TI ARM Lab 5
API and Interrupts

National Science Foundation
Funded in part, by a grant from the National Science Foundation
DUE 1068182

Acknowledgements
Developed by Craig Kief and Brian Zufelt from the COSMIAC Research Center at the University of New Mexico. Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. Originally Funded by the National Science Foundation.

Lab Summary
This lab introduces the concepts of the API and interrupts.

Lab Goal
The goal of this lab is to continue to build upon the skills learned from previous labs. One of the strongest powers that exist for microcontrollers is their ability to remain in a low power state for extended periods of time and then when an input occurs, to be able to wake up and do work.

Each of these labs add upon the previous labs and it is the intention of the authors that the students will build (with each lab) a better understanding of the ARM processor and basic C code. Even though these tutorials assume the student has not entered with a knowledge of C code, it is the desire that by the time the student completes the entire series of tutorials that they will have a sufficient knowledge of C code so as to be able to accomplish useful projects on the ARM processor.

It is assumed that the "student" will be installing the packages in accordance with the instructions in Lab1. That process will provide the framework and load all the support files/projects for the associated workshop. Each Lab will have two files. For example, there is a generally a Lab x project and a Lab x Solution project. The only difference between these two projects is the code in the main.c file. The exception is in the interrupt project (Lab 5) where the …startup_ccs.c file is also edited in the solution. The reason this was done this way was to provide the student with two choices. They can follow the tutorial in the Lab and type in the associated code identified in the tutorial. This will allow for errors to be made and learning to be achieved through the debugging process. The alternative is that the student has a fall back option if everything goes bad and they can just look at/copy the source file contents from the "Solution" projects.

Learning Objectives
The student should begin to become familiar with the concept of APIs and interrupts. API, an abbreviation of ***a****pplication* ***p****rogram* ***i****nterface*, is a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer then puts the blocks together. In our case, we would refer to them as a set of functions.

