Lab 4: GPIOs; Switches and LED Interfaces
Use GPIOPinRead/GPIOPinWrite and PinMux Software Tool

Nima Karimian

Preparation

- Review the content of lecture 7, 8
- Before you start the lab, make sure you have completed Lab 1, 3 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.
- Download the Tiva C Series PinMux Utility from Canvas (Canvas->Software->TM4C_PinMux.rar). Uncompress the file and install the PinMux software tool on your PC.

References

- Getting Started with the Tiva TM4C123G LaunchPad Workshop Student Guide and Lab Manual (Chapter 1 and Chapter 3) (Canvas -> Reference Materials -> TM4C123G_LaunchPad_Workshop_Workbook.pdf)
- TivaWare Peripheral Driver Library User's Guide (Canvas-> Reference Materials)
- Tiva TM4C123GH6PM Microcontroller Data Sheet (Canvas-> 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 and the PinMx graphical tool. 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 Canvas->Labs->Lab4 report submission. The lab report should include

- The students' names, emails and IDs
- The answers of all the questions/assignments in the lab handout
- Discussion and suggestions: Through your lab experiments, what have you learned? Do you have any suggestions for future labs, lectures 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

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. Download and save the Sample Projects

1. Download the sample projects from Canvas (Canvas->Labs->sample_projects). There are four sample projects: *switch*, *switch_PinMux*, *toggle*, and *toggle_PinMux*. You already worked on *switch* and *toggle* in Lab 3. *switch* and *switch_PinMux* perform the same functionality. The only difference is that *switch* is developed by manually writing the initialization code and *switch_PinMux* is developed with the help of the PinMux graphical tool and the TivaWare Library. They both develop 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* and *toggle_PinMux* also perform the same functionality. They both implement 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-2.1.0.12573. First create a new directory "my_projects" (you may have already created it in Lab 2) under C:\ti\TivaWare_C_Series-2.1.0.12573\examples\boards and then save and uncompress the sample projects under "C:\ti\TivaWare_C_Series-2.1.0.12573\examples\boards\my_projects" as shown in Figure 1.

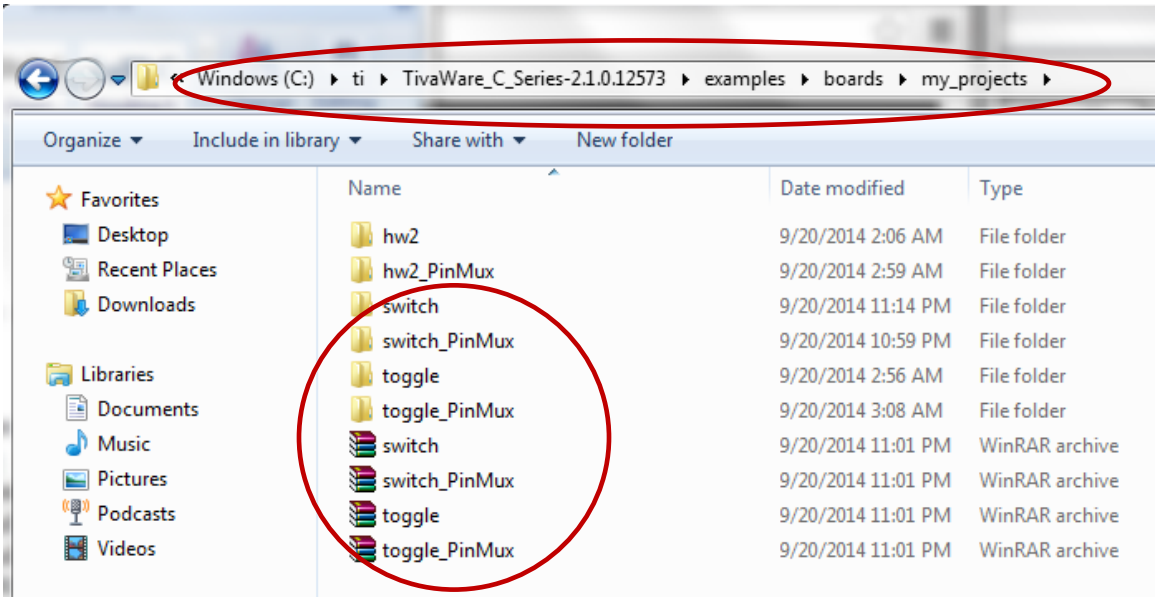


Figure 1

B. Build and Run the Sample Projects

Like what you did in Lab 3, load, compile, download, and run the sample projects in Keil uVision, and see the results.

