EGH456 Embedded Systems

Laboratory Exercise 1

Introduction to Embedded Systems Programming

Learning Objectives

Upon completing this laboratory exercise and reviewing the accompanying documentation, you will be able to:

- Utilize Texas Instruments Tiva C Series boards to explore embedded application programming.
- Develop applications using Visual Studio Code in combination with PlatformIO.
- Compile, execute, and debug *C* programs.
- Modify and run sample programs.
- Comprehend the overall process of embedded application development.
- Understand the interplay between hardware and software at the device level.
- Work with *C* macros.
- Distinguish between direct register-level hardware access and access via the peripheral driver library.

Reference Documents

The primary textbook for this unit is *Fundamentals of Embedded Software with the ARM Cortex-M3.* This text covers essential theoretical concepts and outlines the general process of embedded systems programming using *C* and *assembly.* However, since it is based on the Cortex-M3 architecture, it does not address the specific hardware platform used in this laboratory. Because the hardware and software development platforms provided here are more advanced, you are encouraged to supplement your learning by consulting the technical documents related to the laboratory component. Additional recommended textbooks include:

- *Real-Time Interfacing to ARM Cortex-M Microcontrollers* by Jonathan W. Valvano (Vol. 2, 2014)
- *Fundamentals of Embedded Software – Where C and Assembly Meet* by Daniel W. Lewis, Prentice Hall

- *Real-Time Embedded Systems* by J. Wang (2017), Hoboken, NJ: Wiley.
- Gadre, D. V. & Gupta, S. (2018) *Getting Started with Tiva ARM Cortex M4 Microcontrollers: A Lab Manual for the Tiva LaunchPad Evaluation Kit* (1st ed., 2018) [Online]. New Delhi: Springer India.

The TI Tiva C Series EK-TM4C1294XL boards, which feature the Cortex-M4 architecture, are equipped with peripherals such as LEDs, push-button switches, a USB interface, and an Ethernet interface. In addition, these boards support application-specific *booster pack* plug-in modules that enable rapid prototyping for a wide range of applications, including capacitive touch, wireless sensing, and LED lighting control. Numerous online resources, including video tutorials, are available on the unit website.

Detailed information about the processor and the application development system is provided in technical documents from the vendor, which have been made available on the Canvas site for this unit. The following documents are especially useful for getting started:

- PlatformIO IDE for VSCode
- Cortex M4 Technical Reference Manual
- EK-TM4C1294XL Development Board User's Guide
- TM4C1294NCPDT Microcontroller Datasheet
- TivaWare Peripheral Driver Library User's Guide

Many books on *C* programming are available from the QUT library. For example:

- *C Programming for the absolute beginner* by Michael A. Vine, Permalink, 2007.
- *Further Topics in C Programming* by Gookin Dan, Permalink, 2015

Preparation Activities

Make sure that you can identify and use the following at your workstation in the laboratory.

- The Tiva C Launchpad development boards.
- The host PC and its USB connection to the board.
- Software installed on the host PC – including Visual Studio Code.

Open the key technical documents previously mentioned either from the links and browse through them such that you are aware of their contents and can look up necessary information when required. These documents are also available on Blackboard under Learning Resources → Labs → Lab1.