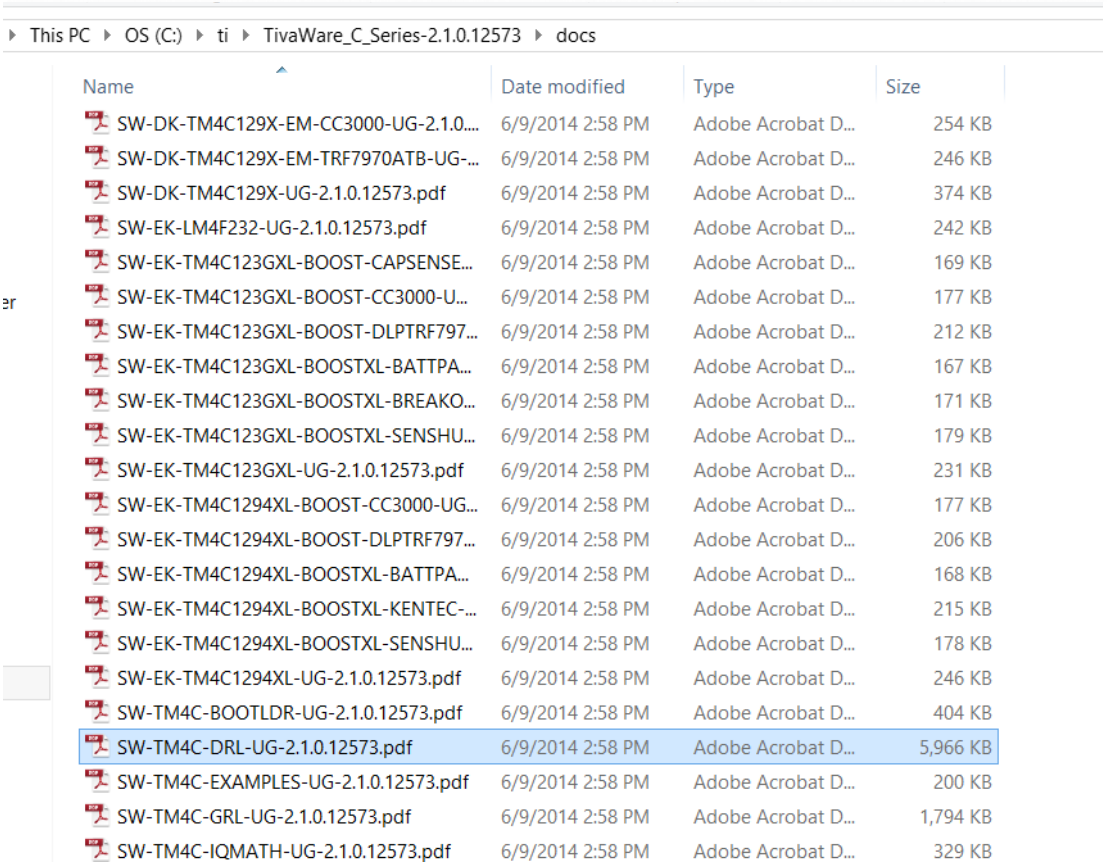

Figure 1. 700 Page Users Guide

In the documentation of the installation you performed, there is a 6G pdf file called the TivaWare™ Peripheral Driver Library User's Guide. This 700 page document contains a complete list of these functions and how to use them. As an example, you can search for GPIOPinWrite. These functions are contained on Read Only Memory (ROM) contained permanently on the microcontroller. The beauty of this is that you never have to work hard to find these API functions. They are always on the microcontroller chip. The beauty of the API functions are that they reduce the amount of resources used thus the amount of power consumed. They are optimized for the specific chip. This is very important for low power applications. For example, in previous labs we used a “for” loop for creating a time delay. In fact, there is a delay function in the ROM that is called SysCtlDelay. To use this function, all you do is to call the function and pass in an input integer of how long you want to delay.

The next important section covered in this lab is the concept of the interrupt. Brian always uses the analogy of the automobile airbag deployment system. In all the projects we have done prior to this, we have always allowed the system to flow from top to bottom. This is done to allow for a smooth and predictable flow of the program. Unfortunately, if you are using this ARM processor in your automobile to control the airbag system and you have an accident, you really don’t want to wait until the currently running routine finishes. You want it to immediately stop what it is doing and activate the airbag NOW. This is done via a concept called an interrupt. You will be using an Interrupt Service Routine (ISR) to provide that immediate halting capability.

Time Required
Approximately 2 hours

Lab Preparation

It is highly recommended that the student read through this procedure once before actually using it was a tutorial. By understanding the final goal it will be easier to use this as a tutorial. This Lab will introduce the Tiva C Series TM4C123G LaunchPad Evaluation Kit which is a low-cost evaluation platform for ARM® Cortex™-M4F-based microcontrollers from Texas Instruments. The design of the TM4C123G LaunchPad highlights the TM4C123GH6PM microcontroller with a USB 2.0 device interface and hibernation module.

The EK-TM4C123GXL evaluation kit also features programmable user buttons and a Red, Green, RGB LED for custom applications. The stackable headers of the Tiva C Series TM4C123G LaunchPad BoosterPack XL Interface make it easy and simple to expand the functionality of the TM4C123G LaunchPad when interfacing to other peripherals with Texas Instruments' MCU BoosterPacks.

Equipment and Materials
Access to the four software packages: Code Composer, Tivaware, Driver file and ATE Workshop installer are required and should have been installed in accordance with Lab 1. In addition, the user should have access to the Tiva evaluation kit (EK-TM4C123GXL) and (for Labs 3-9) the Digilent Orbit board. It is assumed that the student has already completed Lab 1 and the software is installed.

Software needed	Quantity
The installation files are covered in Lab 1. The software package can be downloaded as one large zip file from this URL: http://cosmiac.org/thrust-areas/education-and-workforce-development/community-portal/	1
Hardware needed	**Quantity**
The hardware required is the Tiva LaunchPad Kit. The kit and the ORBIT daughter card can be purchased from the Digilent Corporation www.digilentinc.com	1

Additional References

This page is for general information: http://www.ti.com/tool/ek-tm4c123gxl on the Tiva LauchPad Kit. If the link above is no longer valid, just google the Tiva LaunchPad and it will pop up. The full set of tutorials is available on this website: http://cosmiac.org/thrust-areas/education-and-workforce-development/microcontrollers/ate-developed-material/

Lab Procedure : Install/Connect board to computer

Plug in the supplied USB cable to the top of the Evaluation Kit and Digilent Orbit board as shown in Figure 2. Ensure the switch on the board is set to “DEBUG” and not “DEVICE”. It is assumed that the evaluation board is properly installed and operating. Launch the Code Composer software.