C. Use the PinMux Graphical Tool

The software development of initializing IO ports can be simplified by using the Tiva C Series PinMux Utility. To demonstrate the use of PinMux, we develop a sample project *switch_PinMux*, which implements the same functionality as *switch* does. This tutorial explains how to implement the software in *switch_PinMux*.

1. The *switch_PinMux* project can be created in exactly the same way as how we create *switch* as described in Lab 3. The only difference is the code in *switch_PinMux.c* will be like this (see the code below and on the next page).

```
#include <stdint.h>
#include <stdbool.h>
#include "switch_PinMux.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "inc/tm4c123gh6pm.h" //manually added by the programmer
```

Generated by PinMux

Manually added by the programmer

```
void
PortFunctionInit(void)
{
    //
    // Enable Peripheral Clocks
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    //
    // Enable pin PF2 for GPIOOutput
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);

    //
    // Enable pin PF0 for GPIOInput
    //

    //
    //First open the lock and select the bits we want to modify in the GPIO commit register.
    //
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x1;

    //
    //Now modify the configuration of the pins that we unlocked.
    //
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);

    //*****************************************************************
    //  The code above is automatically generated by PinMux
    //  The code below should be manually added by the programmer
    //*****************************************************************

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

}
```

Generated by PinMux

Manually added by the programmer

```
int main(void)
{

    //initialize the GPIO ports
    PortFunctionInit();

    //
    // Loop forever.
    //
    while(1)
    {

      if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0)==0x00)  //SW2 is pressed
      {
          GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x04);
      }
      else
      {
          GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0x00);
      }
    }
}
```

All developed by the programmer

2. Next we will learn how to use PinMux to automatically generate code for IO port initialization. If you haven't installed the PinMux software, download Canvas->Software->TM4C_PinMux.rar, uncompress it, and install it on your computer.

3. Launch the PinMux software by double clicking the **Tiva C Series PinMux** icon on your desktop. You will see the user interface which allows you to select the **Device Series** and **Device** as shown in Figure 2. Select "TM4C123x series" and "TM4C123GH6PM" from the menus respectively and then click **Go**.

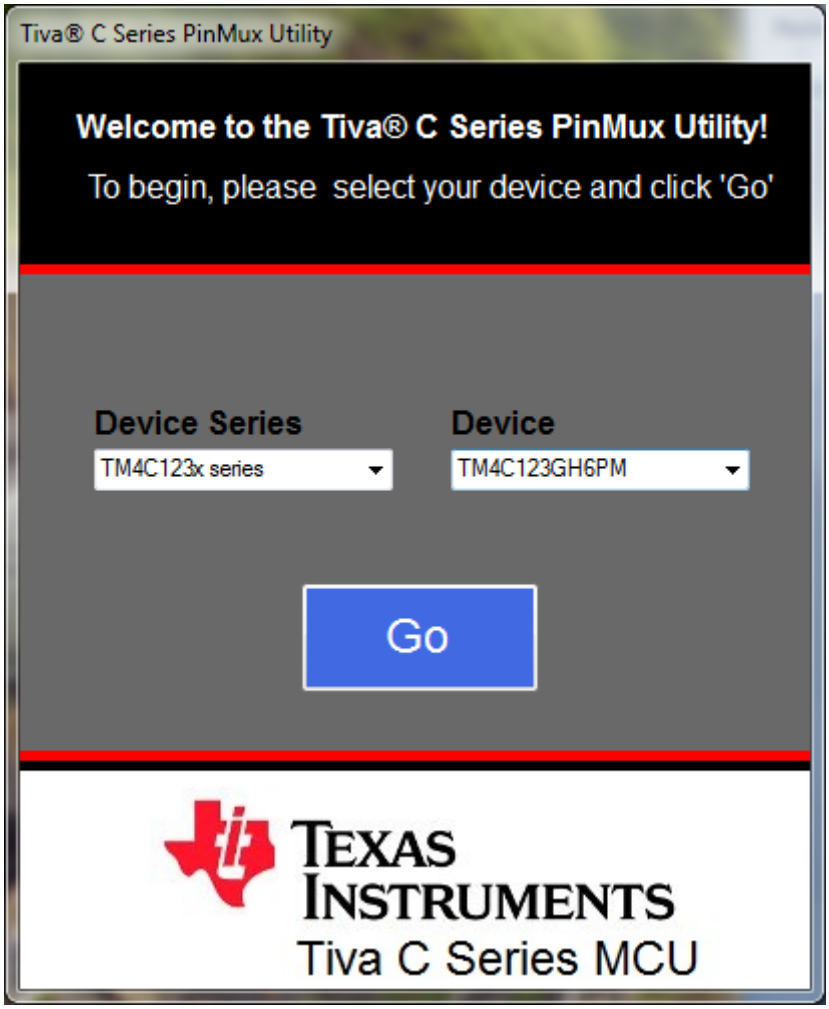
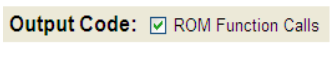


