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, Brian Zufelt, and Jacy Bitsoie at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. *Funded by the National Science Foundation (NSF).*

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. This lab helps the student to continue to gain new skills and insight on the C code syntax and how it is used in the TI implementation of the ARM processor. Each of these labs add upon the previous labs and it is the intention of the authors that the student 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.

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. In the documentation of the installation you performed, there is a 30M pdf file called the LM4F120H5QR ROM User's Guide. This 330 page document contains a complete list of these functions and how to use them. These functions are contained on Read Only Memory (ROM) contained on the microcontroller. The beauty if 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. 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 of an interrupt. You will be using an Interrupt Service Routine (ISR) to provide that immediate halting capability.

Grading Criteria
N/A

Time Required
Approximately one hour

Lab Preparation
It is highly recommended that the student read through this procedure once before actually using it as a tutorial. By understanding the final goal, it will be easier to use this tutorial as a learning guide. The other tricky part is related to how your software was loaded. While this series of tutorials was being developed, a new way to load the software (and preload the tutorials) was developed. This new method simplifies the linker and compiler options so that you no longer have to do this process as was shown in tutorial 4.

Equipment and Materials
Access to Stellaris LM4F120 LaunchPad software and evaluation kit (EK-LM4F-120XL). It is assumed that the student has already completed Lab 4 and the software is installed properly. It is also assumed that the board is plugged in and the power is applied

Software needed	Quantity
Download Stellaris LaunchPad™ software from the TI website http://www.ti.com/tool/SW-EK-LM4F120XL and the lab Installer from http://www.cosmiac.org/Microcontrollers.html	1
Hardware needed	Quantity
The hardware required is the TI Stellaris LaunchPad Kit and the Digilent Orbit board	1

Additional References
Current TI ARM manuals found on TI web site: www.ti.com/lit/ug/spmu289a/spmu289a.pdf .

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 1. 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. The easiest way to find it, is to go to Start, All programs and then look in the area identified in Figure 2.

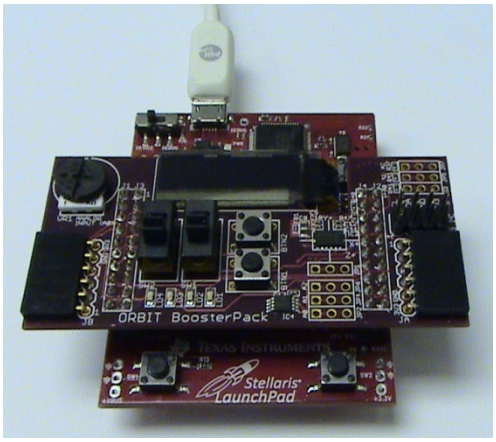

Figure 1. Board Assembly and Attachment

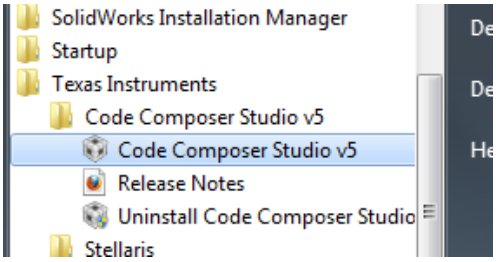

Figure 2. Code Composer Launch

When presented with the picture shown in Figure 2, it might be easier to right click on the **v5** logo and then choose to create a shortcut for the desktop to launch quicker.