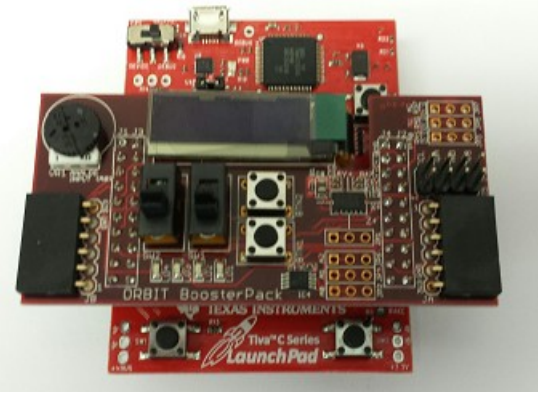

Figure 2. Board Assembly and Attachment

If the auto installer was used, it will automatically load up shells that are/were used for the workshop. The student will know if they have used the auto installer program as they will be presented with a picture as shown in Figure 3 where the shells of all lab projects are installed. There are two ways to proceed. One is to just go to the Solution and run it. The other is to continue with typing the code into the Lab 5 main.c and startup files.

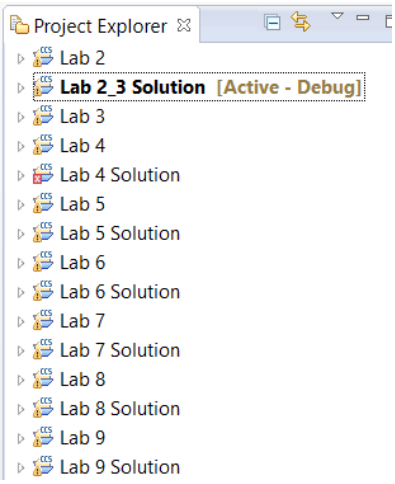

Figure 3. Project Options

Figure 3 shows the shell of the projects. This has created the shell for all the tutorials. Expand the ALAB5 project and open the main.c file.

```
13
14 ****************************************
15 Chip type               : ARM TM4C123GH6PM
16 Program type            : Firmware
17 Core Clock frequency    : 80.000000 MHz
18 ****************************************
19
20 #include <stdbool.h>
21 #include <stdint.h>
22 #include "inc/tm4c123gh6pm.h"
23 #include "inc/hw_memmap.h"
24 #include "inc/hw_types.h"
25 #include "driverlib/gpio.h"
26 #include "driverlib/pin_map.h"
27 #include "driverlib/sysctl.h"
28 #include "driverlib/uart.h"
29 #include "inc/hw_i2c.h"
30 #include "driverlib/i2c.h"
31 #include "inc/hw_ints.h"
32 #include "driverlib/interrupt.h"
33 #include "driverlib/timer.h"
34
```

Figure 4. Lab 5 Expanded

As can be seen in Figure 4, the header files have already been integrated into the project. This is the nice feature of the installation package that was created for the workshops but that can be also used for future projects with these two boards. The starting code should be shown. It is also available at the end of the tutorial as Attachment 1.

This tutorial starts out where Lab 4 ended. As is always the case, first in main.c is the header files. Always use the same nine shown here. Secondly, it is necessary to enable the ports and then turn on the specific I/O pins that will be needed. Lab 4 was designed to blink lights and was triggered by (and stopped by) pushing a button. However, in Lab 4, the breaks were done without interrupts. This means that the only time that the program switches over to blinking lights is when the running routine is complete. We start out with the same code as Lab 4 but change the delays to slow it down for visualization. This starts out the same as where we ended in Lab 4 but we are going to add the interrupts for Lab 5.

