

## Lab 3: TM4C123 Microcontroller I/O Ports

### Switches and LED Interfaces

Xiaorong Zhang

---

#### Preparation

- Review the content of lecture 5, 6
- Before you start the lab, make sure you have completed Lab 1 and installed Keil uVision, TI's TivaWare Development Kit, and windows drivers for the LaunchPad on the computer. You will need the actual physical LaunchPad to complete this lab.

#### References

- Getting Started with the Tiva TM4C123G LaunchPad Workshop Student Guide and Lab Manual (iLearn -> Reference Materials -> TM4C123G\_LaunchPad\_Workshop\_Workbook.pdf)
- TivaWare Peripheral Driver Library User's Guide (iLearn-> Reference Materials -> SW-TM4C-DRL-UG-2.1.0.12573.pdf)
- Tiva TM4C123GH6PM Microcontroller Data Sheet (iLearn-> Reference Materials)

#### Purpose

The general purpose of this lab is to familiarize you with the TM4C123 microcontroller and develop C programs to configure and access the parallel ports on the microcontroller by using the TivaWare Peripheral Driver Library. Software skills you will learn include flow chart, bit-masking operations, toggling, if-then, looping, delay, and writing friendly software.

#### Demonstration and Submission

You will have one week to complete the lab. You can discuss with your group members and complete the lab work together. Every group will need to write and submit a lab report to iLearn->Labs->Lab3 report submission. The lab report should include

- The students' names, emails and IDs
- The answers of all the questions asked in the lab handout
- Discussion and suggestions: Through your lab experiments, what have you learned? Do you have any suggestions for future labs or improvement on the learning experiences?

If you finish the lab experiments during the lab time, please demonstrate your results to the instructor. The instructor may ask questions regarding your program. After the demonstration, you can leave. **The latest demonstration time will be the beginning of next lab. The lab report is due at 6pm on the day of your next lab.**

Again, you can work with your group members on all the lab activities, but make sure you understand all the materials.

### **Procedure**

This lab provides a step-by-step tutorial which will familiarize you with the Keil  $\mu$ Vision integrated development environment (IDE) and teach you how to create a new project, compile, download, and run the project on the launchpad. The tutorial provides two examples on how to initialize and configure the GPIO ports on the microcontroller to interface with the switches and LEDs on the launchpad. You will also learn how to use the functions provided in the TivaWare Peripheral Driver Library to simplify the development process.

After you go through the tutorial, please answer all the questions listed at the end of the handout.

## Tutorial: Getting Started with Keil $\mu$ Vision; Two Examples

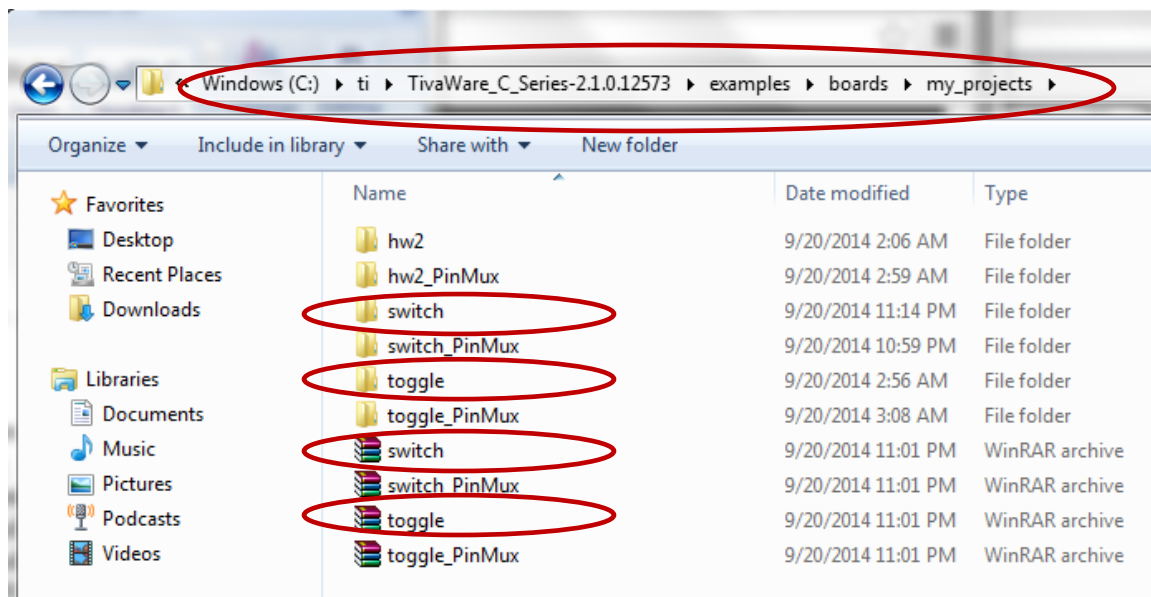
### Procedure

Plug in the LaunchPad into a USB port of your computer. Turn the LaunchPad power on. Make sure you have installed all the software tools and drivers listed in the “Preparation” section of this document.

#### A. Start the Keil $\mu$ Vision IDE and Run the Sample Projects

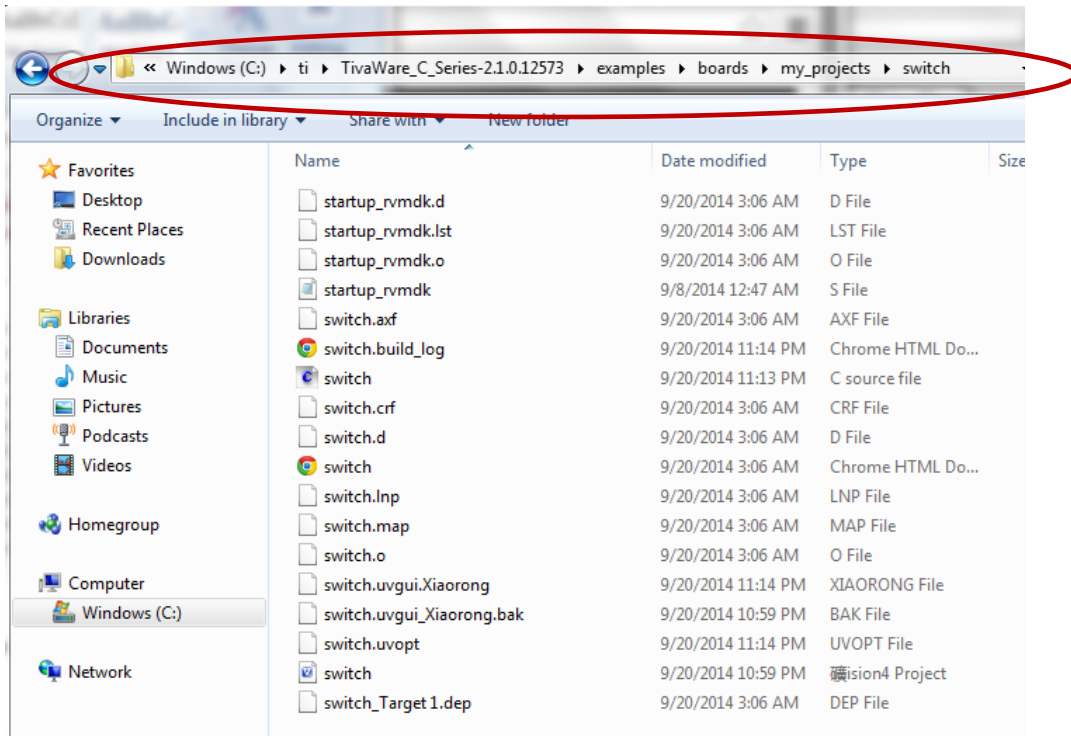
1. Download the sample projects from iLearn (iLearn->Labs->sample\_projects). There are two sample projects: *switch* and *toggle*. *switch* develops a system which interfaces with the blue LED and the switch SW2 on the launchPad. When SW2 is pressed, the LED is turned on. When SW2 is released, the LED is turned off. *toggle* implements a system that toggles the on-board red LED repeatedly.

2. The downloaded sample projects must be saved in a specific directory in order to run correctly. It is assumed you already install the TivaWare library in C:\ti\TivaWare\_C\_Series-2.x.x.xxxxx. First create a new directory “my\_projects” (you may have already created it in Lab 2) under C:\ti\TivaWare\_C\_Series-2.x.x.xxxxx \examples\boards and then save and decompress the sample projects under “C:\ti\TivaWare\_C\_Series-2.x.x.xxxxx\examples\boards\my\_projects” as shown in Figure 1.