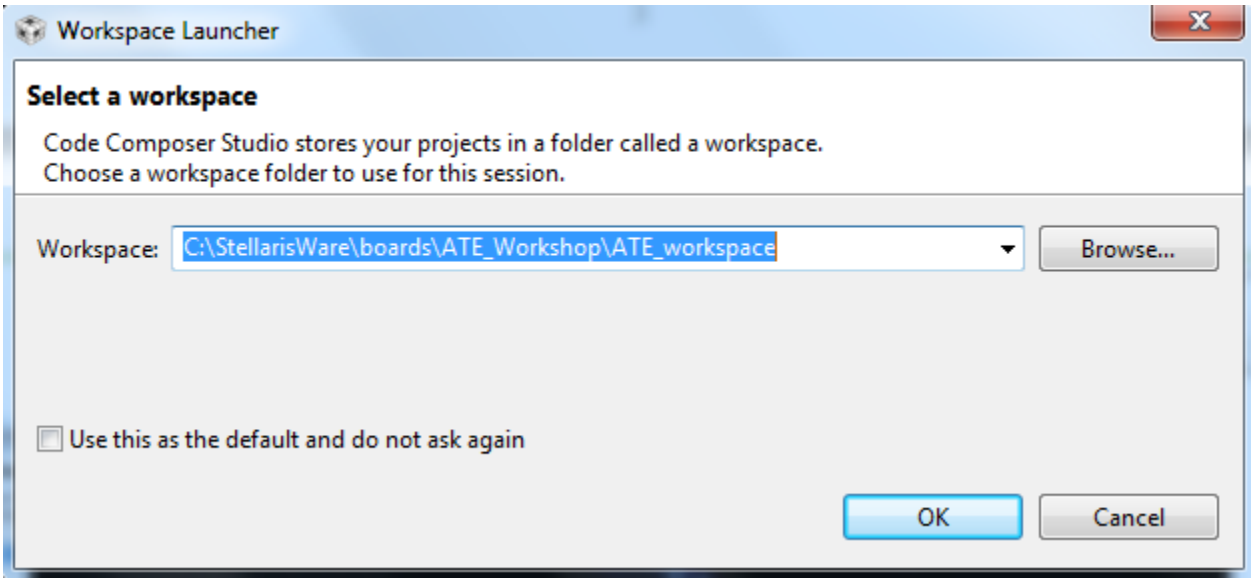

Figure 3. Workspace Launcher

The designer will be presented with the choices shown in Figure 3. The Code Composer wants to know where the workspace is located. If the auto installer was used, it will automatically load up shells that were used for the workshop. In that case, use the path as shown in the picture in Figure 3. The student will know if they have used the auto installer program as they will be presented with a picture as shown in Figure 4 where the shells of all lab projects are installed. Be aware though that there are no "solutions" folders. These were only created to assist with the internal tutorial development.

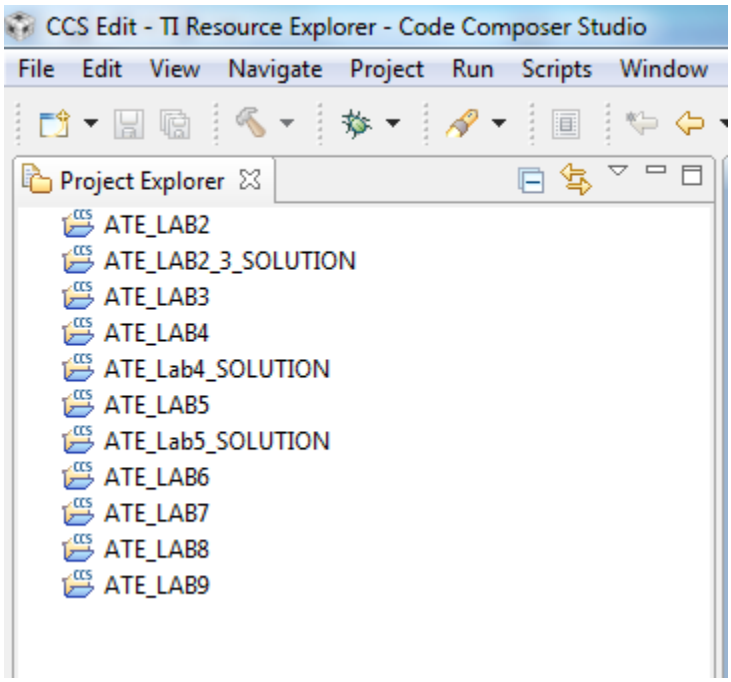

Figure 4. Shell of Projects

This has created the shell for all the tutorials. Expand the ATE_LAB5 project (not the solution).

