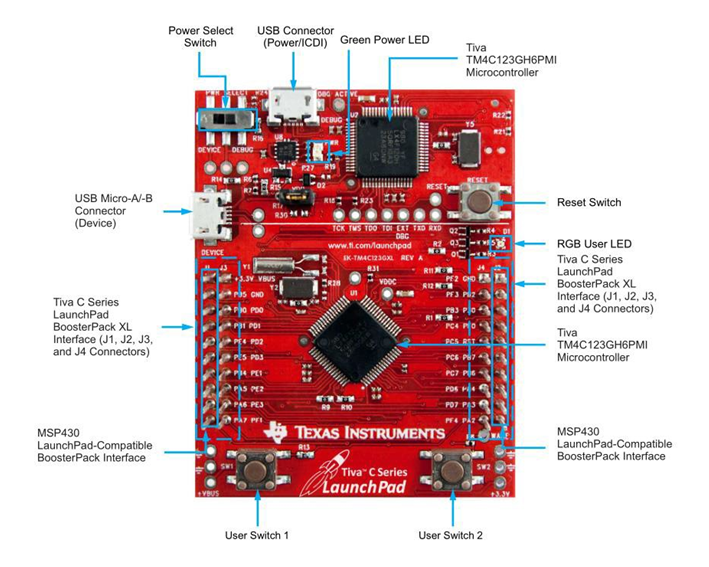


Figure 1 Tiva C Series LaunchPad Components

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](#_bookmark1) and [Figure 3](#_bookmark2))

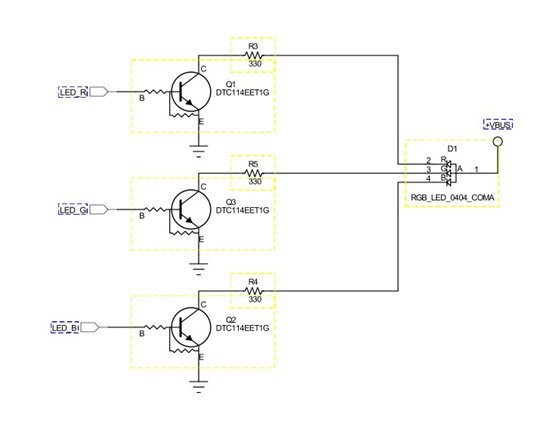


Figure 2 Onboard LED Components

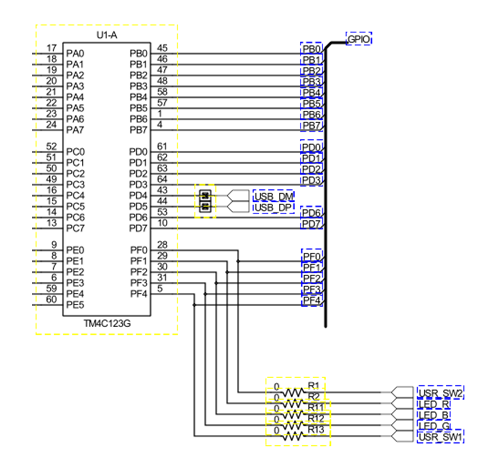


Figure 3 - Port F Pinout

The purpose of this lab work included teach the student to create software pointers. Then use those pointers to update registers in the bit level of the ARM processors memory. Those methods were them utilized to implement switches and LEDs both on the development board and peripheral to the development board. Lastly, The students were asked to design a finite state machine traffic light controller and implement it in code and embedding the development board.

Procedure:

The Lab came in two parts. In Part A of the lab the student was given the proper registers that would need to be enabled through pointer variables to turn on an onboard LED. These registers included the following. The RCGCGPO register; which is used to enable a port to listen to the onboard clock. Next the GPIODEN. This is used to digitally enable pins on a port. By default the mode of all pins on the port would be analog. For this lab some pins will be used for digital purposes. Next, it was necessary to update values in the GPIODIR register to an ON value so that those pins would be set to output rather than input. This was required on pins 1, 2, and 3 of Port F.