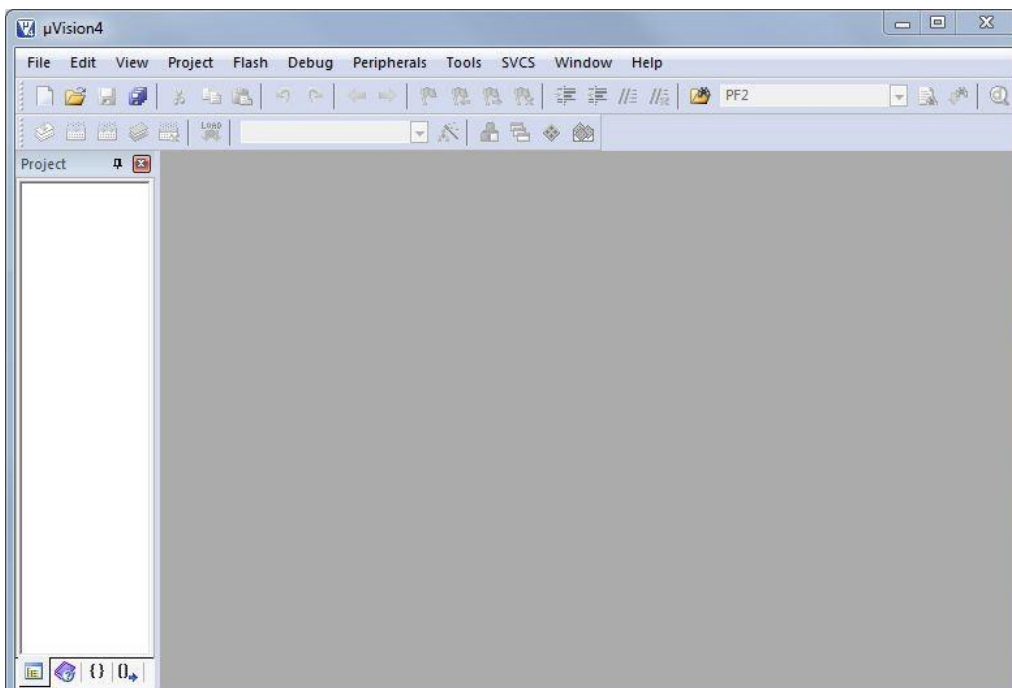


**Figure 1**

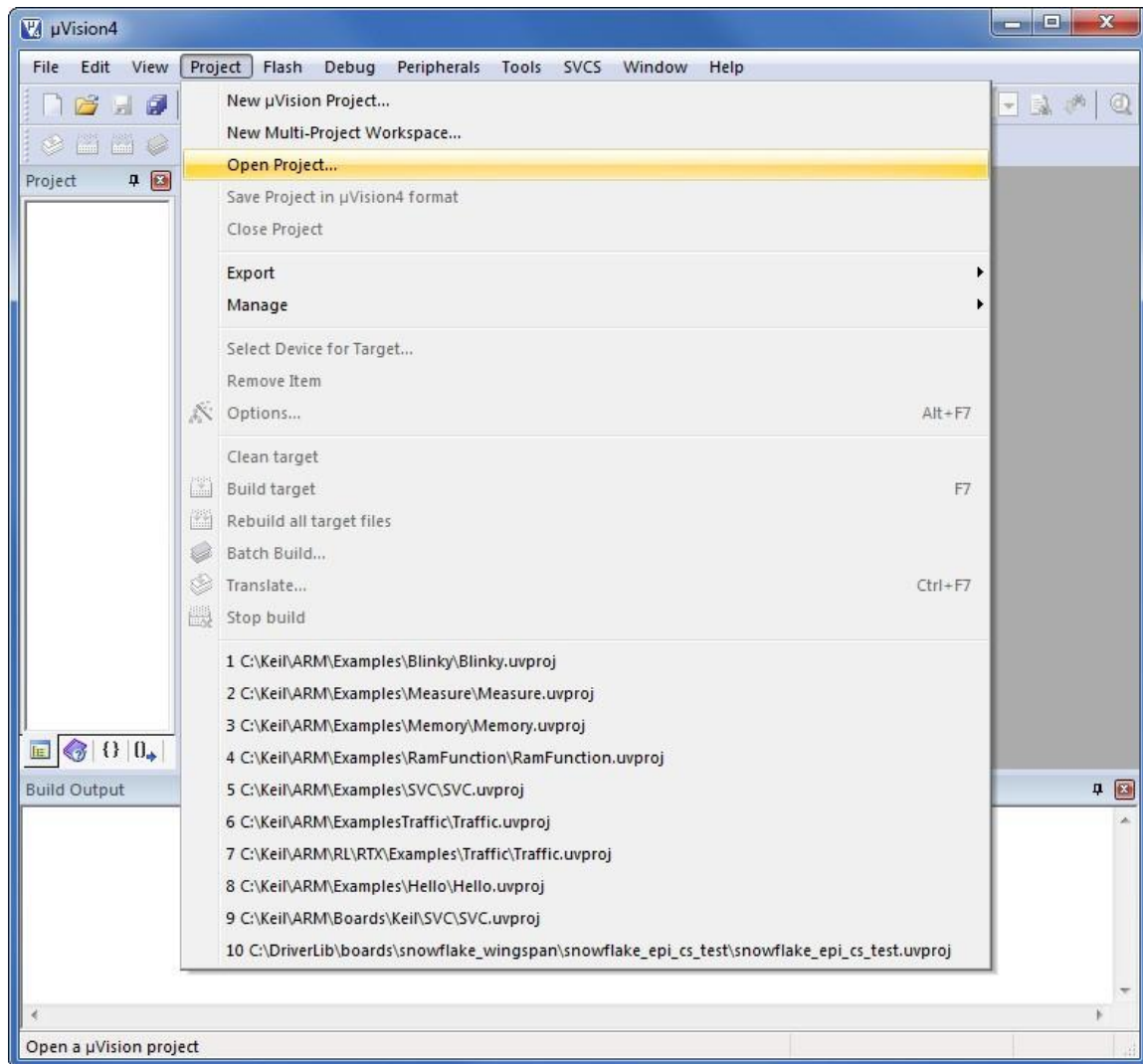
Again, make sure you save them in the correct directory. If you double click the *switch* folder, it will look something like this as shown in Figure 2. Make sure the directory is C:\ti\TivaWare\_C\_Series-2.x.x.xxxxx\examples\boards\my\_projects\switch\, not C:\ti\TivaWare\_C\_Series-2.x.x.xxxxx\examples\boards\my\_projects\switch\switch\.

**Figure 2**

3. Start the Keil  $\mu$ Vision IDE by double-clicking the icon on your desktop or by selecting it from the Windows Start Menu. When the IDE loads, it was a blank screen (see Figure 3).

**Figure 3**

4. From the Project menu, select Open Project... (See Figure 4)



**Figure 4**

5. Use the dialog box to navigate to the *switch* project. From the location where you installed TivaWare, the *switch* project should be located in: `C:\ti\TivaWare_C_Series-2.x.x.xxxx\examples\boards\my_projects\switch`

6. Select the *switch.uvproj* project file and click Open. The project opens in the IDE (see Figure 5).

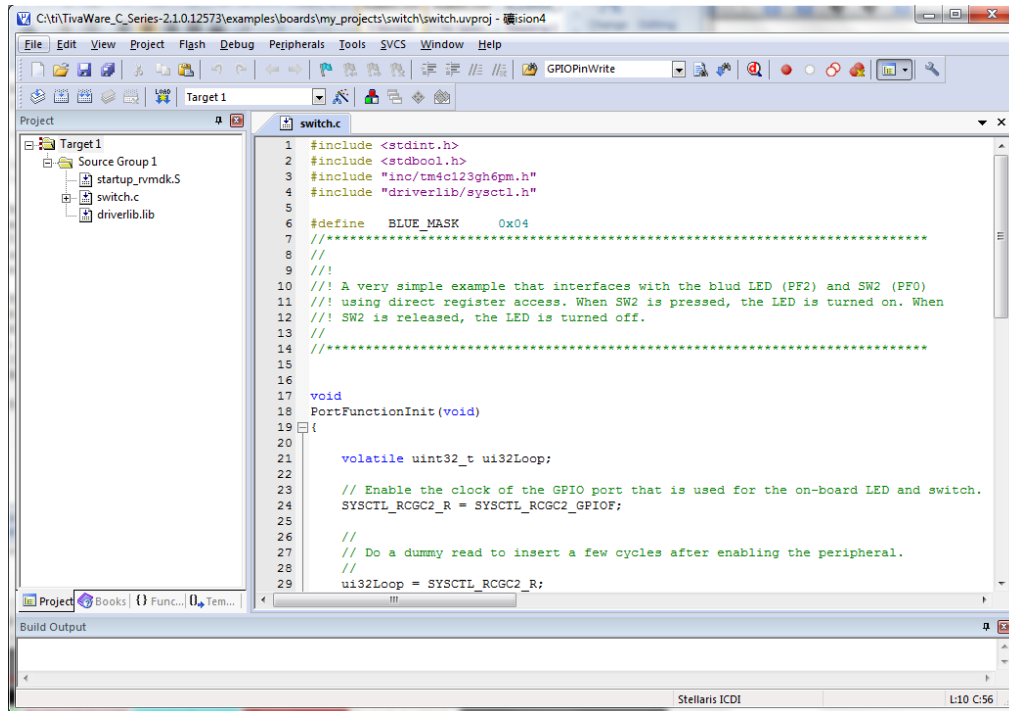


Figure 5

7. Build the project. Select Project → Rebuild all target files, or click the Rebuild all button (icon) (see Figure 6).

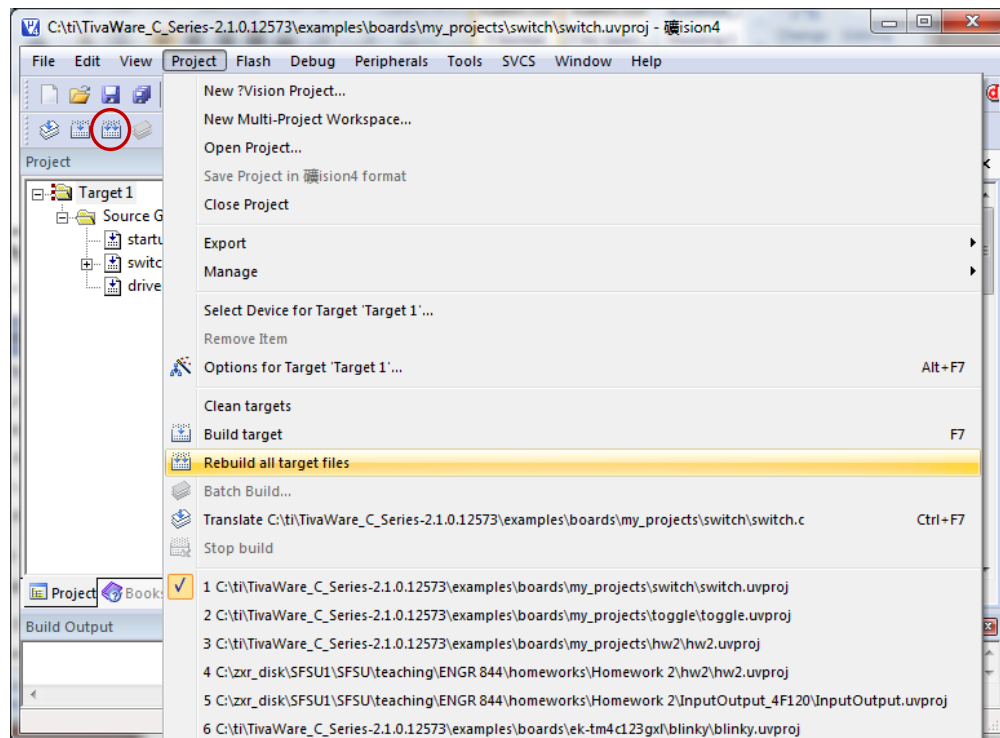


Figure 6

All of the source files are compiled and linked. The activity can be seen in the Build window at the bottom of the  $\mu$ Vision IDE. The process completes with an application named *switch.axf* built with no errors and no warnings (see Figure 7).

```
Build Output
Rebuild target 'Target 1'
assembling startup_rvmdk.S...
compiling switch.c...
linking...
Program Size: Code=1100 RO-data=16 RW-data=0 ZI-data=352
".\switch.axf" - 0 Error(s), 0 Warning(s).
```

Figure 7

8. Load the program into the flash memory (make sure you have plugged in the LaunchPad). Select Download from the Flash menu, or click the Download button (icon) (see Figure 8).

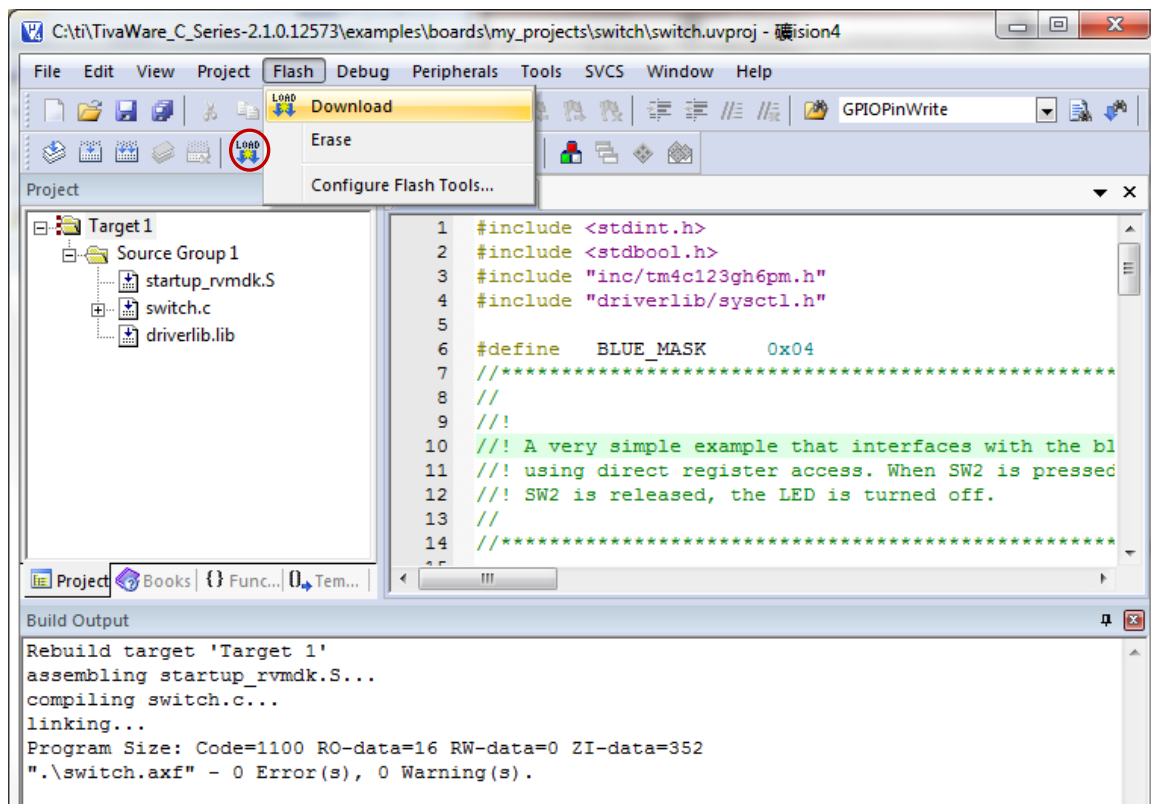


Figure 8

The process takes a few seconds. A progress bar will show at the bottom of the IDE window as the device is programmed. When it is finished, the Build window will show that the device was erased, programmed, and verified OK. The project is now programmed into the flash memory of the microcontroller on the LaunchPad.