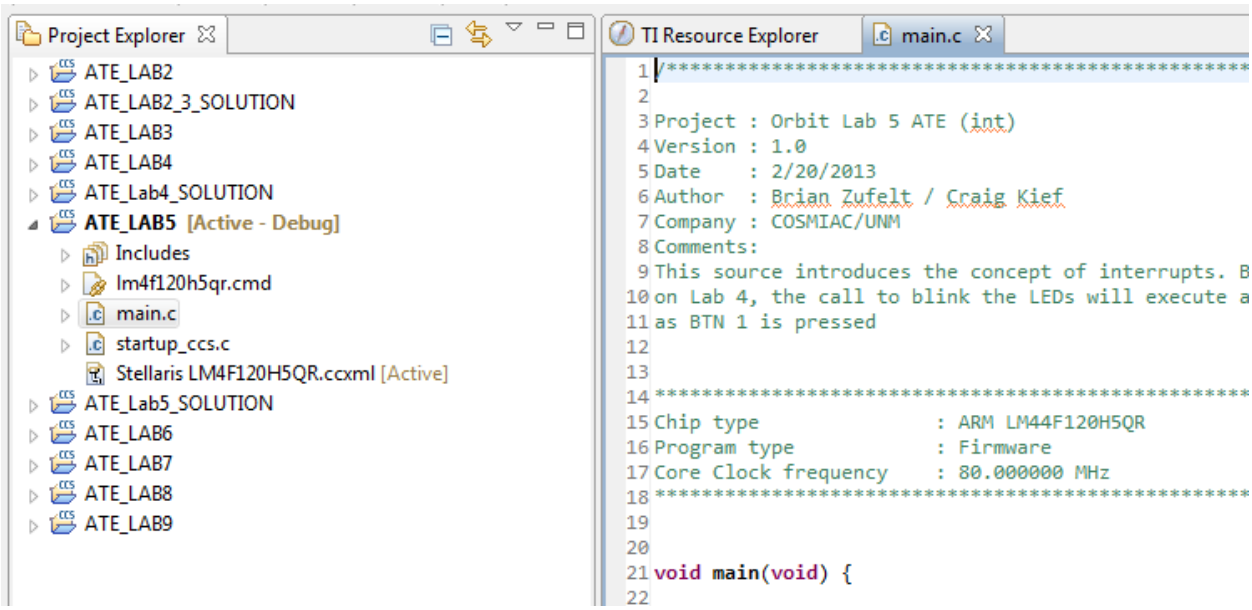

Figure 5. Lab 5 Expanded

As can be seen in Figure 5, the startup_ccs.c file has 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 tutorial 4 ended. Tutorial 4 was designed to blink lights and was triggered by to be stopped by pushing a button. However, in four, the breaks were done without interrupts. We start out with the same code but change it. This starts out the same as where we ended in Lab 4 but we are going to add the interrupts.. The only change between tutorial 4 and this one is that we increased the delay time (slowed down the sequence for effect).

Code should be as shown in attachment 1.

Run it. Lights blinking slower due to added time in sysctldelay. When blue, hit button 1. The designer can see how it must finish cycle to escape normal program. 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);  //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 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. 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.

```
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 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.

```
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 6. Debug Icon

This input defines what is called to as a 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 7. Debug Icon

Immediately when the button is pressed, it jumps to the interrupt. When released, it goes immediately back to where it came from.

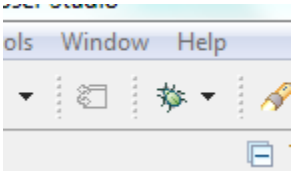

Figure 8. Debug Icon

Go to debug

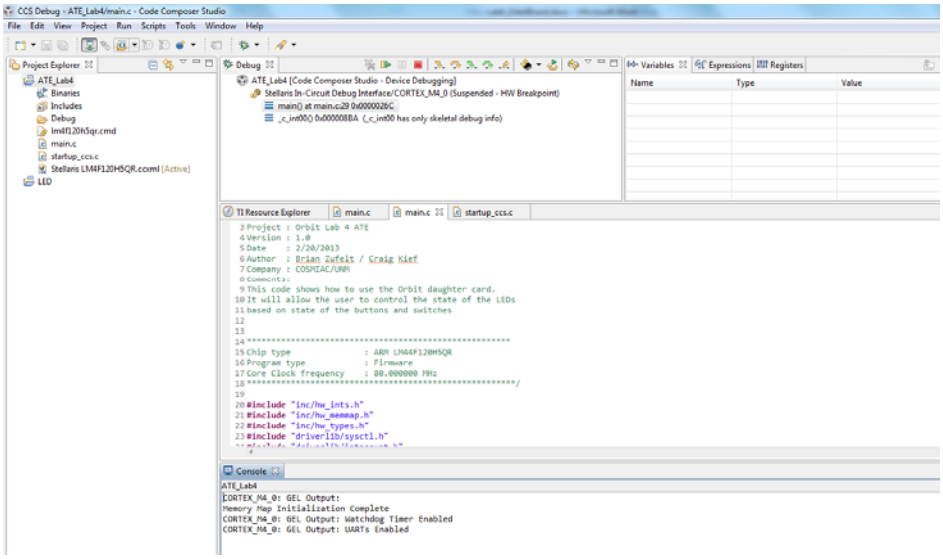

Figure 9. Screen Sample

Rearrange desktop view to look like this.

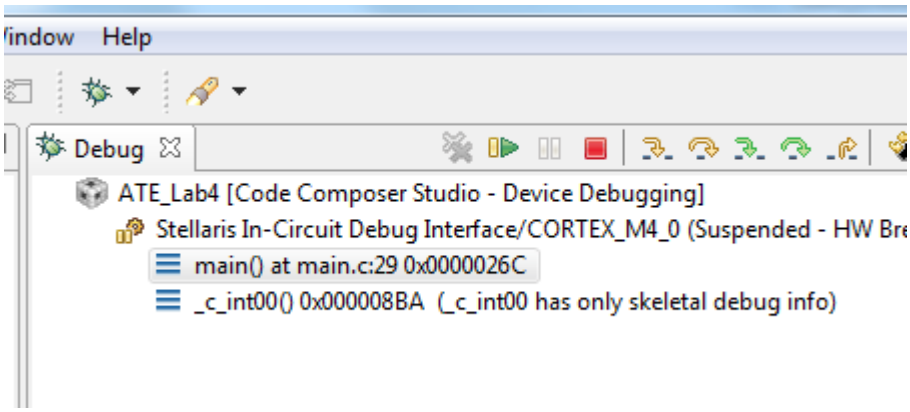

Figure 10. Run Project File

Run or resume to make it dance. 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

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

Project : Orbit Lab 5 ATE (int)
Version : 1.0
Date    : 2/20/2013
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 LM44F120H5QR
Program type            : Firmware
Core Clock frequency      : 80.000000 MHz
**************************************************/

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.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 1: Block Diagram of the Pins Used in Projects