Assessment

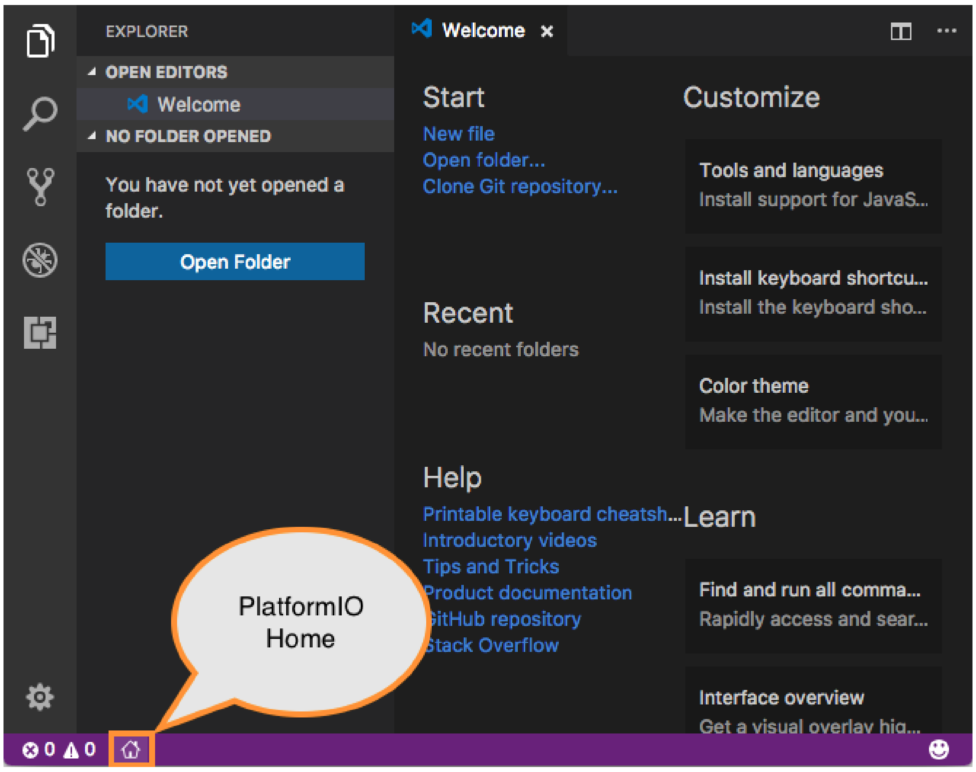
Show evidence of having performed the modifications and run the programs as required to the tutor. Answers to questions must also be written down and shown in class. You may work in a group of 2 (or 3 with permission from the tutor if there is a need for it). Each member must be present in the laboratory at the workstation and be participating in the exercise to receive credit.

Laboratory Tasks - Setup

Open Visual Studio Code. The first thing you should do is install the PlatformIO IDE extension and familiarise yourself with the software. The quick start guide provides details on installation of the software and example workflows. Seek help from a tutor if there are any issues with installation.

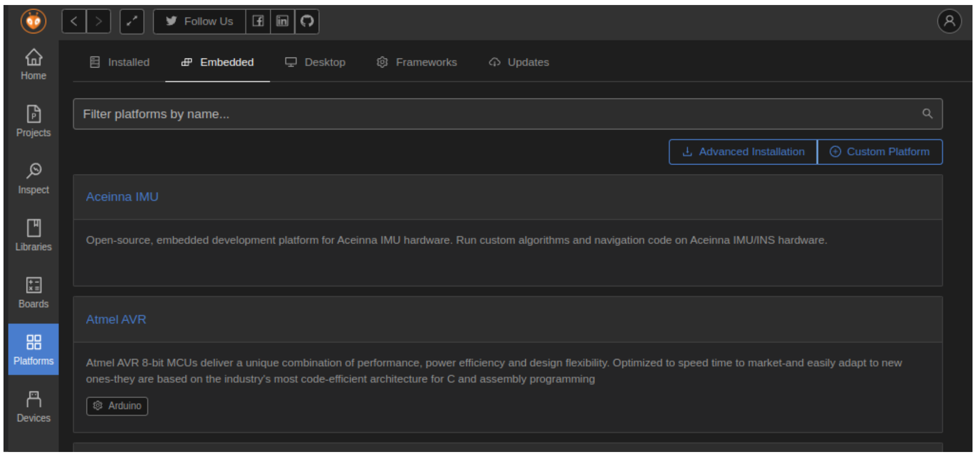
Next, we need to install the TI TIVA platform. This enables PlatformIO to build and run code on Tiva C Series boards. To install the TI TIVA platform:

1. Open the PlatformIO Home by clicking the home button on the bottom toolbar



You have not yet opened a folder.