9. Debug and run the program. Select Start/Stop Debug Session from the Debug menu or click the Debug button (icon) (see Figure 9).

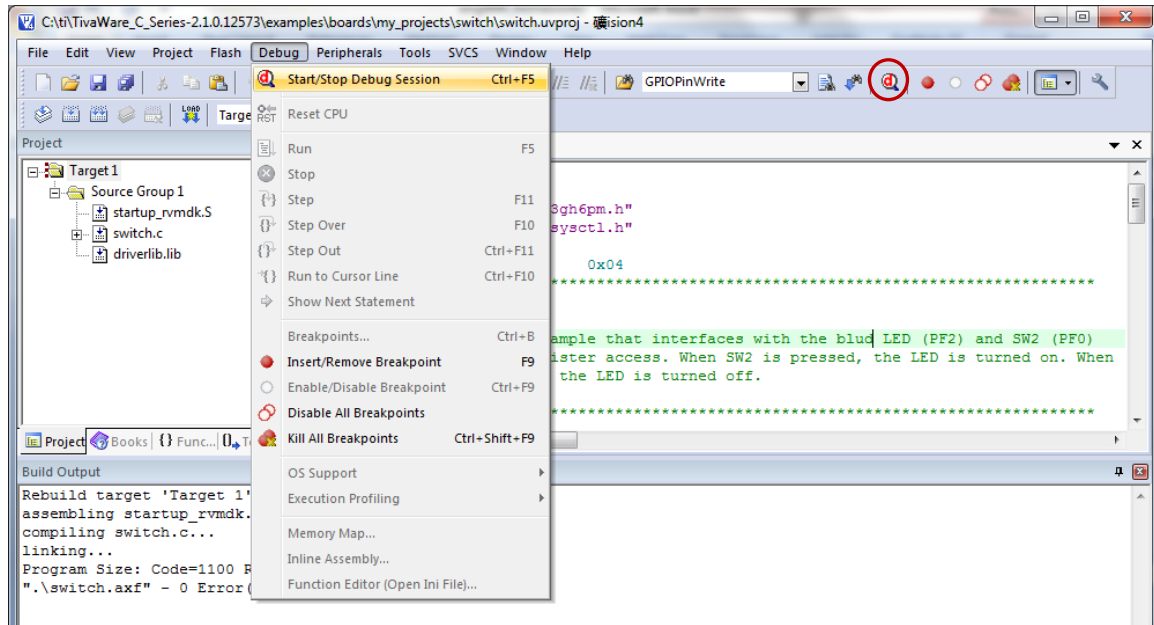


Figure 9

The IDE switches to debugging mode. The processor registers show in a window on the left, the debugger command window is visible at the bottom, and the main window shows the source code being debugged. The debugger automatically stops at main (see Figure 10).

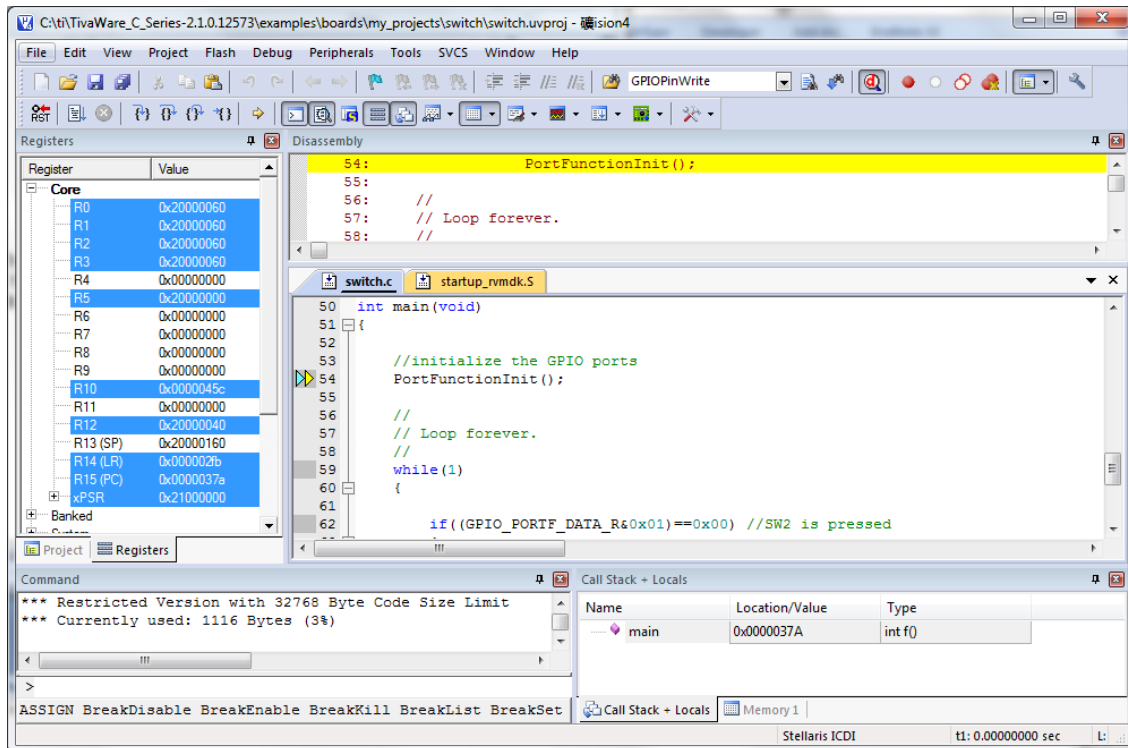


Figure 10



From here, you can:

- Examine and modify memory.
- Program variables and processor registers.
- Set breakpoints.
- Single step through a program.
- Perform other typical debugging activities.

To run the program, select Run from the Debug menu, or click the Run button (icon) (see Figure 11).

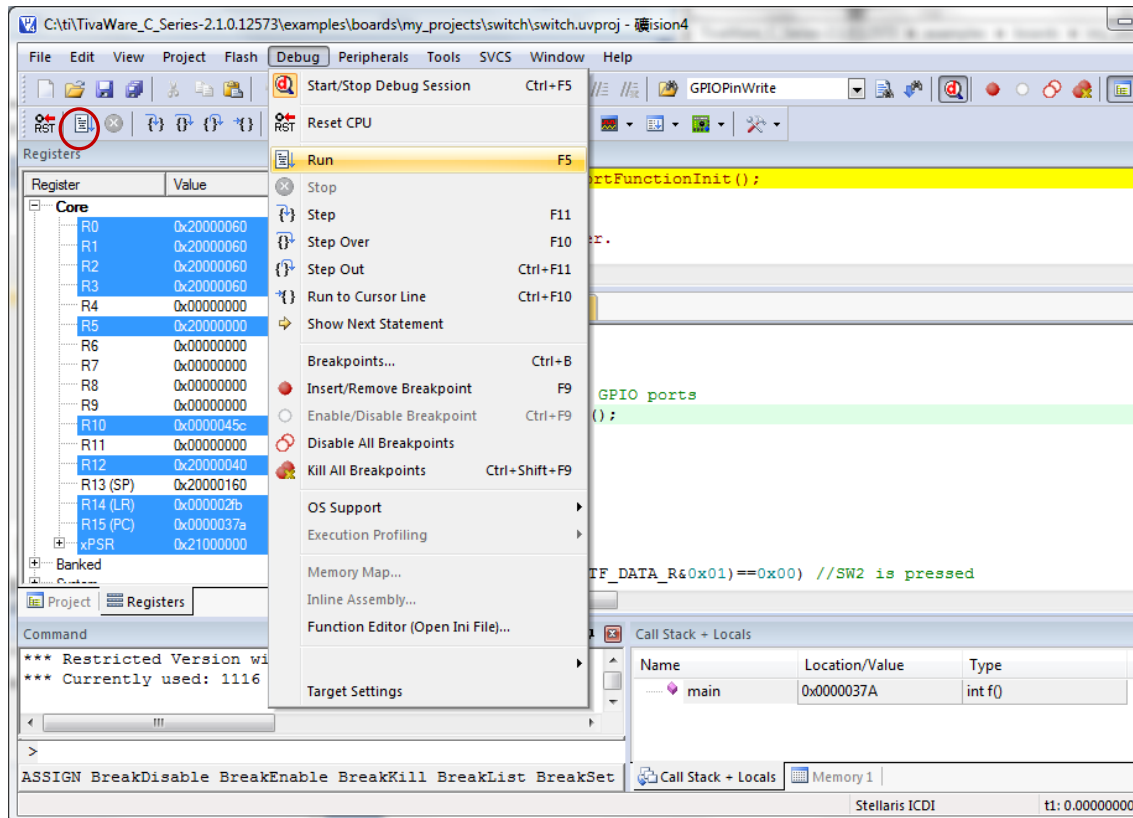


Figure 11

The application runs. When you press SW2, the blue LED is turned on, otherwise the LED is off.

## B. Build and Run Sample Project *toggle*

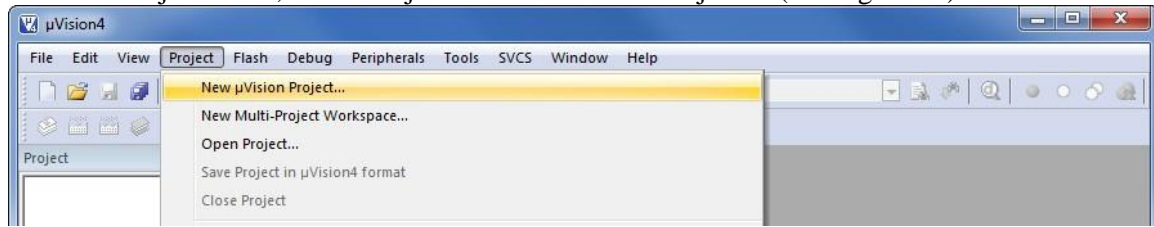
Follow the instructions in Section A, load, and run the other sample project *toggle*, and see the results.

## C. Create a New Project

Once you have gone through the sample projects, you may want to create your own project to start development. While you can always start with an existing, simple project, sometimes you

may want to create a new project. In this tutorial, we will duplicate the process of implementing the *switch* project step by step.

1. In the Project menu, select Project → New uVision Project... (see Figure 12)



. Figure 12

2. The Create New Project dialog box appears (see Figure 13). To use the functions in the TivaWare library, we need to do some configuration. In this example, we create the project within the TivaWare tree by saving it in the existing *C:\ti\TivaWare\_C\_Series-2.x.x.xxx\examples\boards\my\_projects* directory. We first create a new folder named as *my\_switch*, and then create the project file *my\_switch.uvproj* in the *C:\ti\TivaWare\_C\_Series-2.x.x.xxx\examples\boards\my\_projects\my\_switch* directory (see Figure 14) and click “save”.