Figure 2

4. You will then see the user interface that allows you to graphically configure your device peripherals in an intuitive and rapid manner (see Figure 3). There is an option on the top right of the user interface 

, which is checked in default. This option allows the user to choose whether to generate code using TivaWare API calls stored in the internal ROM

of the microcontroller or in the flash memory. For the sample projects used in this lab, this option is unchecked, which means the generated code will be stored in the flash memory.

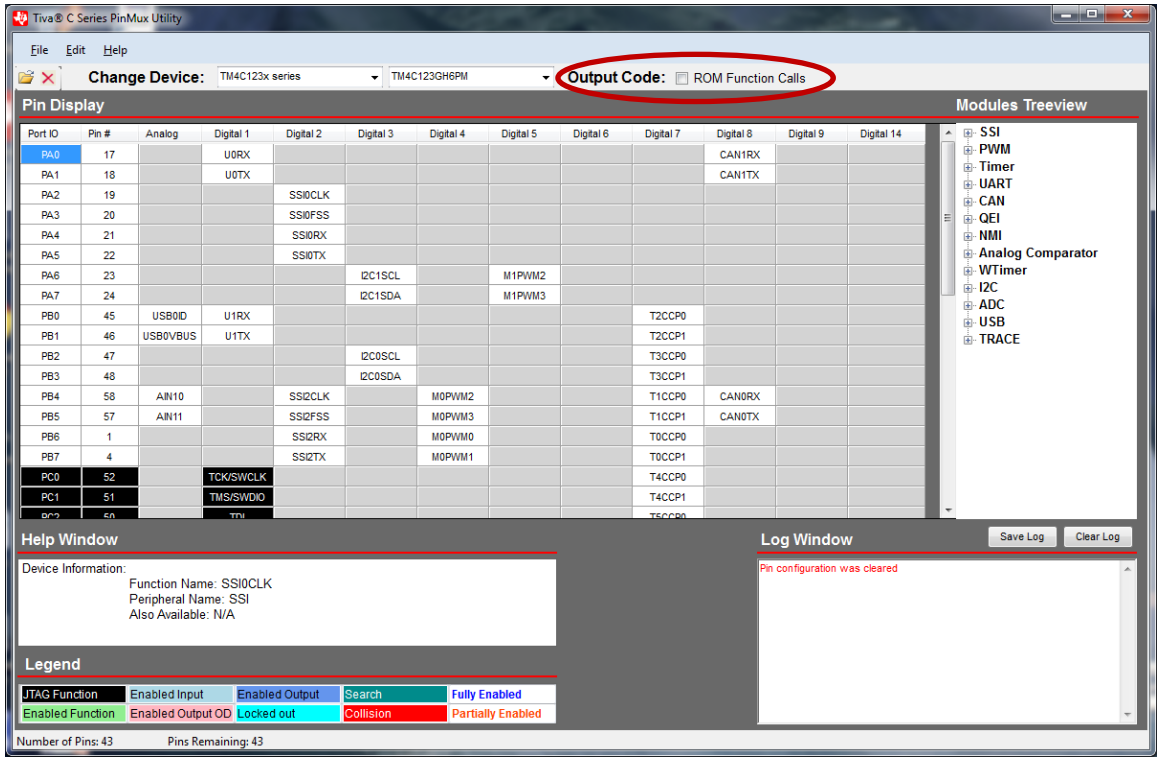


Figure 3

5. For the *switch_PinMux* project, we need to configure PF0 as a regular digital input port and PF2 as a regular digital output port. To configure PF0 as a GPIO input, scroll down the list of pins and find PF0, right-click the pin number '28', choose *Enable GPIO Function* in the drop down menu, and then *Enable as Input* in the sub drop-down menu (see Figure 4). To configure PF2, find PF2 in the list of pins, right-click the pin number '30', choose *Enable GPIO Function*, and then *Enable as Output* (see Figure 5).