The same registers required update values for the implementation of push-button-switch inputs. Also, other registers needed to have their bits updated for the inputs. There was a GPIOLOCK, which when given a specific 32 bit code would unlock the GPIOCR register for updating. It would only remain unlocked until a value was read from the board. Then the GPIOCR register, a commit register, needed to be de-committed on the bits wherewith the push-button-switch(es)would connect. Lastly, for these buttons a register named GPIOPUR, which is the pull-up register was needed both for calisthenics and for pulling the bit values up to true unless brought down by a button depressment.

As side note, something should be said about "pointers." To update values on these bits it was necessary to use a coding tool called a pointer. A pointer is a type of variable, in this case a volatile unsigned 32 bit integer, which has been denoted with a "\*" between the variable type and variable name in the variable declaration. This pointer variable could then be given a value, and that value would need to be, for usefulness, the 32 bit location in memory where the pointer was designed to point. In our case, multiple pointers were created to point at the registers mentioned above. The pointers were then given a dereferencing "\*" just before there variable name was invoked. This dereferencing was a way of changing the actual values of the bits in the memory which was being pointed to. In the end the tool looked something like this:

#DEFINE POINTERVARIABLENAME (\* ((VARIABLETYPE \*) VALUE\_A))

In this pseudo example then "VALUE\_A" is the value held by POINTERVARIABLENAME and is set to be the value denoting the place in the in the memory such as 0x40005000.

Here is a specific example created in this lab by the student:

#define Pointer\_for\_Updating\_RCGCGPIO (\*((volatile uint32\_t \*) 0x400FE608))

With this tool called the Pointer, the student was able to update values in the memory simply by invoking the name of the pointer variable and then assigning to it, by "=", an updated number. Here's an example of what that might look like:

Pointer\_for\_Updating\_RCGCGPIO = 0x21;

This line places into the memory at the location where the pointer is pointing (0x400FE608) the value 0x21. This is a hexidecimal value and could also be recalculated in binary as 0b00000000 00000000 00000000 00100001 or just 0b100001.

Pointer were used throughout the lab to update, via derefernced pointers, the states of bits in the registers being pointed to.

At this point the student was asked to download the IAR Embedded Workbench IDE software to a laptop computer and prepare the software with the correct configurations. This was to cause the student to begin to gain familiarity with the software.

Next, the student was asked to begin writing line code within an IDE workspace and a C-based project instantiation. The goal of the first section of the lab was learning to both use the hardware to output data to light an LED and input data from a butting-switch. The code written by the student can be found at the end of the report.

An investigation was made into the delay between the depressment of the onboard button-switch and the lighting of the LED. In figure \_\_\_ below it can be seen that the time passage between the ppint where the button signal reaches 50% of its final voltage and the time when the LED signal reaches 50% of its final voltage is measured to be 6 ns.

