18-847SH: Wireless Software Systems Architecture

Lab 2: Getting familiar with PowerDue microcontroller

Fall 2019

Lab Due Date: 10/02/19

Introduction

The labs in this course are designed to familiarize yourself with the tools and techniques for developing software for wireless embedded systems. The objectives of this lab is:

- Learning to work with the PowerDue
- Appreciate the synergy between software and hardware platforms
- Working with on-board components through corresponding software libraries
- Introduce yourself to FreeRTOS (using the library in Arduino IDE)
- Practice writing technical reports

At the end of this lab, you are expected to get comfortable writing and deploying code on the PowerDue. While working with embedded systems, looking up schematics, components, libraries and documentation should be second nature to you. You are also expected to write technical reports and express your analysis on what you've learned rather than just writing what was done.

All the setup for this and future labs has been done for you. We have put together three desktops booted with Linux OS exclusively for programming the PowerDue. Read and follow the below rules for accessing the desktops: (Rules are valid until end of course)

- Login using username: **WssaFall19** and Password: **wss@fall19**
- Feel free to use Terminal for navigating, however do not use prefix **sudo** for any of the terminal commands, you are not provided with sudo access.
- Do not unplug PowerDue boards from the machine, if there is a pressuring need, contact the TA.
- Do not install or attempt to install any third party software. All the required softwares for every lab will be given to you.
- You are allowed to save helper files (like PDFs, datasheets). You can save your code progress by pushing your incremental changes to your CMU GitHub repo and **delete** all your main code after you are done, before you log-off.

- You can clone your remote GitHub repo after every log in and continue where you left off. Using Email to transfer code is strictly discouraged.
- Always log-off, do **not** shutdown the system. Also, avoid the use of lab desktops for personal work.

The PowerDue is a customized ArduinoDue and may be programmed using the same tools as any Arduino microcontroller. We will be using the Arduino IDE to program the PowerDue Microcontroller using Arduino programming syntax which is similar to C/C++. The following section introduces you to Arduino programming. You are neither required nor allowed to connect PowerDue boards to your personal laptops. PowerDue boards are very sensitive, handle with care. Refer to this document that highlights the rules of regulations of using PowerDue boards.

Part 1: Introduction to Arduino Development

Arduino is a platform that eases development for various microcontrollers. Adopting its software platform allows us to write code for different boards by abstracting most of the common details, but still allowing full control over the software being pushed into the microcontroller.

- Open Arduino IDE, by typing **$ *~/arduino*** command on the terminal.

Deploying your first sketch on arduino
That's it!, all the setup required for the lab has already been done for you. Feel free to refer to some arduino tutorials to get an idea about arduino programming (the built-in example programs are also a great place to start, but do not run any of them as they are not customized for PowerDue board). You are now ready to deploy software on the PowerDue.The following example introduces deploying code onto the device and controlling the board's LED array.