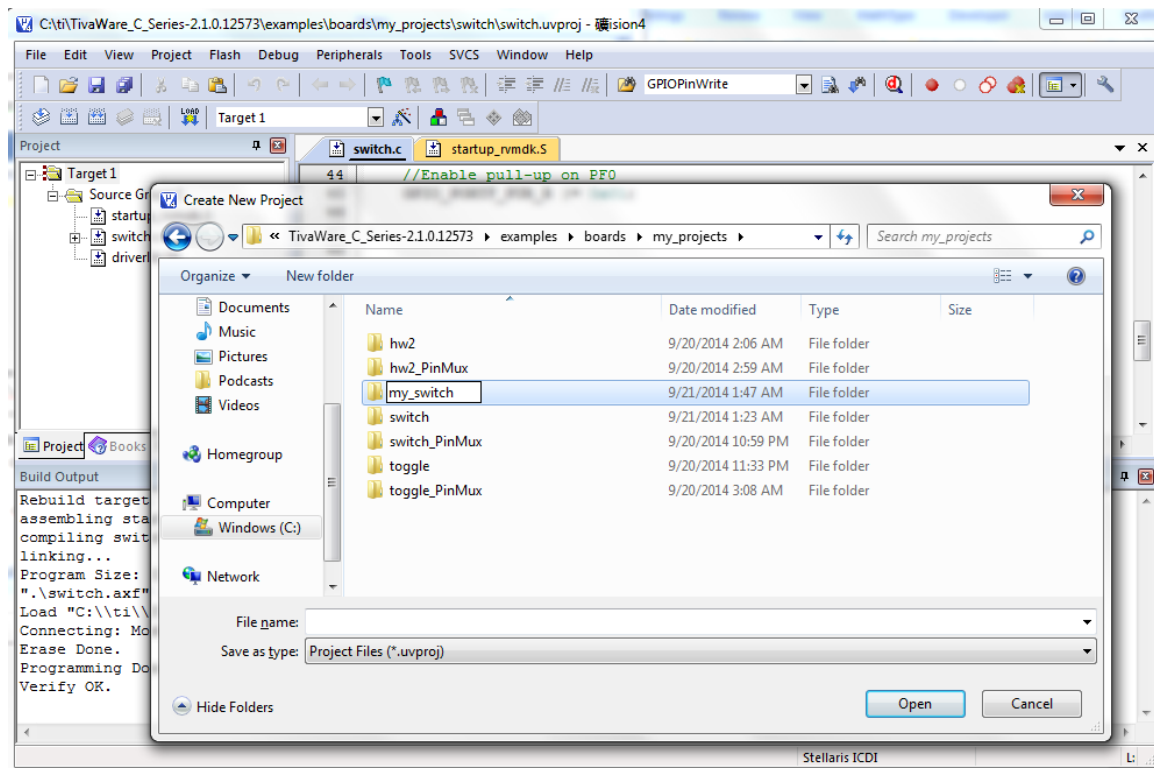


Figure 13

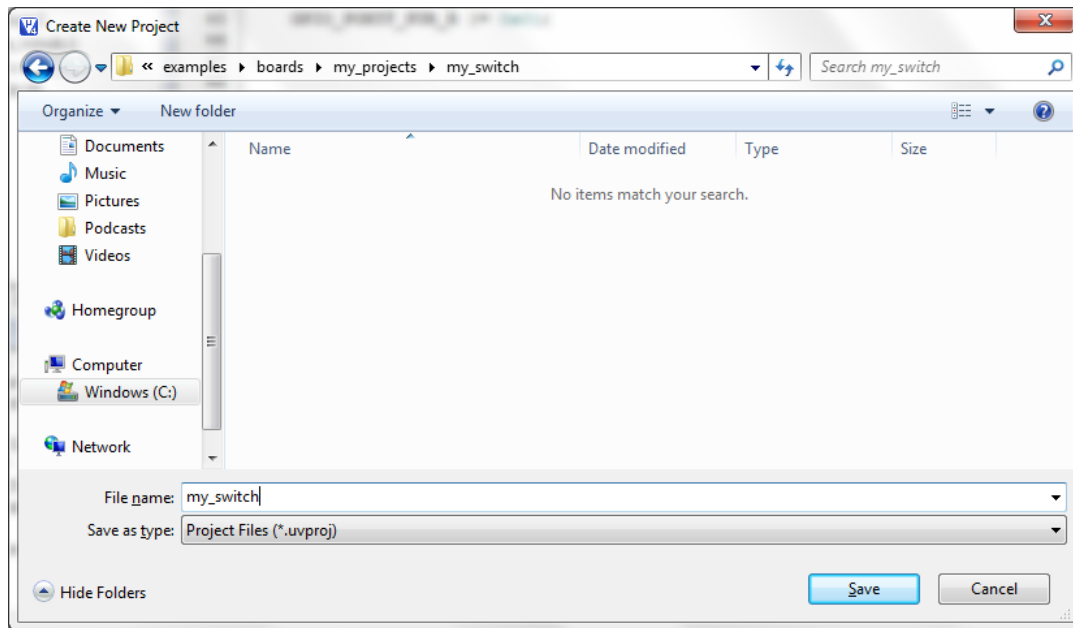


Figure 14

3. Once the project file (.uvproj) is saved, a dialog window appears asking you to select the device that you are using (see Figure 15). Select the appropriate device under the “Texas Instruments” list. The device we need to select is TM4C123GH6PM.

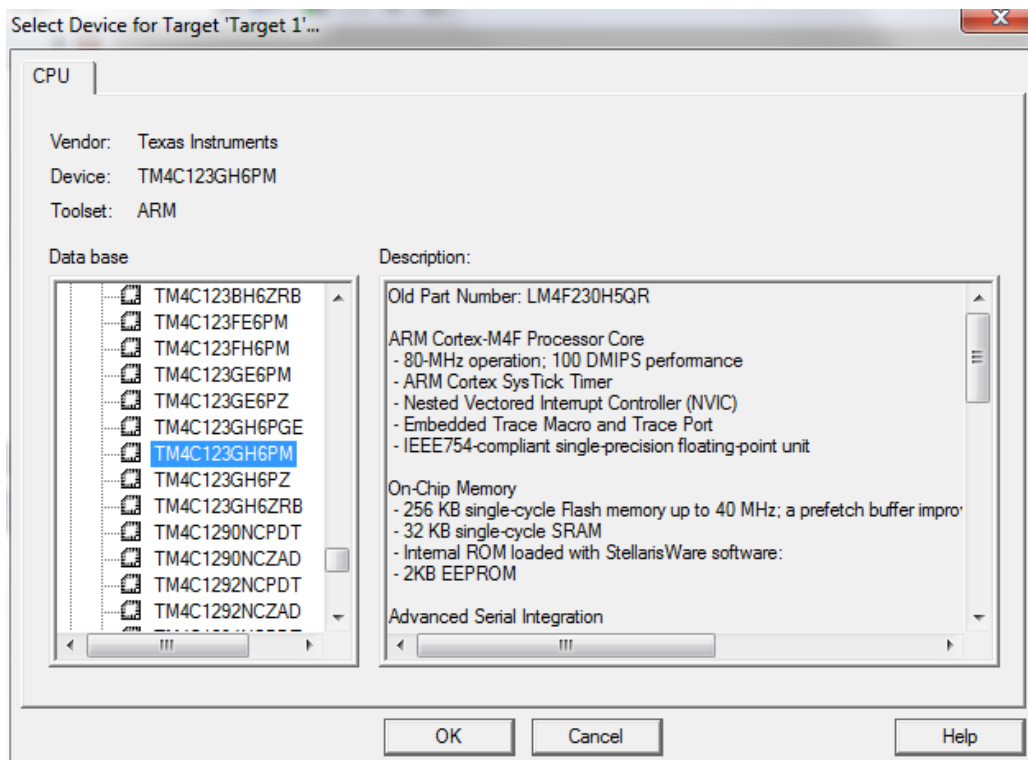


Figure 15

4. The tool asks whether you want to add startup code to the project (see Figure 16). Select “No” because we will add the startup code provided by the TivaWare library.

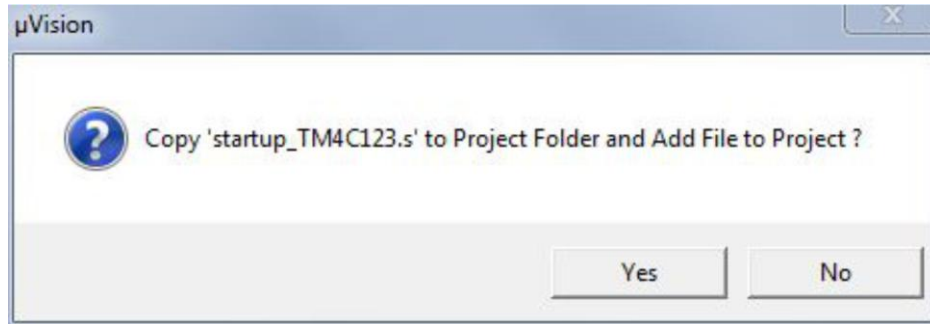


Figure 16

The empty project appears as shown in Figure 17.

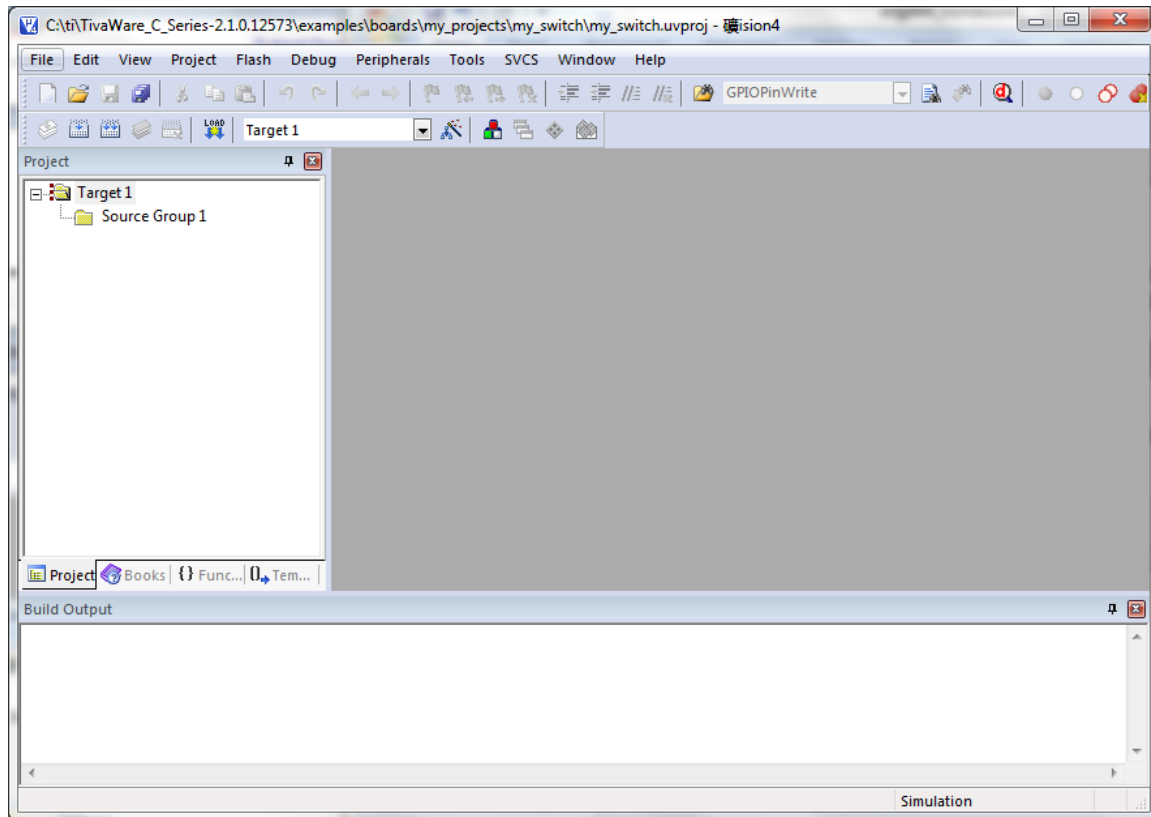


Figure 17

5. Now we will add the startup code to the project. Go to the directory where stores the sample project *switch* and copy the startup file **startup\_rvmdk.s** and paste it in the *my\_switch* directory (see Figure 18).

