Week 2 – An Integrated Development Environment for Embedded Systems - TI's Code Composer Studio (CCS)

This handout for Week-2 is intended for ENCE361 students and complements code and other associated documentation that will be useful to help you complete your embedded systems project. This document outlines a recommended procedure for your second laboratory week and assumes you have either purchased a Tiva Kit (from the CopyCentre) and brought it to the lab, or that you are working with another student who has.

1. Laboratory set and group project

During Weeks 2, 3 and 4 you will have an assortment of laboratory exercises to complete. These will help you to complete your first Project Milestone in Week 5.

The set of three prescribed laboratories is designed to assist you by providing instruction and exercises on: (i) a professional development environment and commercial toolchain, (ii) familiarisation with several peripheral modules, such as the ADC and accelerometer, and (iii) introduction on the TivaWare API and Orbit BoosterPack display (daughterboard) to help students write C code to complete their development.

2. Laboratory books

Record notes from your labs over Weeks 2 and 5, as well as notes for your project, in an individual lab book. Record in your book which tasks have been completed, results and where you have found information that you needed to complete tasks, e.g. the location of important information in the datasheets and manuals. A soft-covered exercise book, such as a Warwick A4 Exercise book is ideal for this.

3. The Tiva LaunchPad & Orbit BoosterPack

The Tiva LaunchPad is a small, self-contained development board made by Texas Instruments. The specific board you will be using for this project features the TM4C123GH6PM Cortex-M4 microcontroller (MCU) [1], which supports two user-programmable push-button switches (SW1, SW2), an RGB LED, a number of peripherals (ADC, timers, etc.) and a USB interface. Fitted atop the Tiva is a *daughterboard* (top printed circuit board) that extends the functionality. This daughterboard is referred to in the literature as an Orbit *BoosterPack*, and supports a digital display and potentiometer that you will use in future weeks.

Documentation on the TM4C123 microprocessor and the Code Composer Studio (CCS) tool-chain, in addition to application notes and errata, can be found on your course Learn pages.

4. Learning Outcomes

- Develop confidence to build and modify a simple microprocessor application.
- Use a debugger to help find and correct programming faults.

Your objective this week: *Understand the fundamentals of Code Composer Studio and locate and use API functions to help you modify provided C code using the GPIO peripheral.*

5. Recommended Procedure

5.1. CCS Tutorial

In the *Projects and Laboratories* page on the ENCE361 Learn site you will find a folder, *Week-2*, containing a PDF tutorial for Code Composer Studio: **CCS 9_2_0 Tutorial.pdf**. Closely carry out the procedures outlined in this tutorial. **NB**: It is important to keep the same directory structure throughout your lab work. In later labs you will be able to define your own project directory but for now, follow the instructions. Completion of this tutorial should take between 30 and 50 minutes.

5.2. Analysing the source code – `week2_blink.c`

After building your first CCS project you should study the source code. Find and study the `main` function in *week2_blink.c*, which can be downloaded from the *Laboratory source code* folder on the *Projects and Laboratories* page on Learn. Note how variables are defined. We recommend using the unsigned integer types, such as **`uint8_t, uint32_t`**, etc., similar to what was used in ENCE260, **not** the standard C language types, e.g. `unsigned`. These definitions are in *stdint.h.*
As with most embedded systems programming, there are a number of initialization tasks to perform. Your first programs in this course will have more lines of code associated with initialization than for anything else! In particular, note the API[1] functions that are called. The documentation that defines API functions used with the Tiva can be found in two documents:

1. *Tiva Launchpad User Guide*, [2] and,
2. *TivaWare Peripheral Library User Guide* [3] (API function guide)

As you progress, you will find the following document also useful:

3. *Orbit BoosterPack Reference Manual* [4].

These are on the *Data Sheets and Manuals* page on Learn. The *Launchpad User Guide* defines hardware resources supported on your Tiva board (like switch button details). The *Peripheral Library User Guide* defines API functions for your MCU peripherals, such as analogue-to-digital converter (ADC), timers, GPIO, etc. The *Orbit BoosterPack Reference Manual* outlines the electrical connections and port addresses for you OLED display. You will routinely need to consult these documents as well as the processor datasheet [1] from now on.