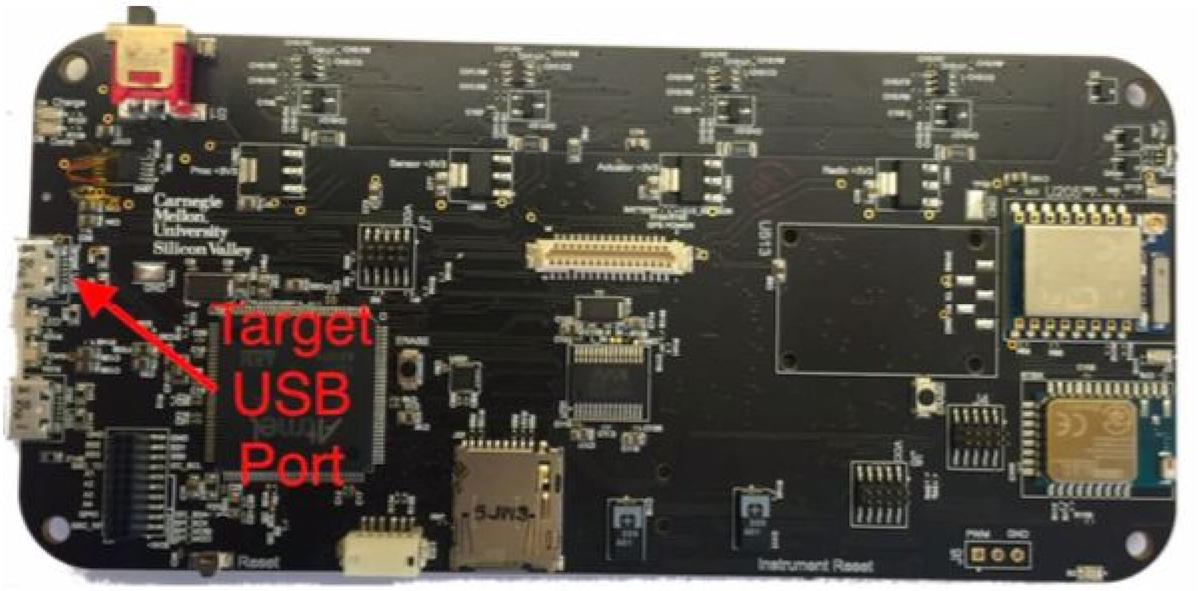
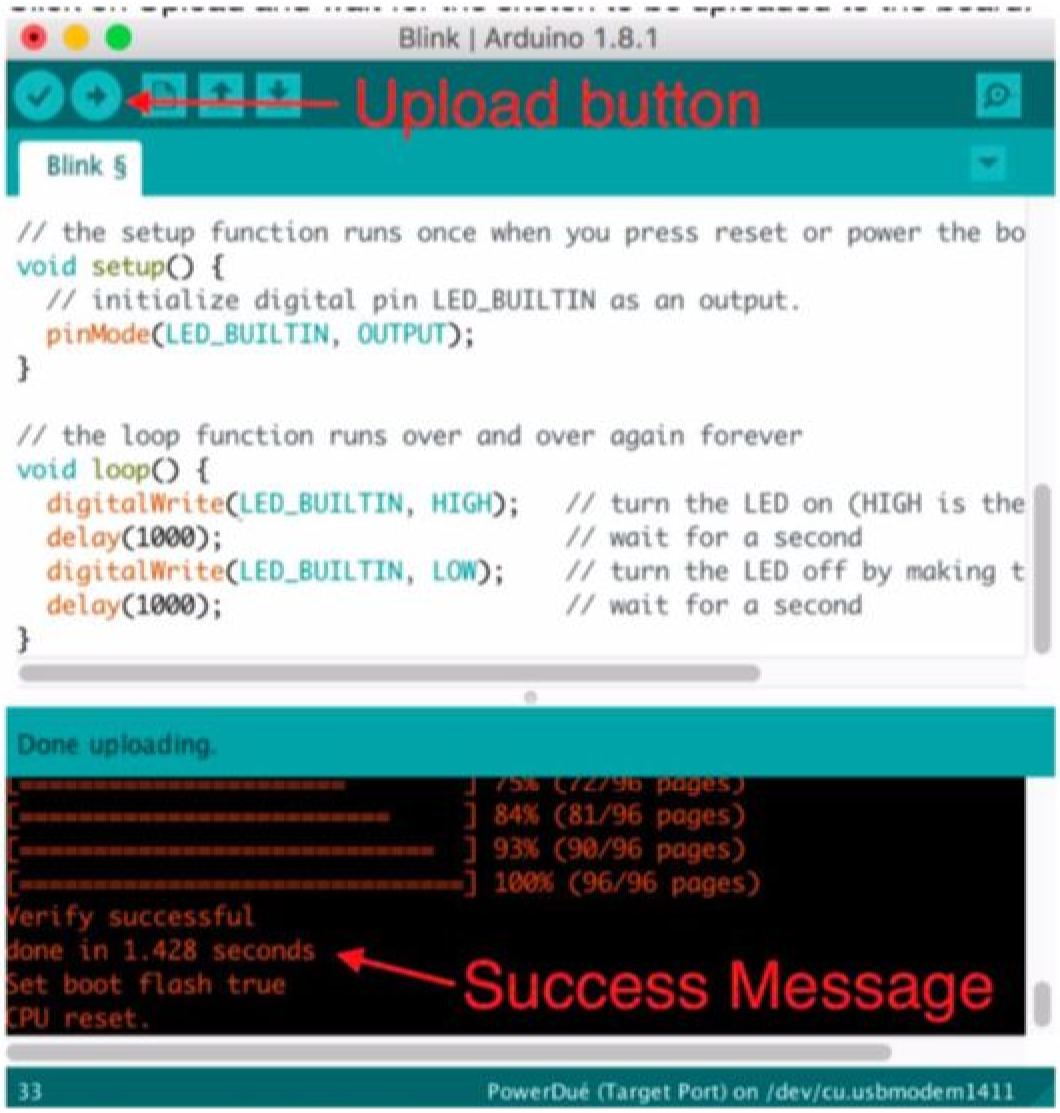