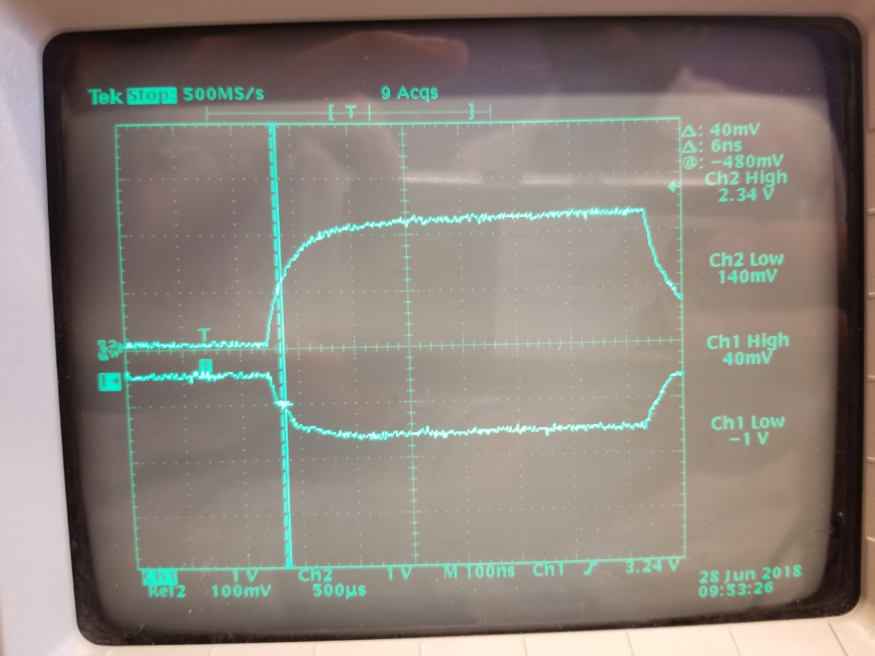
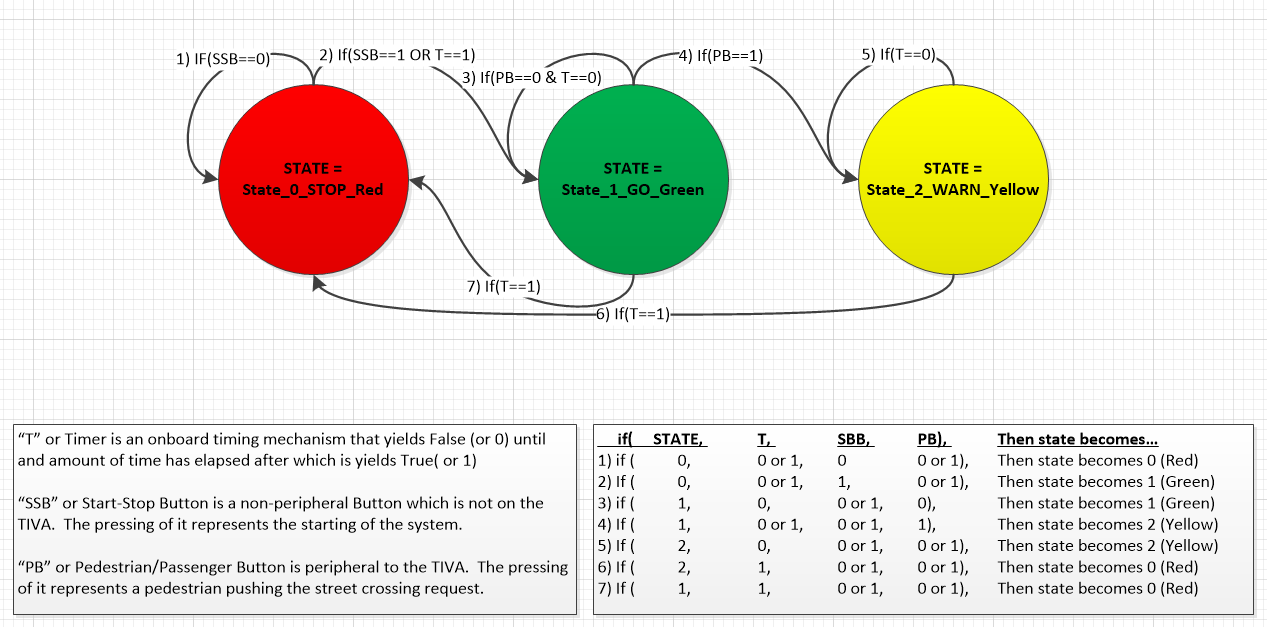


Figure 4 Oscilloscope output showing measured delay

Next, In the second part of the lab, Part B, the same code principles were used to now take input from a peripheral button, and then with the use of programming structures light a peripheral LED. In connecting the switch and LED it was important that by using a breadboard, IC 7406, and shunting capacitors the LED and board were protected from overload damage.

After this point, wherein the student was able to function the onboard and off board switches and LEDs, the student was asked to design an finite state machine of a traffic light controller system. They system can be seen in the following figure \_\_\_\_



As can be seen in the image, the Finite State Machine was designed to have three states, and to progress through those states with the use of pedestrian walk buttons, start/stop buttons, and for-loop based delays.

This Finite State Machine was then implemented in software upon the development board hardware. The code utilized while-loops, for loops, enumeration data types, register updates through pointer instantiations. The code written by the student can be found at the end of this report.

The hardware implementation of the Finite State Machine was derived from a lab instruction based design. It was updated to allow for another button and two more LEDS. Figure \_\_\_\_ shows the derived form of the hardware circuit implementation of the Finite State Machine.