2. Navigate to the 'Platforms' and then 'Embedded' tabs



Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's most code-efficient architecture for C and assembly programming

3. Search for 'tiva' and install the TI TIVA platform

Some systems require the further installation of the USB drivers needed to upload code to the embedded system. If so, download the spmc016a.zip file from the computer lab page and follow these steps:

1. Unzip the files and locate the **stellaris_icdi_debug.inf**, **stellaris_icdi_com.inf** and **boot_usb.inf** files

2. For each **.inf** file right click and select install
3. Follow the prompts to complete the installation of the usb drivers
4. Unplug and reconnect your device to ensure USB drivers are reloaded

Next, we need to download the TivaWare Software Development Kit (SDK) from the computer lab canvas page or using this link. Specifically, download the installer *SW-EK-TM4C1294XL-2.2.0.295.exe* which contains the latest software for the TI Tiva C series EK-TM4C1294XL boards. You may need to make a free account to proceed. Go through and run the .exe installer and check where the SDK is installed on your computer, as you will need this information later.

Lastly, we need to be able to import an existing PlatformIO project to run on the board. For most of this unit, the teaching team will provide example projects that form the basis of the laboratory exercises. These projects have already been configured to run on the TI Tiva C series EK-TM4C1294XL boards. If using for the first time it is recommended that you create a workspace folder (i.e. egh456_workspace) in your student (H:\) drive. Next download the example project *blinky* from the workshop page on canvas and export it to your workspace folder. To import this project into PlatformIO:

1. From PlatformIO home, click 'Open Project' and navigate to the *blinky* project folder
2. Open *platformio.ini* and inspect the file. This contains all the configuration settings for the PlatformIO project. The only thing you need to change are the include paths, which should point to where the TivaWare SDK has been downloaded on your system.

PlatformIO Tips and Tricks

There are a number of common issues faced when using PlatformIO and VSCode:

1. **Include path syntax**. Ensure only single forward slashes are used when declaring the include path in the *platformio.ini* file.
2. **Nested folders.** Unzipping projects can result in nested folders of the same name on some systems. PlatformIO requires the *platformio.ini* file to be in the root directory of a project, so you must be sure to import the inner most folder.
3. **Importing projects.** Be sure to import project folders using the PlatformIO home page, as described in this document. If you use File -¿ Open Project functionality of VSCode, the project will not be initialised with PlatformIO.

4. **Hanging debugger process.** Attempts to launch debugger mode from VSCode can generate a hanging PlatformIO process. If this occurs, you will get an error stating that the file *firmware.elf* cannot be accessed. Restarting VSCode does not solve this problem, and instead you need to kill the platformio.exe process manually (either using task manager or command line interface)

Task A (2 marks)

First download the *blinky* example from the computer lab canvas page and then unzip and import it into platformIO using the open project button on the home page. Make sure to import only the inner/second folder of the zip file when importing the project. Once imported, ensure the project is selected on the bottom toolbar and then click the tick symbol to build the project. After it has compiled, go to the Debug menu on the side toolbar to run the project. Again, ensure the correct project (blinky) is selected and click the run and debug button to launch debugging mode. Alternatively, you can use the menu and select "run/start debugging" or press the shortcut F5 to start debugging. Once platformIO has uploaded the program and switched to debugging mode you can press the play (blue triangle) button to run the code. A blinking LED should confirm that the programming is running.

Note: This program includes the *Software Driver Model* as opposed to the *Direct Register Access Model*. Section two of the TivaWare Peripheral Driver Library User's Guide provides insight into how the two models are different.

Explore the project folder for this example and find the header files and libraries that are included. Open and inspect the *C* source file *blinky.c* and answer the following questions:

1. What are the data types and sizes of the arguments used in the GPIO pin write function for the example program? (Hint: Check definitions in the included header file) (**0.4 marks**)
2. Identify at least one macro used in this program. Where is it defined, and what does it expand to in terms of actual memory addresses? (**0.4 marks**)
3. What is the base memory address of the register controlling the LED port in this program? (Hint: Refer to the relevant header file.) (**0.4 marks**)
4. How is a time delay created between the LED ON and LED OFF states in this program? (Check the source code) and if you want the LED to blink twice as fast, which part of the code would you change and what value would you use? (**0.4 marks**)

5. Suppose the LED fails to blink after you upload the code. List at least two potential causes (hardware or software) and how you would debug each one. (**0.4 marks**)

Task B (1 Mark)

Download the example project *grlib_demo* from the workshop page on canvas and export it to your workspace folder. Explore the project folder and examine the header files, the driver libraries, and the *c* source code.