Code should be as shown in attachment 1 for a starting point. As is always the case, click on the bug and then on the green triangle. The lights should be blinking slower due to added time in sysctldelay. Press and hold button 1. The designer can see how it must finish cycle to escape normal program. Do this enough times to be confident that you fully understand when the system breaks away to start blinking lights.

Hit red square to cancel the debugger and return to CCS edit mode.

In main.c, add the following code right before the “while (1) loop”

```
//-----------------------------------------
// Setting up interrupts - makes BTN 1 the interrupt
```

```
        IntEnable(INT_GPIOD_TM4C123); //sets the interrupt controller to interrupt on GPIO port D
        GPIOPinIntEnable(GPIO_PORTD_BASE, GPIO_PIN_2); // BTN 1
        GPIOIntTypeSet(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_RISING_EDGE); // Rising Edge
        IntMasterEnable();  // Turns on all interrupts

    //-----------------------------------------------
```

At bottom of main.c (before the final "}"), type/paste this in:

```
void GPIO_PORTD_isr(void){

    GPIOPinIntClear(GPIO_PORTD_BASE, GPIO_PIN_2 ); // Clear Interrupts

    while(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_2)){     // Listen for BTN 1

        // All LED Blink
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF);
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xFF); // White Output

        SysCtlDelay(2000000);

        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00);
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);
        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00); // White Output

        SysCtlDelay(2000000);
    }
}
```

What we have done is add in the ISR function. It is the exact same code as above but now is acted upon by a push of the button immediately instead of after the finish of the routine.

Go back up the main.c code and delete this unnecessary code that is now accomplished via the ISR. For historical purposes, it can also be commented out by putting a "/*" at the start of the block and a "*/" at the end of the block.

```
while(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_2)){     // Listen for BTN 1

    // All LED Blink
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF);
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xFF); // White Output

    SysCtlDelay(2000000);

    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00);
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00); // White Output

    SysCtlDelay(2000000);
        }
```

We need to tell the Interrupt Controller were to go to find the ISR. Go into the startup_ccs.c program. Take extreme care when editing this section. Errors created in this section will cause an error at boot up and will take a long time to find and correct.

This first change declares the function for the compiler but doesn't tell it that it is an ISR. The fact that ISR is in the function name is for human benefit. As shown in Figure 5, type/paste in the following code shown below into the startup_ccs.c file right below the extern void _c_int00(void); function call.

extern void GPIO_PORTD_isr(void);

```
19
20 //*****************************************
21 //
22 // External declaration for the reset
23 // processor is started
24 //
25 //*****************************************
26 extern void _c_int00(void);
27
28 extern void GPIO_PORTD_isr(void);
29
```

Figure 5. Insert ISR Function Call

The line of code below defines what is called to in the vector table. It defines what code will handle the interrupt. Where it is pasted is what is important. It is at a specific location in the table.

GPIO_PORTD_isr, // GPIO Port D

```
62     IntDefaultHandler,              // The PendSV handler
63     IntDefaultHandler,              // The SysTick handler
64     IntDefaultHandler,              // GPIO Port A
65     IntDefaultHandler,              // GPIO Port B
66     IntDefaultHandler,              // GPIO Port C
67        GPIO_PORTD_isr,              // GPIO Port D
68     IntDefaultHandler,              // GPIO Port E
69     IntDefaultHandler,              // UART0 Rx and Tx
```

Figure 6. ISR Code

Copy/type the code shown above into the location shown in Figure 6. Take care to place this in the correct location. The indentation is shown only for visualization purposes. Save all files and click on the bug and green triangle. If everything else goes to heck and you find errors you can't resolve, don't hesitate to copy all of the main.c solution and just paste it into the project.

Immediately when the button one is pressed, it jumps to the interrupt. The lights immediately begin to blink. This is what happens when you have to immediately stop what is going on and jump for an error. When button one is released, it goes immediately back to where it came from.

When done, remember to hit the red square to exit debug mode.