Note that for all the future labs and your project, we will use the same fixed startup file.

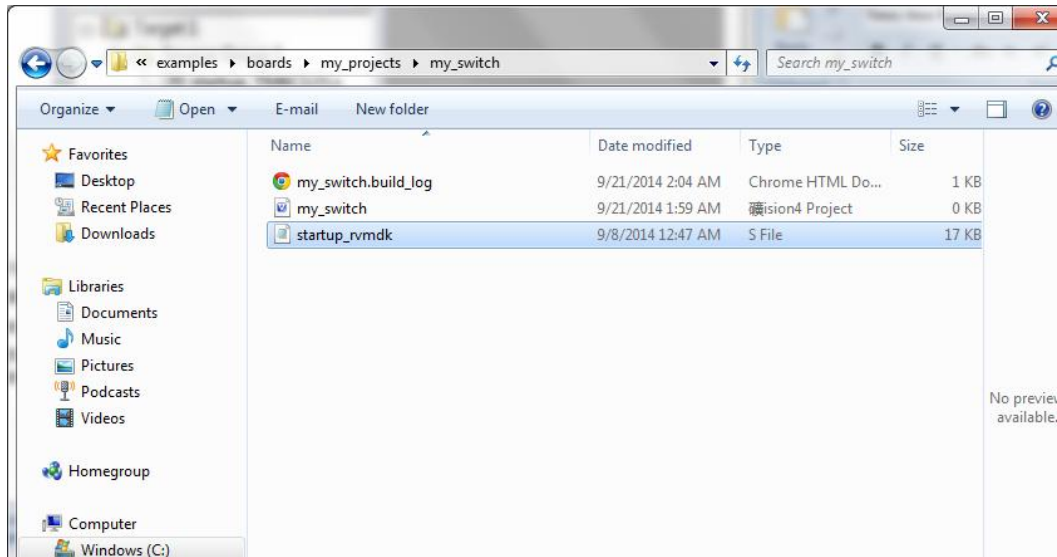


Figure 18

6. Next we will add the startup code to our project. Right-click the *Source Group 1* folder in the  $\mu$ Vision IDE and select *Add Existing Files to Group 'Source Group 1'...* (see Figure 19). When the dialog box pops up to find the file, browse to your project directory, select **startup\_rvmdk.s**, and add it to *Source Group 1* (see Figure 20).

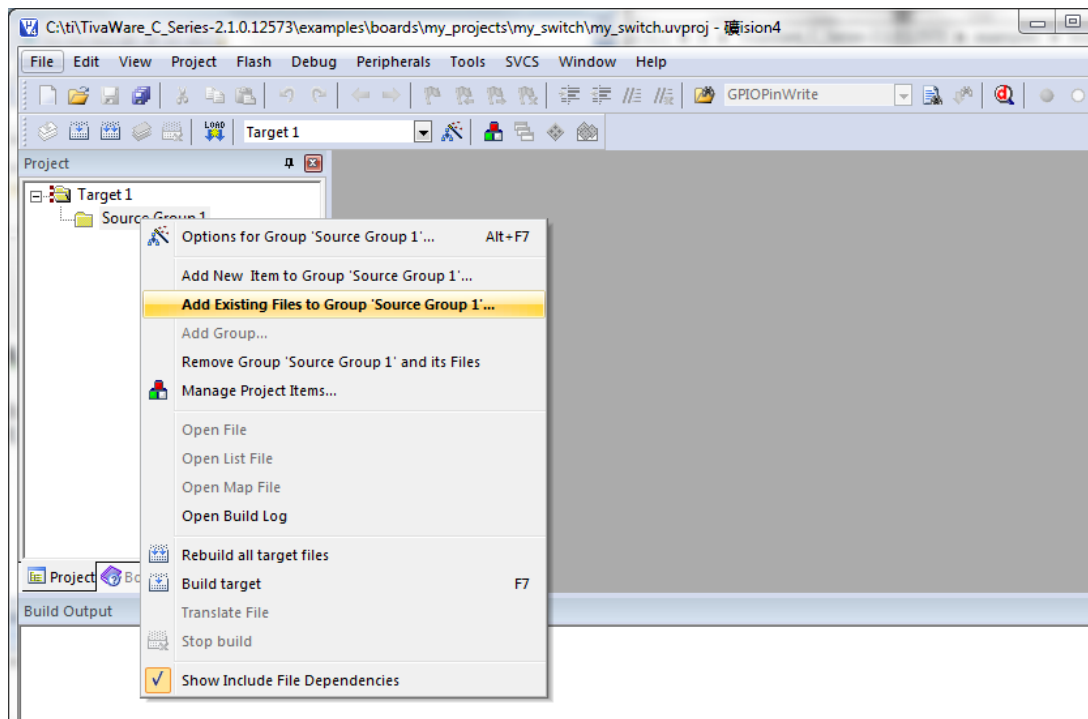


Figure 19

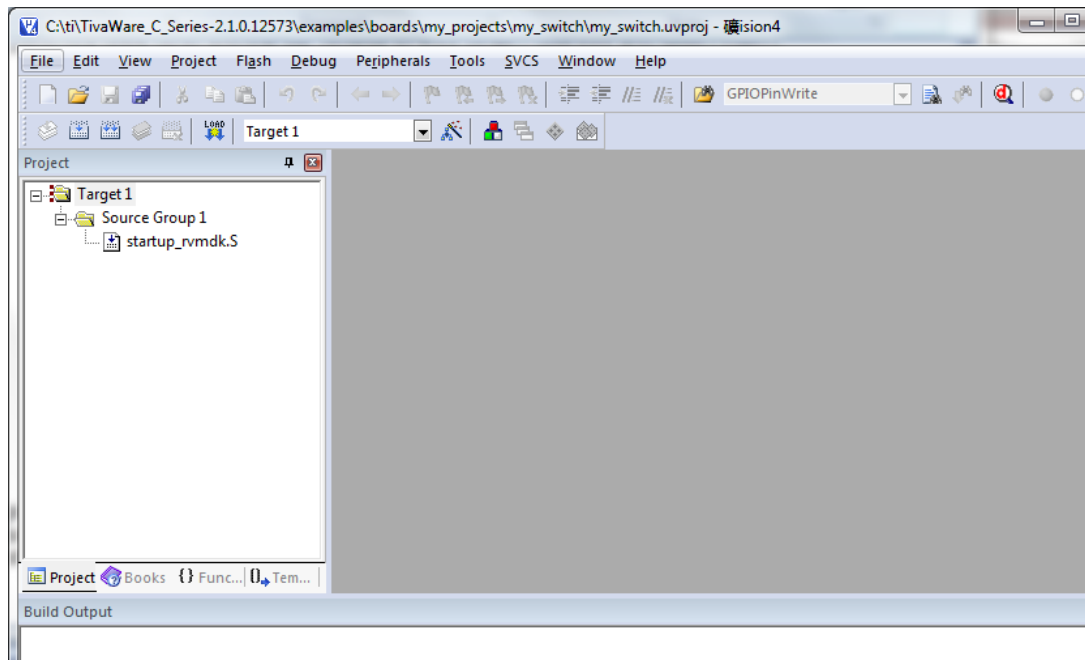


Figure 20

7. Add TivaWare Hooks. To use the functions in the TivaWare library, we need to add the **driverlib.lib** file to the project. Right-click the *Source Group 1* folder in the µVision IDE and select *Add Existing Files to Group 'Source Group 1'...* When the dialog box pops up to find the file, browse to the directory *C:\ti\TivaWare\_C\_Series-2.x.x.xxxx\driverlib\rvmdk*, select **driverlib.lib**, and add it to *Source Group 1* (see Figure 21).

Note: You must tell the file browser to look for \*.lib file types, so change the Files of type drop-down from C Source file (\*.c) to Library file (\*.lib) or All files.

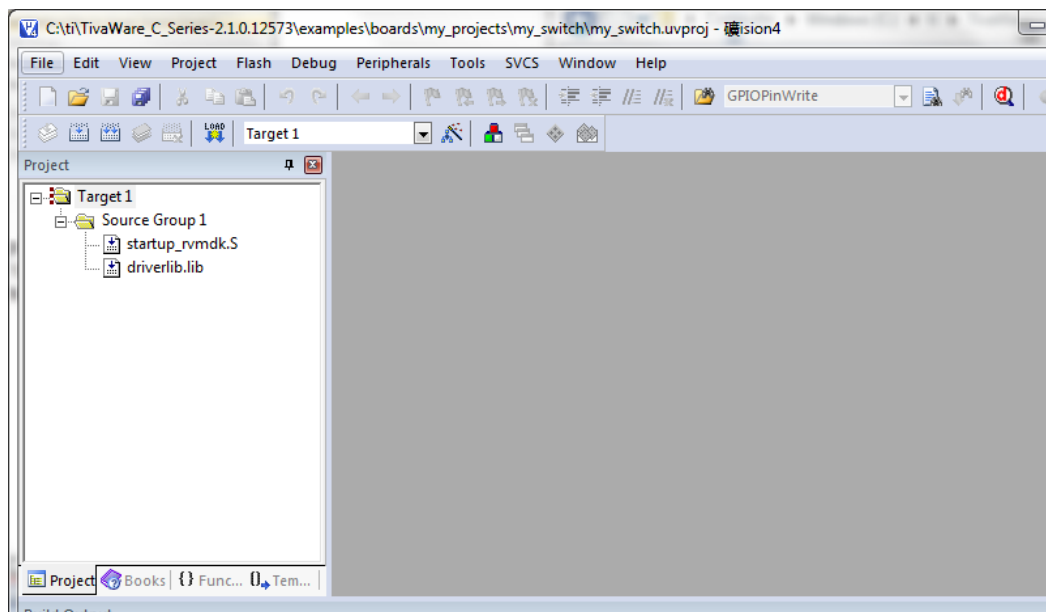


Figure 21

8. Create our main C program file by right-clicking the *Source Group 1* folder in the  $\mu$ Vision IDE and select *Add New Item to Group 'Source Group 1'...* (see Figure 22). When the dialog box pops up to add a new item, choose the file type as C File (.c), and name the new file as *my\_switch* and click Add (see Figure 23).