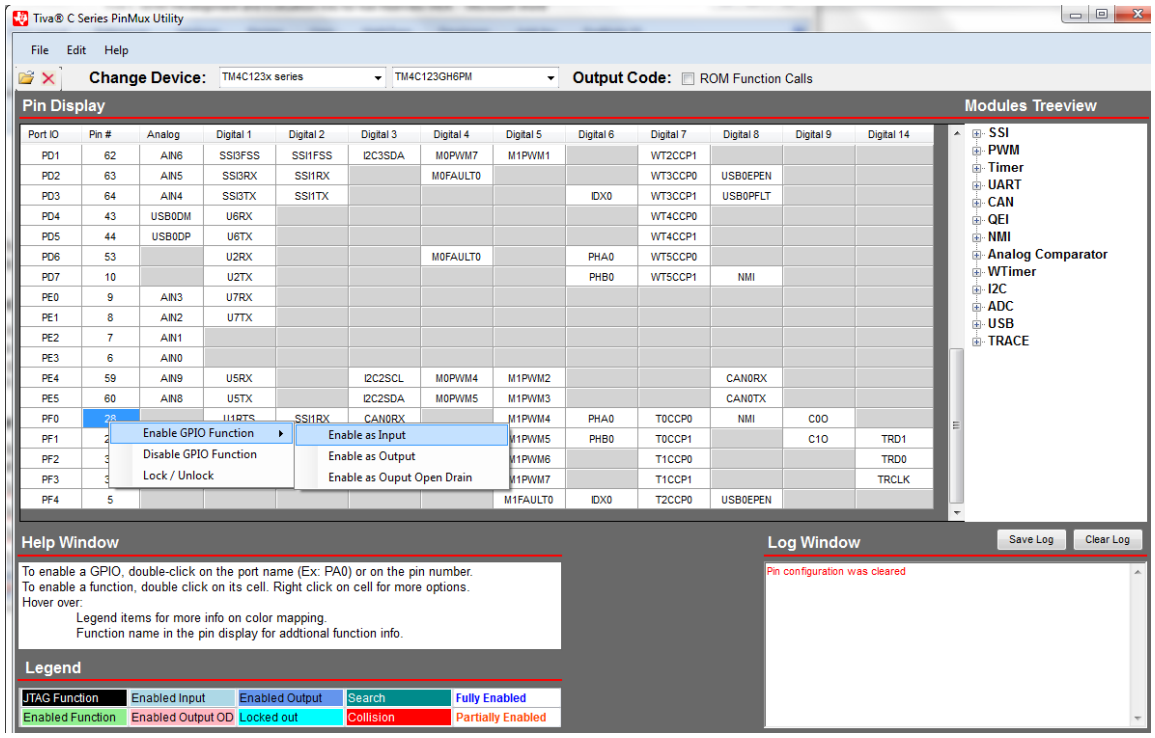


Figure 4