Figure 1: PowerDue board has *target* and *instrument* ports to flash program onto, always flash to target port only

Turn on the PowerDue.

• Under Tools > Port, select the appropriate USB port (usually titled as *PowerDue Target*) where the PowerDue is connected
• Under Tools > Board, select PowerDue (Target Port)
• Open up the Blink Example (File > Examples > 01.Basics > Blink)
• Click on Upload and wait for the sketch to be uploaded to the board.

Once it succeeds, the board will start running the example code.



```
// the setup function runs once when you press reset or power the bo
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making t
  delay(1000);                       // wait for a second
}
```

Done uploading.

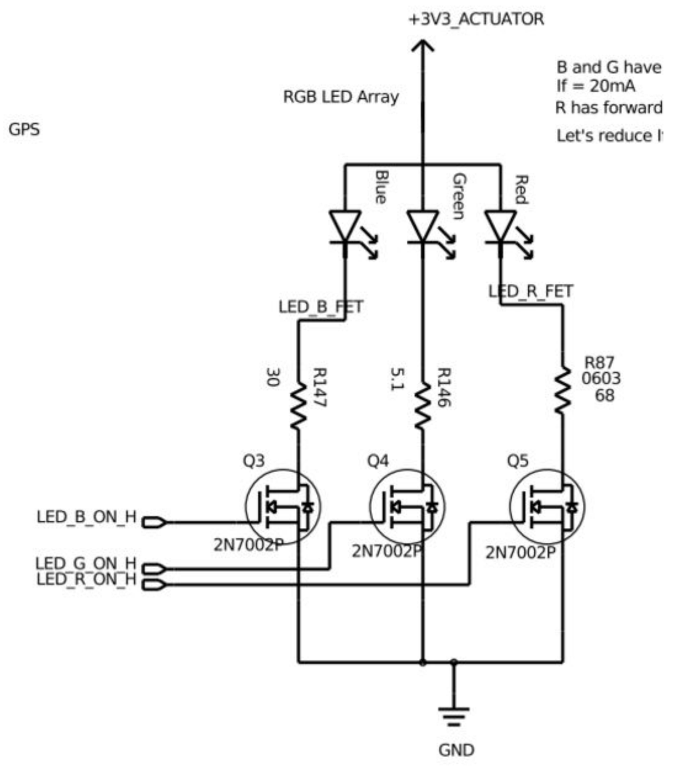
```
[======================        ] 75% (72/96 pages)
[=========================     ] 84% (81/96 pages)
[============================  ] 93% (90/96 pages)
[==============================] 100% (96/96 pages)
Verify successful
done in 1.428 seconds
Set boot flash true
CPU reset.
```

33   PowerDué (Target Port) on /dev/cu.usbmodem1411

Figure 2: Snapshot of Arduino IDE after uploading the code. Ensure the correct port and board are chosen before flashing the code

- **Why does this example not work for the PowerDue?**

The default Blink example toggles an LED pin on and off every second. Most Arduino boards have an LED connected to a digital pin denoted by the macro "LED_BUILTIN". However for the PowerDue, LEDs are mapped to different pin numbers. The PowerDue has an RGB LED array as indicated in the schematic below.



With access to the full schematic of the PowerDue, try to address the following questions:

- Which Arduino pins are connected to the LED array? (*Hint: they are PWM pins*) ?
- Can you get the Blink example to work on the PowerDue?
- Can you modify the Blink example such that you toggle the Red LED twice every second?
- How about toggling it Green or Blue?
- Can you toggle the LED to be White?
- What other colors can you generate?