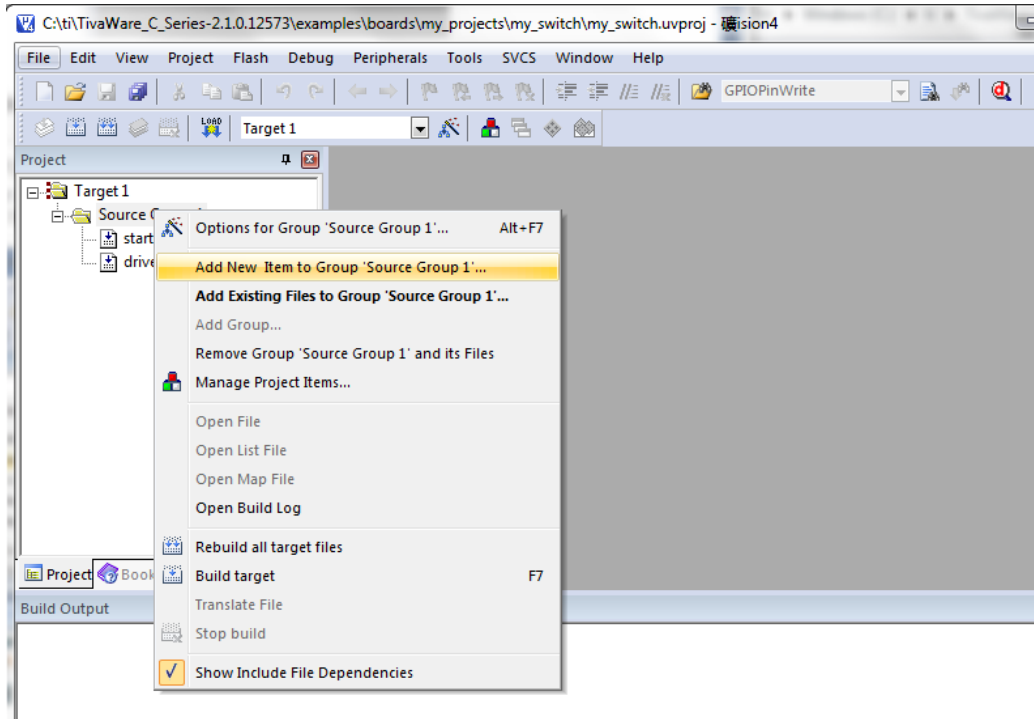


Figure 22

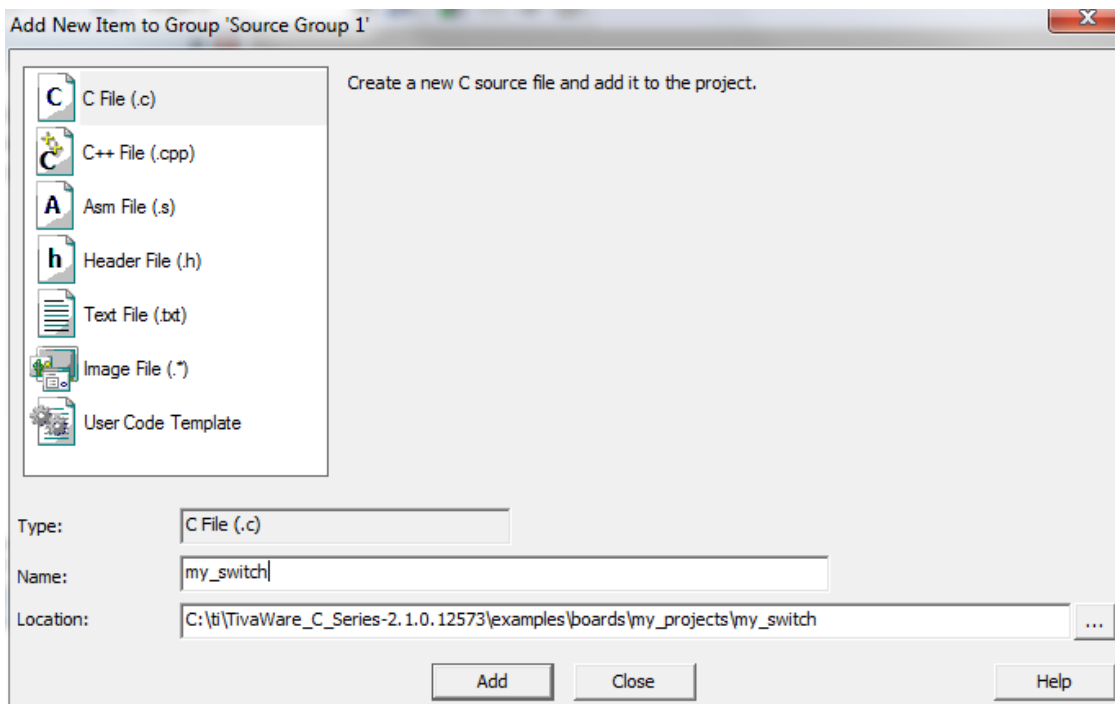


Figure 23

9. Now the file *my\_switch.c* is empty. We just copy the code in the file *switch.c* in the sample project *switch* and paste it in our *my\_switch.c* file. So far we have included all necessary files for the project (Figure 24).

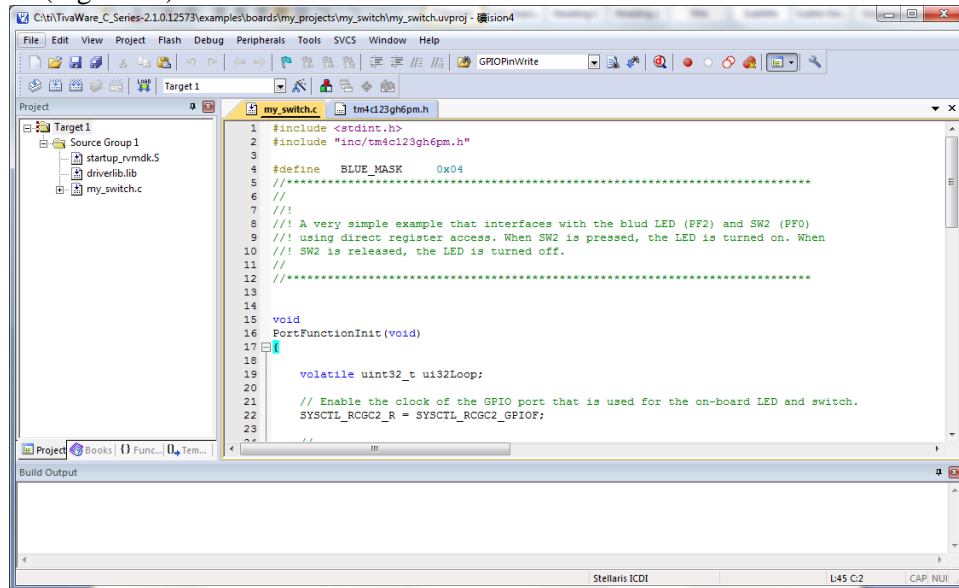


Figure 24

10. Before we can build the project correctly, we need to do some project configurations. Go to Project → Options for Target 'Target 1.' The Options for Target 'Target 1' dialog box appears (see Figure 25).

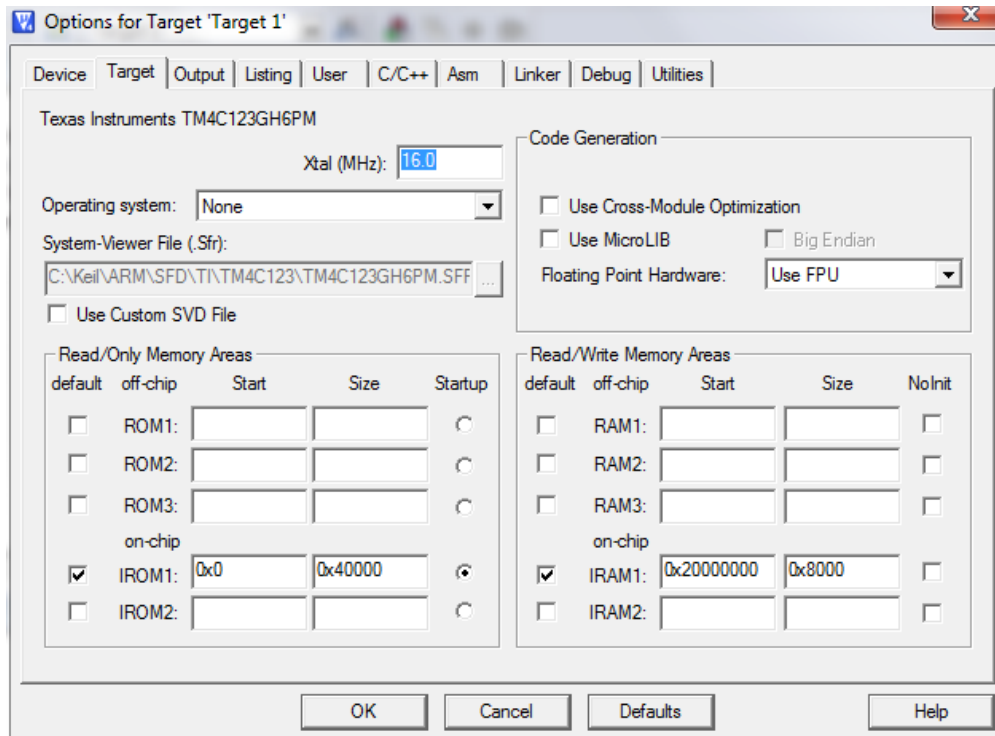
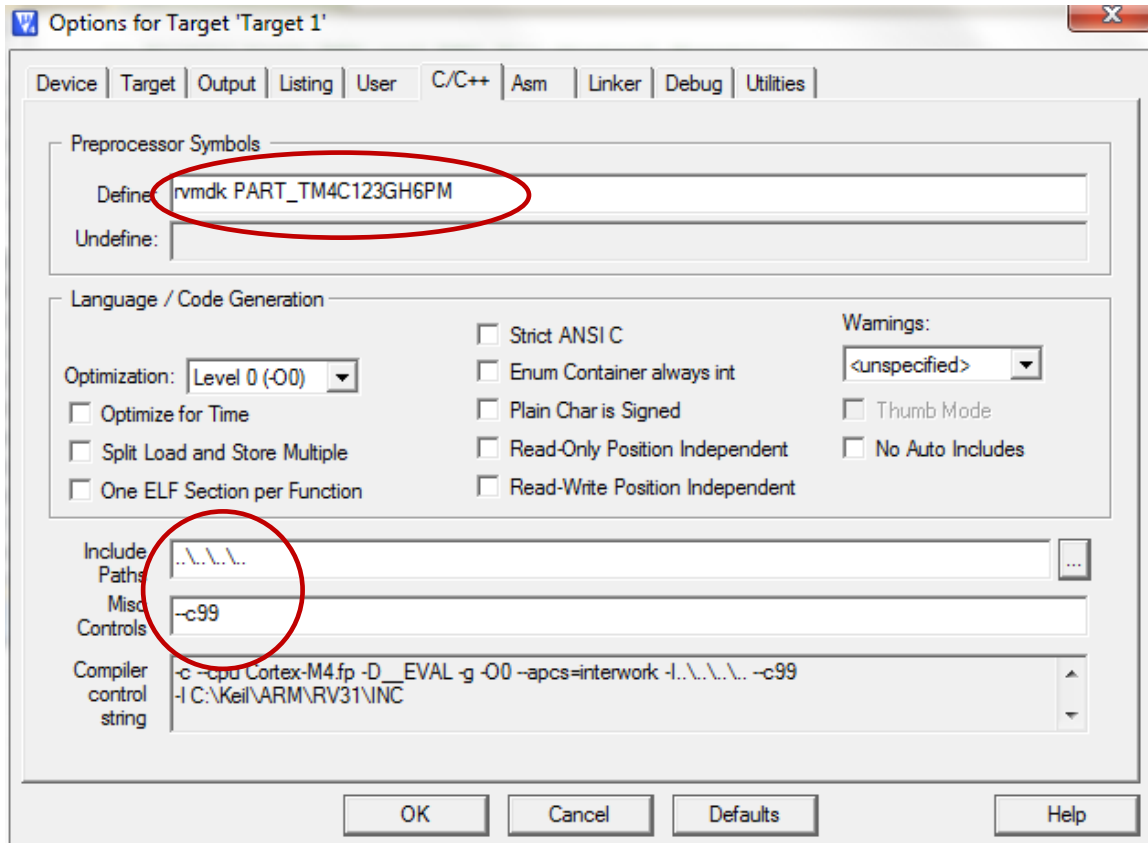


Figure 25



**11. Configure C/C++ Tab.** Select the C/C++ tab. Here you need to tell the project that you are using the ARM compiler, so define `rvmdk`. This definition is used in TivaWare to compile the Keil specific sections correctly. Define the part you are using as well as add the top-level TivaWare directory to your include path (see Figure 26).

**Note:** The easiest way to see how to configure your project is to look at the sample projects. You can launch another Keil uVision process and open the sample project *switch*. You can compare each tab in the project options, and do the configurations in your project by exactly copying the configurations from the sample project.