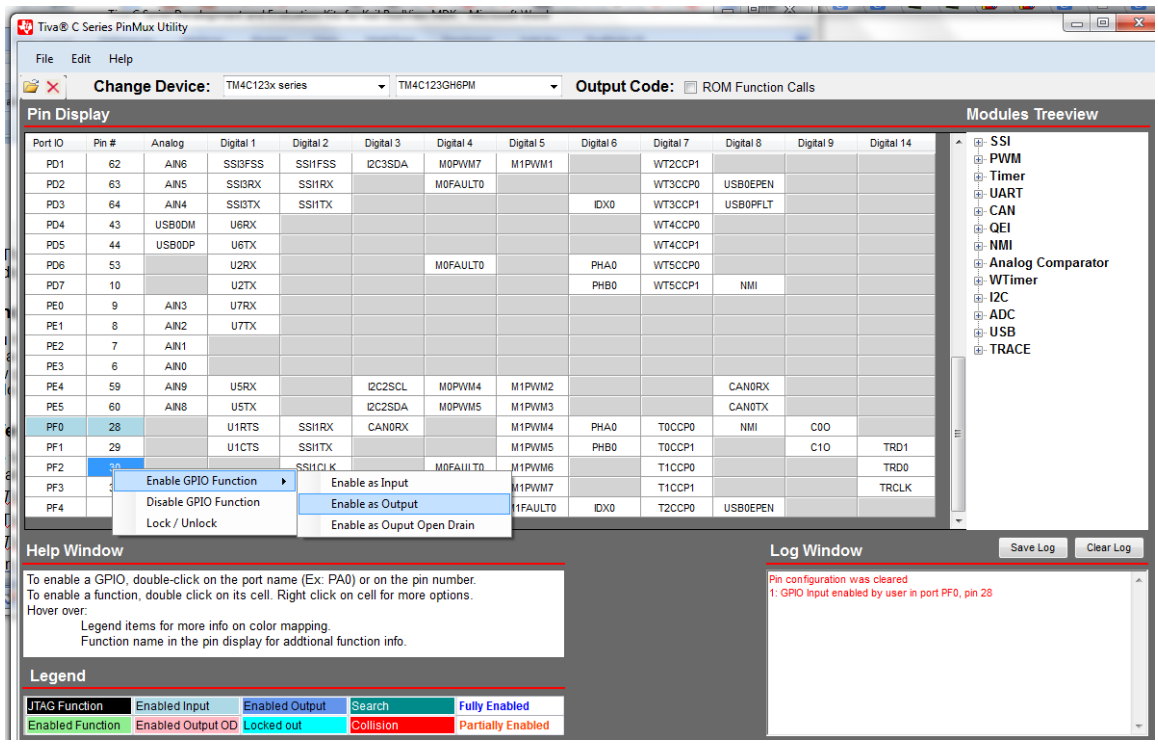
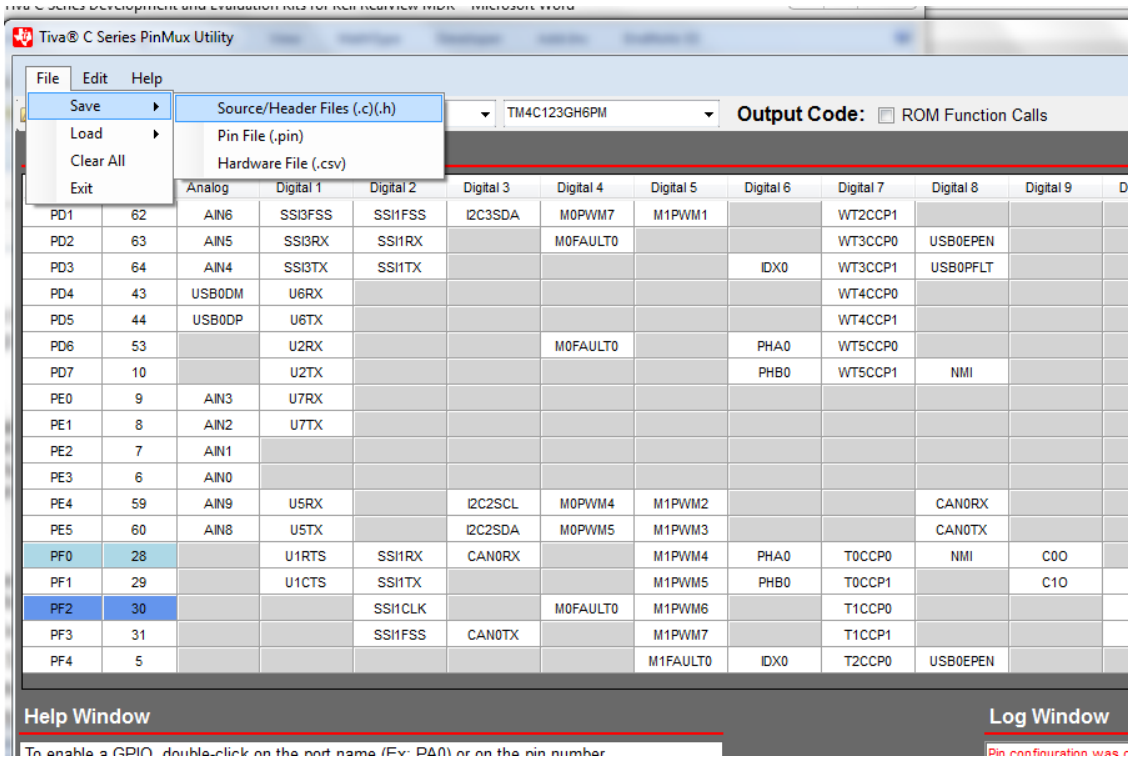