Modify the *grlib_demo.c* program to complete the following:

1. Change the banner text from "grlib demo" to your name and student number. For example: "Ashley n9123456". (**0.5 marks**)
2. Change the banner to a different background colour and the banner text to a different font colour (**0.5 marks**)

Build, run and test your changes. You will need to find the function that can do this and change a parameter as required. Show the source code and demonstrate your working solution to the tutor.

Task C (3 Marks)

Download the example project *blinky_direct_register* from the computer lab page on canvas and export it to your workspace folder. This code uses the *Direct Register Access Model*(DRAM) as opposed to the *Software Driver Model.* Run the program and understand what the code is doing.

Rather than toggling just one or two LEDs, modify the direct register access "blinky" program to cycle through all four on-board user LEDs in a repeating sequence one after each other. Follow these steps as a guide:

1. There are four user LEDs on the TM4C1294XL board. Find the port and pin numbers associated with the remaining two LEDs (**hint**: you could try using search in the EK-TM4C1294 User's Guide, or the TM4CNCPDT Datasheet). (**1 mark**)
2. Enable each of those ports using direct register writes and set each LED pin as a digital output. (**1 mark**)

3. Once each LED pin and port are set up, create a sequence (e.g., LED D1 on, then off, then LED D2 on, then off, and so on) with a short delay between each LED. Repeat this sequence indefinitely, so the LEDs appear to chase or cycle across the board. (**1 mark**)

The TM4C1294NCPDT Microcontroller Datasheet, section 5 and 10, as well as EK-TM4C1294XL Development Board User's Guide section 2.1.5 will be particularly useful for this exercise. If you right-click the entry "SYSCTL_RCGCGPIO_R" in the example code, and click "Open Declaration", you will be taken to a header file. Judicious use of CTRL-F and reading the code comments should allow you to modify the example code to enable the other LEDs on the board. **REALLY BIG HINT**: The problem-solving task for this unit will frequently require you to drill into both the documentation and the header files in this way - get some practice in!

Build, run and test your program. Show the source code and demonstrate it to the tutor. Part marks will be awarded if you can demonstrate partial functionality.

Task D (2 Marks)

The previous lab tasks required using certain GPIO pins as outputs to drive LEDs. GPIOs can also be used as inputs. The TM4C1294XL development board has four switches, two of which are assigned as user switches. You may use either the software driver model or direcr register access model. We recommend the software driver model as typically this is the easiest approach.

1. Configure the two user switches as inputs and verify that each switch press can be detected in your code. You can do this, for instance, by toggling a single LED whenever a switch is pressed. (**1 mark**)
2. Configure the two switches to toggle the LEDs to follow the pattern that you completed in the previous task. One switch should toggle the pattern in the forward order of the pattern and the second in the reverse order. If you did not complete Task C then you may choose a simpler pattern with two LED's but must show some change in the pattern when a different user switch is pressed (i.e. forward and reverse). (**1 mark**)

Hint: You will need to determine which port and pins the two user switches are connected to, afterwhich it should be straightforward to adapt the code from Task A of Lab 1 to enable the switches, and write a simple program to activate the LEDs. If you like, you could also adapt the DRAM code from Task D of Lab 1; the end result will be the same. At some point in your project you may find that a Digital

Storage Oscillscope (DSO) can be a handy tool for hardware debugging. If you're not familiar with the use of DSOs, this task might be a good opportunity to practice. Use the DSO to observe the behaviour of your switches and other GPIO ports.

Optional Extension Task: Switches are electro-mechanical devices, thus they can potentially be subject to electrical noise when used, resulting in "bouncing", which can cause erratic switch behaviour. As an additional optional task you could research (using your favourite search engine) then implement an appropriate method (i.e. in software) for debouncing switches.