**Figure 26**

12. **Configure Linker Tab.** In the linker section, you must add **—entry Reset\_Handler** to the Miscellaneous controls box (see Figure 27).

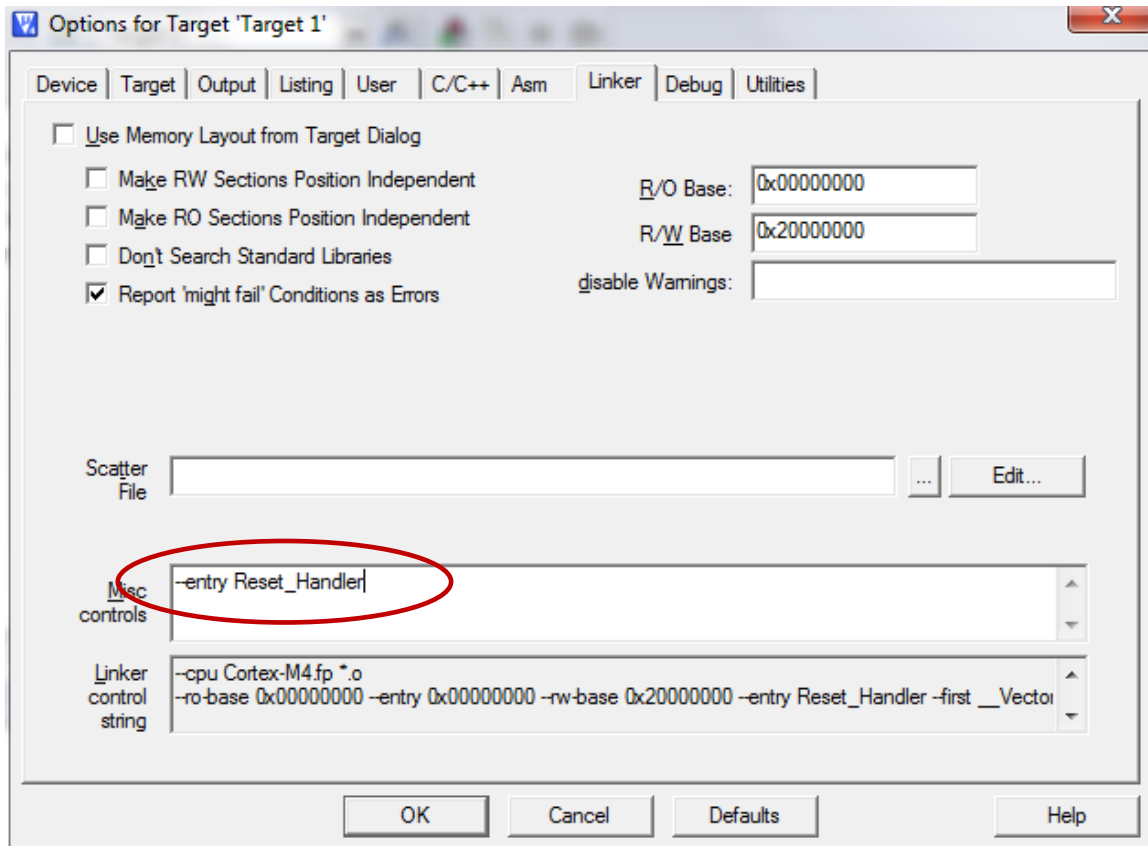
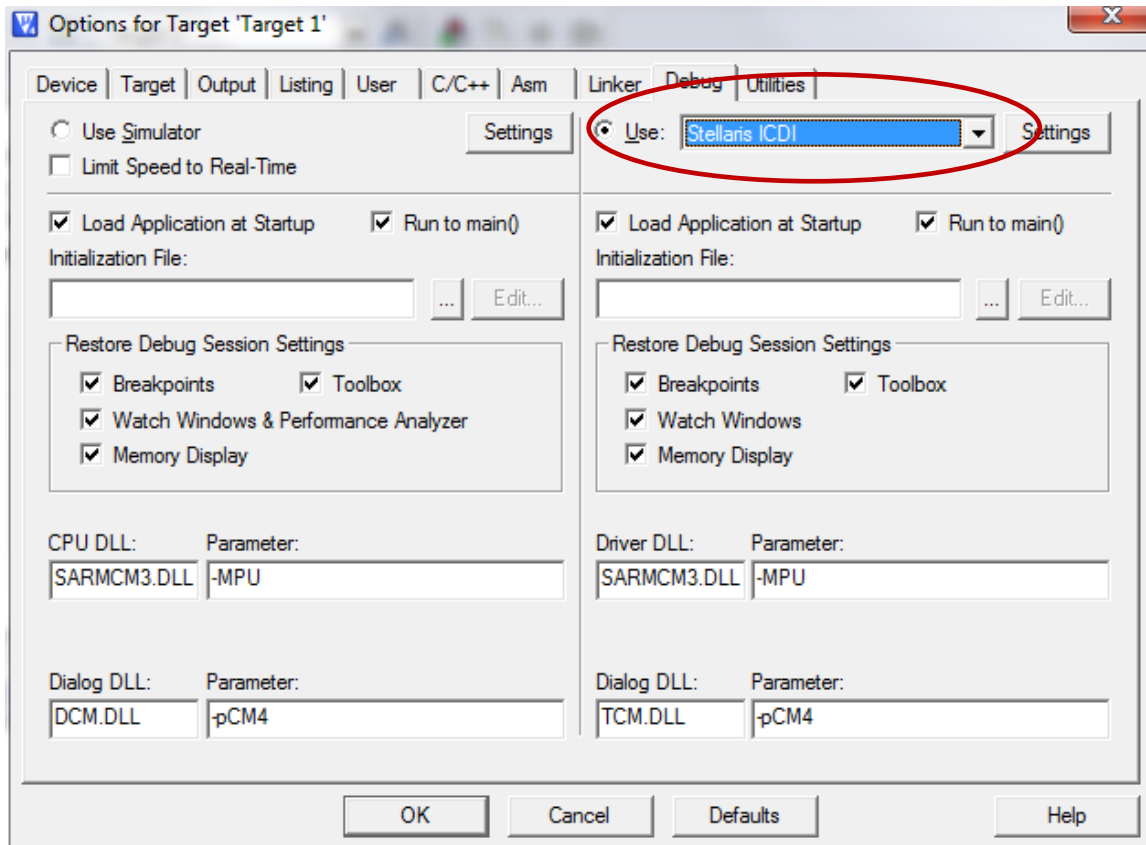


Figure 27

13. **Configure Debug Tab.** The default configuration for a new project is to use the simulator, which is not going to allow you to debug real hardware. Choose the Use radio button on the right side with the ULINK Cortex™ Debugger next to it (this selection is the default in the drop-down box). Choose the Stellaris ICDI option from the drop-down box (see Figure 28).



**Figure 28**

You have now done all necessary configurations for the created new project. You can now build, download, and run the project on your launchpad and see if it works correctly.

#### **D. Understand the Sample Project *switch***

The sample project *switch* implements a system that allows the microcontroller to interface with the on-board switch SW2 (connected with port PF0) and the blue LED (connected with port PF2). When SW2 is pressed, the LED is turned on, otherwise the LED is off.

1. The pseudo code and the flowchart (Figure 29) of the system design are shown below, illustrating the basic steps for the system.

<b>Start</b>	Initialize <b>Port F</b> , configure <b>PF2</b> as output and <b>PF0</b> as switch input
<b>loop</b>	Read the switch and test if the switch is pressed
	If <b>PF0</b> ==0 (SW2 is pressed)

Set **PF2**, so LED is on  
 If **PF0**==1 (SW2 is released)  
 Clear **PF2**, so LED is off  
 Go to **loop**

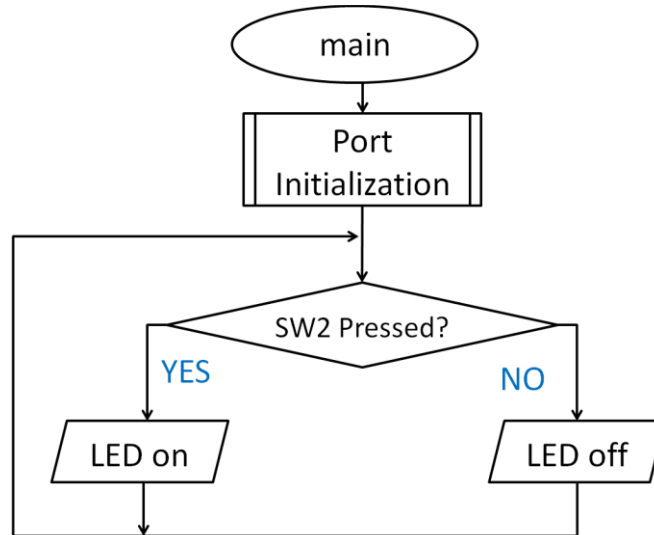


Figure 29

2. The header files included in switch.c are listed below:

```
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
```

The use of the < > restricts the search path to only the specified path. Using the " " causes the search to start in the project directory. For includes like the two standard ones, you want to assure that you're accessing the original, standard files ... not one's that may have been modified.

**stdint.h:** Variable definitions for the C99 standard

**tm4c123gh6pm.h:** Definitions for the interrupt and register assignments on the Tiva C Series device on the LaunchPad board. For detailed information of the register descriptions, please refer to the **Tiva TM4C123GH6PM Microcontroller Data Sheet** (iLearn->Reference Materials).

3. In this example, we initialize the IO ports by configure the register values manually. We first need to figure out the registers need to be configured, and then determine the values to be written to the registers. The register initialization is defined in the function *PortFunctionInit()* (see the code below). The registers such as SYSCTL\_RCGC2\_R, GPIO\_PORTF\_LOCK\_R, GPIO\_PORTF\_CR\_R, GPIO\_PORTF\_DIR\_R are all defined in **tm4c123gh6pm.h**. To find out the definition of a specific register, you can right-click the register name, and choose Go to Definition of 'register name'. Then the definition of the variable in **tm4c123gh6pm.h** will be shown to you (see Figure 30 and Figure 31).

```

void
PortFunctionInit(void)
{

    volatile uint32_t ui32Loop;

    // Enable the clock of the GPIO port that is used for the on-board LED and switch.
    SYSTCL_RCGC2_R = SYSTCL_RCGC2_GPIOF;

    //
    // Do a dummy read to insert a few cycles after enabling the peripheral.
    //
    ui32Loop = SYSTCL_RCGC2_R;

    // Unlock GPIO Port F
    GPIO_PORTF_LOCK_R = 0x4C4F434B;
    GPIO_PORTF_CR_R |= 0x01;           // allow changes to PF0

    // Set the direction of PF2 (blue LED) as output
    GPIO_PORTF_DIR_R |= 0x04;

    // Set the direction of PF0 (SW2) as input by clearing the bit
    GPIO_PORTF_DIR_R &= ~0x01;

    // Enable both PF2 and PF0 for digital function.
    GPIO_PORTF_DEN_R |= 0x05;

    //Enable pull-up on PF0
    GPIO_PORTF_PUR_R |= 0x01;
}