Figure 5

6. Now you will see in the Log Window at the bottom right of the GUI, it shows that PF0 and PF2 have been configured (see Figure 6).

Figure 6

7. To generate the port initialization code, select File->Save->Source/Header File (.c)(.h) (see Figure 7). A "Save Registers" window will pop up. You then need to choose the directory you want to save the generated files and name the files. In this example, we name files as *switch_PinMux*.c and *switch_PinMux*.h, and save the files in the *switch_PinMux* project folder (see Figure 8).



		Analog	Digital 1	Digital 2	Digital 3	Digital 4	Digital 5	Digital 6	Digital 7	Digital 8	Digital 9
PD1	62	AIN6	SSI3FSS	SSI1FSS	I2C3SDA	M0PWM7	M1PWM1		WT2CCP1		
PD2	63	AIN5	SSI3RX	SSI1RX		M0FAULT0			WT3CCP0	USB0EPEN	
PD3	64	AIN4	SSI3TX	SSI1TX				IDX0	WT3CCP1	USB0PFLT	
PD4	43	USB0DM	U6RX						WT4CCP0		
PD5	44	USB0DP	U6TX						WT4CCP1		
PD6	53		U2RX			M0FAULT0		PHA0	WT5CCP0		
PD7	10		U2TX					PHB0	WT5CCP1	NMI	
PE0	9	AIN3	U7RX								
PE1	8	AIN2	U7TX								
PE2	7	AIN1									
PE3	6	AIN0									
PE4	59	AIN9	U5RX		I2C2SCL	M0PWM4	M1PWM2			CAN0RX	
PE5	60	AIN8	U5TX		I2C2SDA	M0PWM5	M1PWM3			CAN0TX	
PF0	28		U1RTS	SSI1RX	CAN0RX		M1PWM4	PHA0	T0CCP0	NMI	C0O
PF1	29		U1CTS	SSI1TX			M1PWM5	PHB0	T0CCP1		C1O
PF2	30			SSI1CLK		M0FAULT0	M1PWM6		T1CCP0		
PF3	31			SSI1FSS	CAN0TX		M1PWM7		T1CCP1		
PF4	5						M1FAULT0	IDX0	T2CCP0	USB0EPEN	