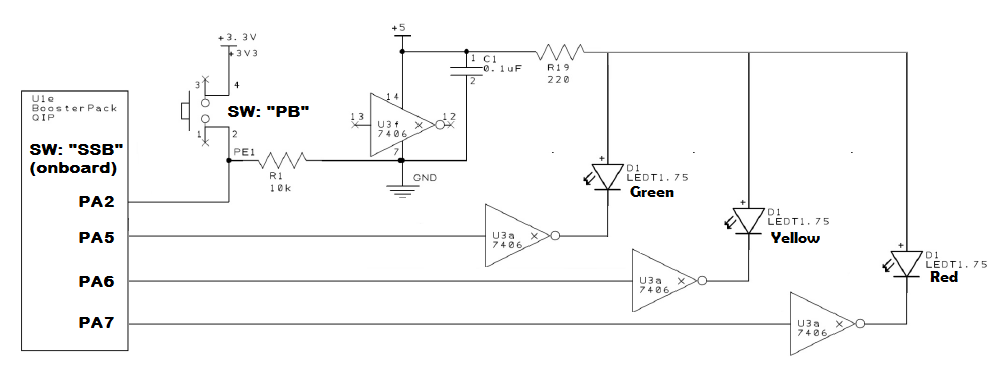


Figure Circuitry for hardware implementation of system

It can be seen in the image that the LEDS are being driven by current that is also run through a 220 Ohm resistor and through a not gate. These additional components were included to set and limit the levels of current flowing through the LEDs and also to limit the amount of current going into pins PA5, PA6, and PA7 of the development board. The capacitor C1 in the figure also is used to shunt the power across the NOT-GATE IC and into the development board.

Results:

The required designs, and implementation in both software and hardware were completed by the student. The student did not experience debouncing issues and needed not to implement denouncing code. The systems were able to be completed to operate as expected.

The student did have questions in the final implementation of the states and transitions between states in the finite state machine. Clarification between passenger and pedestrian, stage and state, whether or not a time delay should be used to move from state "stop" to state "go", and whether the system should ever move from state "go" to state "stop" without first passing through state "warn" would have been helpful.

The student spent more than 20 hours on this lab. In the laboratory he was told by two student that they considered dropping the course for the difficulty of the lab. In my personal perspective I think that the ramping up into a first use of the IDE and the introduction to pointers, registers, and instantiating register pointers vs updating the values of register pointers seemed to be a lot to get used to in a week. None of those issues were as time consuming as the little things like a single incorrect bit in a register address or value. The lesson of the lesson seems to be to always look twice at anything that doesn't feel normal, because it probably hides a mistake in the details.

Conclusion:

The student was able to understand and utilize the software and hardware tools required by the Lab*.* Of special significance was the use of pointers in updating the registers of the GPIO. Implementing code with the use of the IDE coding environment brought clarity to the coding process. It was specifically helpful to learn to monitor the memory and variables while stepping through the line code during code debugging. Designing and implementing a system that is commonly found in a real life application was especially encouraging as the student could see how the processes and tools might be implements in the future.

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:

Lab 1 Part A

// program that turns on the red LED ... updated by student JJosephsen

//#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"

// I was able to find this pointer creation process both in the tmc...h header and in the video at https://www.youtube.com/watch?v=cc16ShyiUG8 near time 10:01

#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

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()

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 SectionA2(); //example program modified to work without TIVA header

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

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

int main()

{

// Uncomment the Selection to compile and upload to board

//SectionA2(); //example program modified to work without TIVA header

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

SectionA5(); //program to incorporate switches

return 0;

}

void SectionA2() //exmaple program modified to work without TIVA header

{

//SYSCTL\_RCGC2\_R = SYSCTL\_RCGC2\_GPIOF; //enable Port F GPIO

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

//GPIO\_PORTF\_DIR\_R = color; //set Port F as output // example code from lab1 directive

Pointer\_for\_Updating\_GPIODIR = RED; //instead, now using the newly defined pointer above

//GPIO\_PORTF\_DEN\_R = color; //enable digital Port F // example code from lab1 directive