```

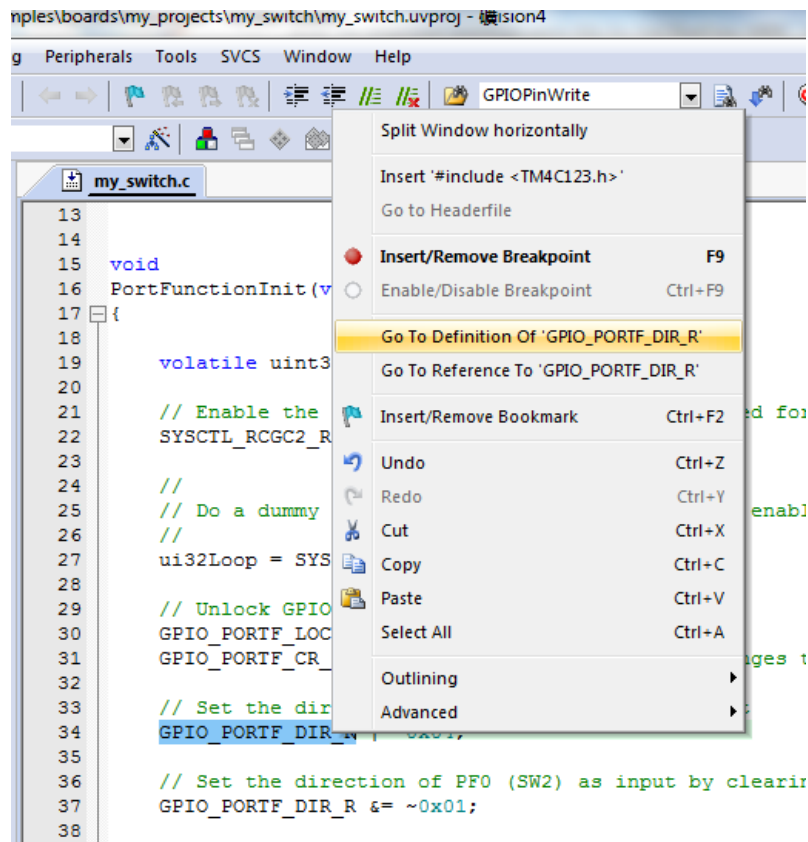
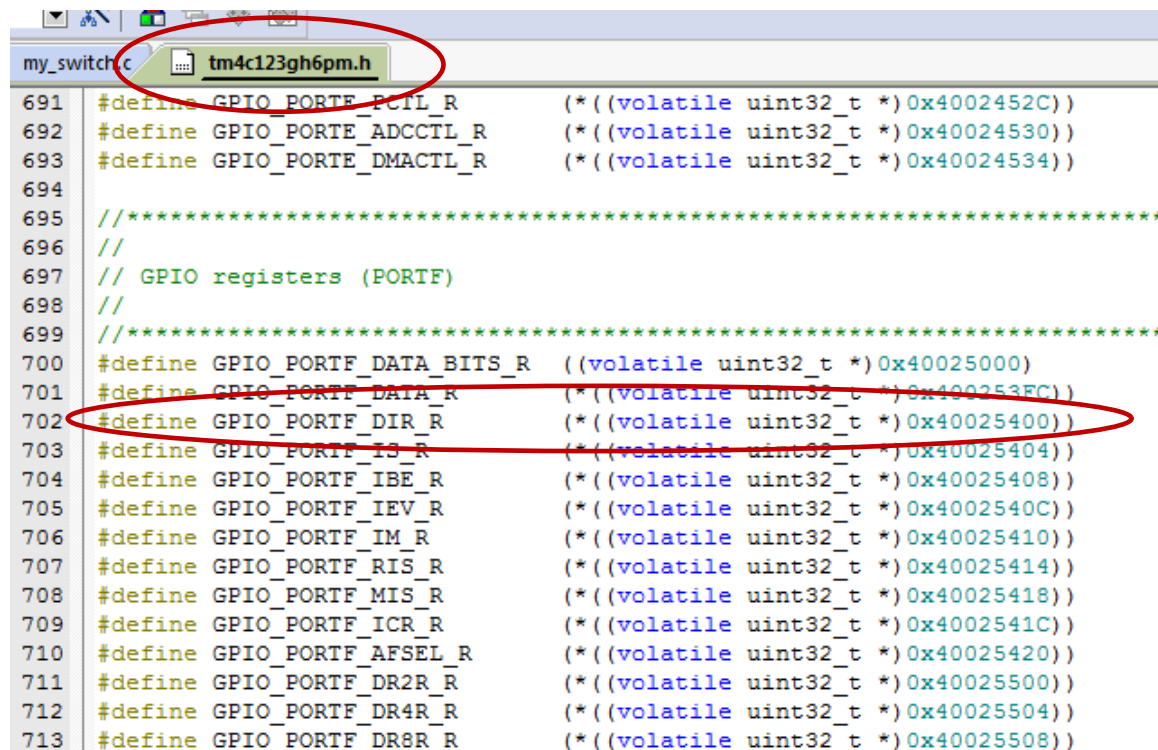


Figure 30



```

691 #define GPIO_PORTF_PCIL_R      (*((volatile uint32_t *)0x4002452C))
692 #define GPIO_PORTF_ADCCTL_R   (*((volatile uint32_t *)0x40024530))
693 #define GPIO_PORTF_DMACTL_R   (*((volatile uint32_t *)0x40024534))
694
695 //*****
696 //
697 // GPIO registers (PORTF)
698 //
699 //*****
700 #define GPIO_PORTF_DATA_BITS_R ((volatile uint32_t *)0x40025000)
701 #define GPIO_PORTF_DATA_R      (*((volatile uint32_t *)0x4002500C))
702 #define GPIO_PORTF_DIR_R      (*((volatile uint32_t *)0x40025400))
703 #define GPIO_PORTF_IS_R       (*((volatile uint32_t *)0x40025404))
704 #define GPIO_PORTF_IBE_R      (*((volatile uint32_t *)0x40025408))
705 #define GPIO_PORTF_IEV_R      (*((volatile uint32_t *)0x4002540C))
706 #define GPIO_PORTF_IM_R       (*((volatile uint32_t *)0x40025410))
707 #define GPIO_PORTF_RIS_R      (*((volatile uint32_t *)0x40025414))
708 #define GPIO_PORTF_MIS_R      (*((volatile uint32_t *)0x40025418))
709 #define GPIO_PORTF_ICR_R      (*((volatile uint32_t *)0x4002541C))
710 #define GPIO_PORTF_AFSEL_R    (*((volatile uint32_t *)0x40025420))
711 #define GPIO_PORTF_DR2R_R     (*((volatile uint32_t *)0x40025500))
712 #define GPIO_PORTF_DR4R_R     (*((volatile uint32_t *)0x40025504))
713 #define GPIO_PORTF_DR8R_R     (*((volatile uint32_t *)0x40025508))

```

Figure 31

4. In this example, we need to configure PF0 as an output port and PF2 as an input port. To access PF0, we need to unlock it first. Here we use two lines of code to unlock PF0:

```

// Unlock GPIO Port F
GPIO_PORTF_LOCK_R = 0x4C4F434B;
GPIO_PORTF_CR_R |= 0x01;           // allow changes to PF0

```

In addition, PF0 is connected with the on-board switch SW2. The switches on the LaunchPad are using negative logic. We need to enable the pull-up register on PF0 in order for SW2 to function correctly.

```

//Enable pull-up on PF0
GPIO_PORTF_PUR_R |= 0x01;

```

5. In the main function *main()*, the system is implemented by converting the flow chart shown in Figure 29 into C program. The main function starts with the port initialization function *PortFunctionInit()*, followed by an infinite while loop which checks the status of SW2 repeatedly.

```

int main(void)
{
    //initialize the GPIO ports
    PortFunctionInit();

    //
    // Loop forever.
    //
    while(1)
    {
        if((GPIO_PORTF_DATA_R&0x01)==0x00) //SW2 is pressed
        {
            // Turn on the LED.
            GPIO_PORTF_DATA_R |= 0x04;
        }
        else
        {
            // Turn off the LED.
            GPIO_PORTF_DATA_R &= ~0x04;
        }
    }
}

```

### E. Understand the Sample Project *toggle*

The sample project *toggle* implements a system that toggles the on-board red LED (connected with port PF1) repeatedly.

1. The pseudo code and the flowchart (Figure 32) of the system design are shown below, illustrating the basic steps for the system.

<b>Start</b>	Initialize <b>Port F</b> , configure <b>PF1</b> as output Initialize the value on <b>PF1</b> as 1 (red LED on)
<b>loop</b>	Delay for a number of cycles toggle <b>PF1</b> Go to <b>loop</b>

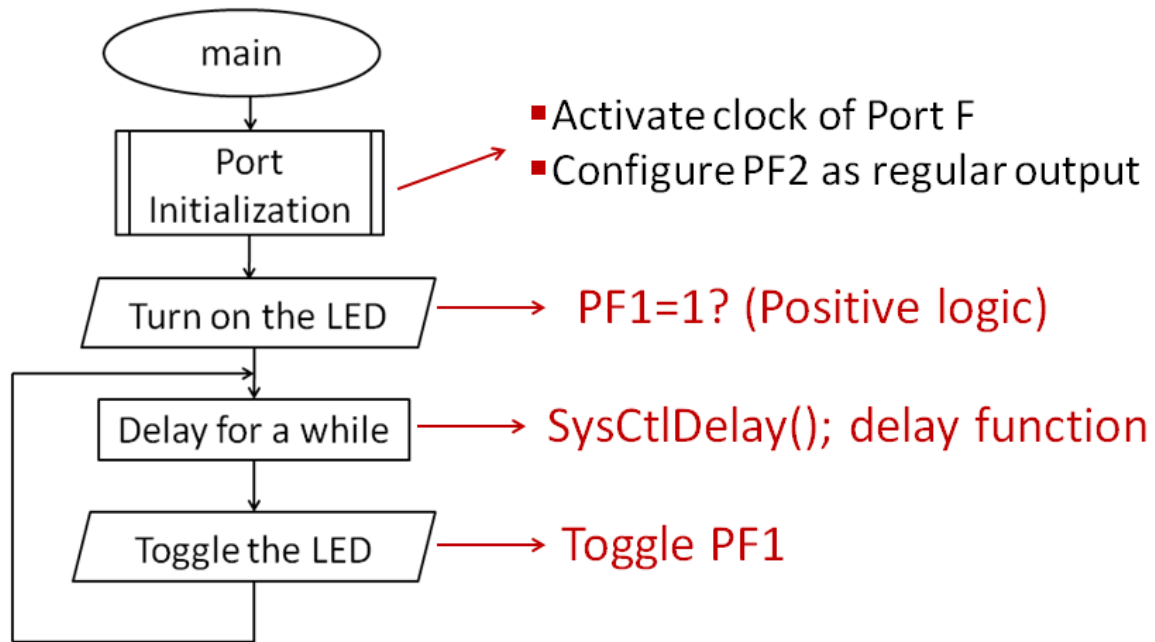


Figure 32

2. **SysCtlDelay**(unit32\_t ui32Count) is a delay function provided in the TivaWare library. It implements a loop timer in which the parameter ui32Count is the number of delay loop iterations to perform. Please search for “SysCtlDelay” in the *TivaWare Peripheral Driver Library User Guide* and find the detailed description of this function.

### Questions

1. Modify the C program in the project *my\_switch* created by yourself in Section C. In the modified system, when **SW2** (connected with **PF0**) is pressed, the **red LED** (connected with **PF1**) is turned **off**, otherwise the **red LED** is **on**. You need to modify the configurations in both the *PortFunctionInit()* function and the *main()* function. In your lab report, please include the following:

- The flow chart of the system
- Explanation of your modifications
- The C program listing with detailed comments for each line of code.
- The execution results of your program (your observations).

2. Create a new project *my\_toggle* by following the instructions of creating a new project in Section C. Copy the code in the file *toggle.c* in the sample project *toggle* and paste it in your created project. Modify the code to make the **green LED** (connected with **PF3**) flash. Also try to make the LED flash **5 times slower**. You need to modify the configurations in both the *PortFunctionInit()* function and the *main()* function. In your lab report, please include the following:

- The flow chart of the system
- Explanation of your modification
- The C program listing with detailed comments for each line of code.
- The execution results of your program (your observations).

Reference: The figure below shows the LED and switch interfaces on the LaunchPad.