Figure 7

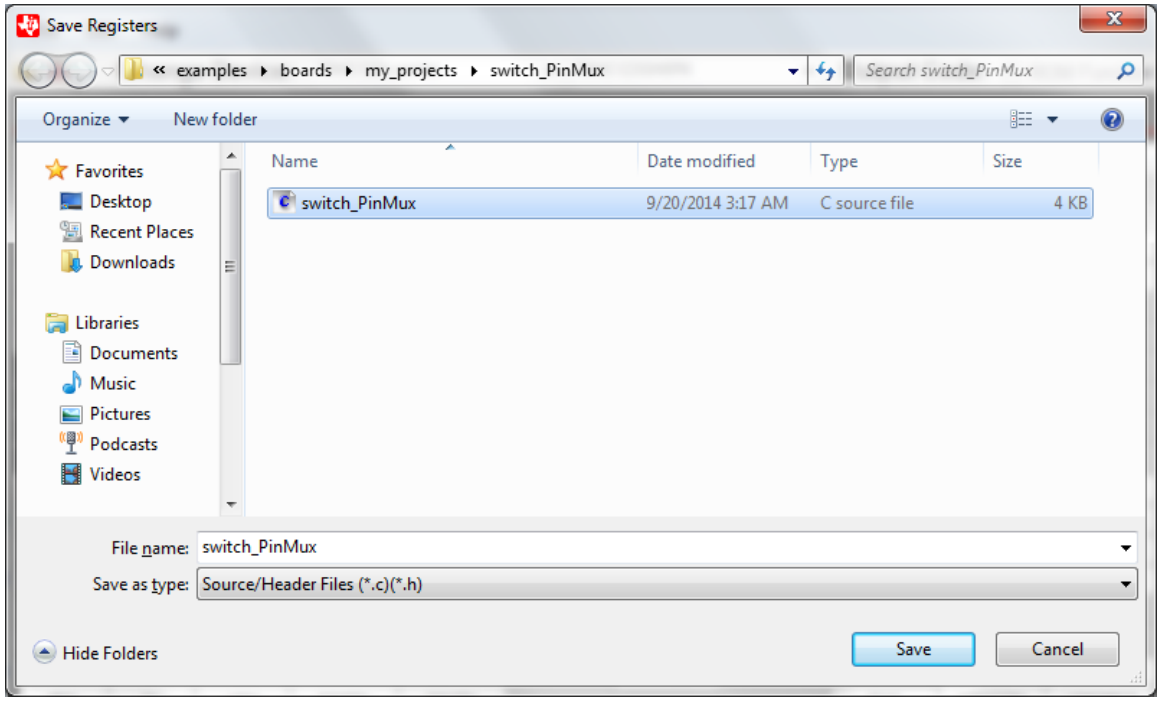


Figure 8

7. If *switch_PinMux.c* already exists, it will ask you whether to overwrite the existing file. If not, two new files *switch_PinMux.c* and *switch_PinMux.h* will be generated in the directory. The code in *switch_PinMux.c* includes a list of header files and a *PortFunctionInit()* function (see Figure 9). You can then start from here, add these two files to your project, and then add any additional lines of code necessary. You don't have to manually write C code to initialize the ports anymore. If you are configuring a GPIO pin to be an input pin for interfacing with the on board switches, you will need to manually add the code for enabling the pull up register.

stdint.h: Variable definitions for the C99 standard
stdbool.int: Boolean definitions for the C99 standard
hw_memmap.h : Macros defining the memory map of the Tiva C Series device. This includes defines such as peripheral base address locations such as GPIO_PORTF_BASE.
hw_types.h : Defines common types and macros
sysctl.h : Defines and macros for System Control API of DriverLib.
gpio.h : Defines and macros for GPIO API of DriverLib. This includes API functions such as GPIOPinRead and GPIOPinWrite.

Note: For any API functions provided by *driverlib*, you can refer to **TivaWare Peripheral Driver Library User's Guide** for detailed descriptions (can be found at C:\ti\TivaWare_C_Series-2.1.0.12573\docs\SW-TM4C-DRL-UG-2.1.0.12573.pdf).

```
#include <stdint.h>
#include <stdbool.h>
#include "switch_PinMux.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"

//*****************************************************************************
void
PortFunctionInit(void)
{
    //
    // Enable Peripheral Clocks
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    //
    // Enable pin PF0 for GPIOInput
    //

    //
    //First open the lock and select the bits we want to modify in the GPIO commit register.
    //
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x1;

    //
    //Now modify the configuration of the pins that we unlocked.
    //
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);

    //
    // Enable pin PF2 for GPIOOutput
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
}
```

Figure 9

Assignment

1. Create and implement two new projects *my_switch_PinMux* and *my_toggle_PinMux*, which perform the same functions as *my_switch* and *my_toggle* did, respectively. *my_switch* and *my_toggle* are the two projects you implemented in Lab 3. In *my_switch_PinMux*, when **SW2** (connected with **PF0**) is pressed, the **red LED** (connected with **PF1**) is turned **off**, otherwise the **red LED** is **on**. *my_toggle_PinMux* makes the **green LED** (connected with **PF3**) flash.

In this lab, you have to use PinMux to initialize the ports and use GPIOPinRead/GPIOPinWrite functions to configure the GPIO data registers.

In your lab report, for each project, please include the following:

- The C program listing with detailed comments for each line of code.
- The execution results of your program (your observations).