Challenge: Change from Button 1 to Switch 2

Attachment 1: main.c file before addition of interrupt

```
/*****************************************************

Project : Orbit Lab 5 ATE (int)
Version : 2.0
Date    : 2/20/2015
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
This source introduces the concept of interrupts. Building
on Lab 4, the call to blink the LEDs will execute as soon
as BTN 1 is pressed


*****************************************************
Chip type               : ARM TM4C123GH6PM
Program type            : Firmware
Core Clock frequency : 80.000000 MHz
*****************************************************/

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"


void main(void) {

        // Setting the internal clock
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

        // Enable Peripheral ports for input/ output
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);  //PORTC
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);  //PORTB
        GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5); // LED 4

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);  //PORT D
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6); // LED 3
        GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_2); // BTN 1

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);  //PORT F
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); // RGB LED on Launchpad

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);  //PORT A
        GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_6); // Switch 2


//-----------------------------------------


//-----------------------------------------

        while(1)
```

```
        {
//              if(GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_6)){    // Listen for the switch, Removed to run code in complete loop

                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4); // LED Blue on Launchpad

                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); // LED 1 on LED 2 Off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 OFF, LED 2 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); //LED 1 off LED 2 off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP



//              else {

                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 8); //  LED Green on Launchpad

                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); // LED 1 off LED 2 Off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 OFF, LED 2 on
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); // LED 1 on LED 2 Off

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP


                while(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_2)){     // Listen for BTN 1

                        // All LED Blink
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF);
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20);
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xFF); // White Output

                        SysCtlDelay(2000000);

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00);
```

```
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00); // White Output

                SysCtlDelay(2000000);
        }


    }
}
```

Attachment 2: main.c solution file

```
/*****************************************************

Project : Orbit Lab 5 ATE (int)
Version : 2.0
Date    : 2/20/2015
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
This source introduces the concept of interrupts. Building
on Lab 4, the call to blink the LEDs will execute as soon
as BTN 1 is pressed


*****************************************************
Chip type               : ARM TM4C123GH6PM
Program type            : Firmware
Core Clock frequency : 80.000000 MHz
*****************************************************/

#include <stdbool.h>
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"

// Defines not needed to be placed in lab
//#define Lab5_Problem
#define Lab5_Solution
// -------------------------------------

void main(void) {

        // Setting the internal clock
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

        // Enable Peripheral ports for input/ output
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);  //PORTC
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);  //PORTB
        GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5); // LED 4

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);  //PORT D
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6); // LED 3
        GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_2); // BTN 1

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);  //PORT F
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); // RGB LED on Launchpad

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);  //PORT A
        GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_6); // Switch 2


//-----------------------------------------
```

```
#ifdef Lab5_Solution
        IntEnable(INT_GPIOD_TM4C123);
        GPIOIntEnable(GPIO_PORTD_BASE, GPIO_PIN_2);
        GPIOIntTypeSet(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_RISING_EDGE);
        IntMasterEnable();
#endif
//-----------------------------------------

        while(1)
        {
//              if(GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_6)){    // Listen for the switch, Removed to run code in complete loop

                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4); // LED Blue on Launchpad

                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); // LED 1 on LED 2 Off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 OFF, LED 2 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); //LED 1 off LED 2 off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP



//              else {

                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 8); //  LED Green on Launchpad

                        // Cycle through the LEDs on the Orbit board
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); // LED 1 off LED 2 Off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 OFF, LED 2 on
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); // LED 1 on LED 2 Off

                        SysCtlDelay(6000000); // Delay, Replaces for LOOP


#ifdef Lab5_Problem
                        while(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_2)){     // Listen for BTN 1
```

```
                        // All LED Blink
                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF);
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20);
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xFF); // White Output

                        SysCtlDelay(2000000);

                        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00);
                        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);
                        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00); // White Output

                        SysCtlDelay(2000000);
                }
#endif

        }

}

void GPIO_PORTD_isr(void){

        GPIOIntClear(GPIO_PORTD_BASE, GPIO_PIN_2 ); // Clear Interrupts

        while(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_2)){     // Listen for BTN 1

                // All LED Blink
                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xFF);
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0xFF);
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20);
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xFF); // White Output

                SysCtlDelay(2000000);

                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00);
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00);
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00); // White Output

                SysCtlDelay(2000000);
        }
}
```

Attachment 3: Block Diagram of the Pins Used in Projects