Pointer\_for\_Updating\_GPIODEN = RED; //instead, now using the newly defined pointer above

//GPIO\_PORTF\_DATA\_R = 0; // clear all Port F // example code from lab1 directive

Pointer\_for\_Updating\_GPIODATA = 0; //instead, now using the newly defined pointer above

//GPIO\_PORTF\_DATA\_R = color; //turn on the red LED // example code from lab1 directive

Pointer\_for\_Updating\_GPIODATA = RED; //instead, now using the newly defined pointer above

}

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(GREEN);

}

else

{

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

}

}

}

void Turn\_On\_Color(int color)

{

//SYSCTL\_RCGC2\_R = SYSCTL\_RCGC2\_GPIOF; //enable Port F GPIO

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

//GPIO\_PORTF\_DIR\_R = color; //set Port F as output // example code from lab1 directive

Pointer\_for\_Updating\_GPIODIR = color; //instead, now using the newly defined pointer above

//GPIO\_PORTF\_DEN\_R = color; //enable digital Port F // example code from lab1 directive

Pointer\_for\_Updating\_GPIODEN = color; //instead, now using the newly defined pointer above

//GPIO\_PORTF\_DATA\_R = 0; // clear all Port F // example code from lab1 directive

Pointer\_for\_Updating\_GPIODATA = 0x0; //instead, now using the newly defined pointer above

//GPIO\_PORTF\_DATA\_R = color; //turn on the red LED // example code from lab1 directive

Pointer\_for\_Updating\_GPIODATA = color; //instead, now using the newly defined pointer above

}

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);

}

Lab 1 Part B

// program for EE474 Lab1 Part B by student JJosephsen

#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))

// 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

int delaytime = 6000;

void Delay(int delaytime); // 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\_STOP\_Red, State\_1\_GO\_Green, State\_2\_WARN\_Yellow};

int CurrentTimerState = 0;

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

// 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

enum system\_state\_data\_type CurrentState;

CurrentState = 0;

CurrentTimerState = 0;

Turn\_OFF\_peripheral\_LED();

while(1)

{

while (CurrentState ==0) //RED/Stopped State

{

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

if (CheckStateOfOnboardSwitch(BUTTON1))

{

Turn\_OFF\_peripheral\_LED();

CurrentState = 1; // State\_1\_GO\_Green

}

}

while (CurrentState ==1) //Green/Go

{

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

if(CheckStateOfPERIPHERALSwitch(PERIPHERAL\_BUTTON\_A))

{

Turn\_OFF\_peripheral\_LED();

CurrentState = 2;

}

}

while (CurrentState ==2) //Yellow/Warn State

{

CurrentTimerState = 0;

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

Delay(delaytime);

CurrentTimerState = 1;

if (CurrentTimerState == 1)

{

Turn\_OFF\_peripheral\_LED();

CurrentState = 0;

}

}

}

}

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

int state = 0;

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

state = state & switchnumber;

if (state == 0x10)

{state = 0;}

else if (state == 0x01)

{state = 0;}

else if (state == 0x00)

{state = 1;}

return (state);

}

int CheckStateOfPERIPHERALSwitch(int switchnumber){ // input is in form of this example "PERIPHERAL\_BUTTON\_A" or "0x04" (for PA2)

int State;

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

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

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

if (State == switchnumber)

{State = 1;}

else if (State == 0x00)

{State = 0;}

return (State);

}

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(){

EnablePortFRegisters();

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 = 0x20; //Turn on PA5 (bit 5)

}

}

void Turn\_OFF\_peripheral\_LED(){

Pointer\_for\_Updating\_GPIODATA\_OnA &= 0x1F; //Turn OFF PA5, PA6, PA7 - why does the gree light pop on?

}

void Delay(int delaytime){

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

{

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

{}

}

}

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 = 0xE4; //enable as digital with "1" Enable PortPin A2, A5, A6, A7

Pointer\_for\_Updating\_GPIODIR\_OnA = 0xE0; //"DIR is DIRECTION (in="0"/out="1")

Pointer\_for\_Updating\_GPIOPCTL\_R\_OnA = ~0xFFF00F00; // This is PortControl on a MUX Pins 765\_\_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

}