5.3. Running the initial source code – `week2_blink.c`

Currently, the only purpose of the *week2_blink.c* program is to flash the LED on the Tiva board. Thus, it is a relatively simple program.

As outlined in the CCS tutorial guide that you completed before starting this part of the lab, add a new project to your existing workspace and call this `Week2_Lab`. Copy the *week2-blink.c* source from Learn into your project directory. Set up the compiler and linker settings, identical to that outlined in the CCS tutorial. After completing this, your program should compile, link and load

[1] Application Programming Interface

into your Tiva board. Push the green 'execute' arrow to start it running. Refer to the CCS tutorial guide if you get stuck.

5.4. Modifying the code

For this lab you are required to modify the source code so that the blinking of the LED will only occur once you press the pushbutton switch SW1, and continue only while pushbutton SW1 is pressed. Sound familiar? Well, you have to start somewhere! There is plenty of scope for making more interesting mods, if you have the time and inclination.

Important: Rather than change the code within the same file, make a new copy and rename it; make your changes in the new file. This is a simple form of source control; once your group's **git** repository is established, get into the habit of using it for your group's source file control.

Once you have more than one *.c file in your project, the compiler needs to know which to compile (it knows that there should only be a single **`main()`**): To fix this, exclude the original file from your build: right-click over the file name in the Project Explorer window and Exclude from Build. Of course, as your code base grows, other .c files may be required that do need to be in the build.

Look through the *Tiva Launchpad User Guide* (*UG*) for port labels used in your program to initialize SW1 as an input, just after, for example, line 44. Once you determine the appropriate port letter, e.g., E, F, or G, and the specific pin number on that 8-bit port, use an appropriate GPIO pin function to set this port pin as an input, and secondly, configure the specific GPIO pin to draw 4 mA through the internal pull-up (remember what these are?). Lastly, around Line 64, write a simple **`if`** construct that tests the port pin, such that the LED only blinks as long as switch SW1 is pressed.

6. Further program development, debugging and testing

As mentioned, TivaWare is an API library and it is intended to help make programming easier by effectively using pre-written functions. The Texas Instruments compiler (based on **gcc**) has been installed on the hard disk drive of each PC host in the Electronics Lab. Similar to the AVR compiler you used in ENCE260, this program compiles and links ARM code. When built, the resulting ELF binary file can be downloaded to your Tiva board by selecting the debug icon. An in-circuit debugger is integrated within CCS and this can be used to test and debug software. For example, set a breakpoint just after the variable **`clock_rate`** is assigned a value. Do this by double clicking on the blue region on the extreme left within your source code window. A blue icon should appear on the left of the line of source you clicked on to show the breakpoint. Next, set up a watch expression or variable for **`clock_rate`**. Run your debugger to a breakpoint that you have set just past your **`clock_rate`** variable declaration. Note the value for **`clock_rate`**. Does this value agree with your expectation?

Now, experiment by altering the code in other ways, recompile and load the program, and test the result. Suggestions include:

A. See if you can use different colour LEDs for your display.

B. Look at the setup used for the CPU clock. Once set up, note that SysCtlDelay(arg) uses a magic number (yes, you can improve on this code) in calculating the parameter. What is the original blink rate? – this can be found without testing. If you increase this number will the flash period be extended or will it reduce? Hint: Refer to SysCtlDelay(arg) in the *TivaWare Peripheral Driver Library* (use search in the PDF) to find what this function does and the relationship to arg. Why is it written in assembler?

C. Can you explain the warning given in the TivaWare peripheral library user guide for the application of SysCtlDelay()?

D. What is the duty cycle of the LED flasher? What code modifications are needed to change this to, say, a duty cycle of 20%? Considering what you now know about SysCtlDelay(), calculate two argument expressions to achieve a duty cycle of 20% while the blink rate stays the same.

7. References

[1] *Tiva TM4C123GH6PM microcontroller datasheet*, Texas Instruments, SPMS376E, 2014

[2] *Tiva C Series TM4C123G LaunchPad Evaluation board*, Texas Instruments, SPMU296, 2013.

[3] *TivaWare Peripheral Driver Library User's Guide*, Texas Instruments, SPMU298D, 2016.

[4] *Orbit Boosterpack Reference Manual*, Digilent Inc., DOC# 6032-502-000, 2013.