Making it Interactive

In the example, the PowerDue is only able to blink the LEDs at pre-programmed time periods. It cannot take inputs once the program starts running to vary the output. **Can you make it interactive so you can have more control over the LEDs and choose any one of the colors for blinking through user input?** For example the user input "blue" must cause the LED to blink red every 2 seconds.

Data can be sent to the PowerDue over USB using the Arduino IDE's serial monitor (button on the top right of the Arduino).

Part 2: Getting started with FreeRTOS

With the above exercise, we saw how inputs and outputs work with the PowerDue. A real-embedded system consists of many different software modules that are working with each other. Given the limited amount of computational resources available on an embedded system, a scheduling mechanism is needed to access these resources. A synchronization strategy for shared data access and modification is also required. To handle these problems while avoiding the complexities of full-fledged operating systems, embedded systems use lightweight real-time operating systems.

FreeRTOS is one such RTOS which provides methods for multiple threads, mutexes, semaphores, software timers and thread priorities. The RTOS only consists of 3 C files. You can find the entire API documentation for the RTOS here.

Deploying your first sketch with FreeRTOS

Let us run the Blink example using threads on FreeRTOS. Modify the blink example to run it using FreeRTOS. Create a task for blinking the LED. Upload the sketch on the board to run it. Note: The loop function should now be empty because all tasks will be managed in FreeRTOS threads.

- What priority level and stack size did you decide for running this thread?
- Bonus Question: What is the minimum stack size which can be set for a task on the PowerDue? Can you find its value?
- Bonus Question: Can you determine the tick rate for the FreeRTOS version running on the PowerDue?

Managing tasks and resources

Once you can answer these questions, we will dive further into FreeRTOS to learn about task management.

- Create individual threads for blinking each possible color combination of LEDs on the PowerDue. Each thread should look something like this:

```
//------------------------------------------------------
/*
 * ThreadRed, turn the Red LED on for 500 milliseconds.
 */
static void ThreadRed(void* arg) {
  // Turn RED LED On.
  SerialUSB.println("Turning the Red LED on");
  pd_rgb_led(PD_RED);

  // Sleep for 500 milliseconds.
  vTaskDelay((500L * configTICK_RATE_HZ) / 1000L);

  // Turn RED LED off.
  SerialUSB.println("Turning the Red LED off");
  pd_rgb_led(PD_OFF);

  // Sleep for 500 milliseconds.
  vTaskDelay((500L * configTICK_RATE_HZ) / 1000L);

  // Done with the thread
  SerialUSB.println("Done");
  while(1);
}
```

- Assign the same priority to each task.
- Flash the sketch onto the board, open the serial monitor and observe the output. What do you observe?

From the above example, you will notice multiple threads running concurrently with each other, with no control over the sequence of execution. What would have happened if each of these threads was trying to access a memory location?

To fix the problem,it was suggested that each of the threads be modified as follows:

```
//----------------------------------------------------
/*
 * ThreadRed, turn the Red LED on for 500 millisecond
 */
static void ThreadRed(void* arg) {
  // Wait until a semaphore is released
  xSemaphoreTake(sem,portMAX_DELAY);
  // Turn RED LED On.
  SerialUSB.println("Turning the Red LED on");
  pd_rgb_led(PD_RED);

  // Sleep for 500 milliseconds.
  vTaskDelay((500L * configTICK_RATE_HZ) / 1000L);

  // Turn RED LED off.
  SerialUSB.println("Turning the Red LED off");
  pd_rgb_led(PD_OFF);

  // Sleep for 500 milliseconds.
  vTaskDelay((500L * configTICK_RATE_HZ) / 1000L);

  // Done with the thread. Release Semaphore
  SerialUSB.println("Done");
  xSemaphoreGive(sem);
  while(1);
}
```

The semaphore was declared in the setup function as:

```
// initialize semaphore
sem = xSemaphoreCreateCounting(1, 0);
```

Modify the threads as above and try to run the program.

- Does it work as expected? If not, can you find the problem in the above code?
- Is the sequence of colors always the same?
- Can you modify this program to always blink the LEDs in the following sequence?

  Red -> Green -> Blue -> Red + Green -> Red + Blue -> Green + Blue -> Red + Green + Blue

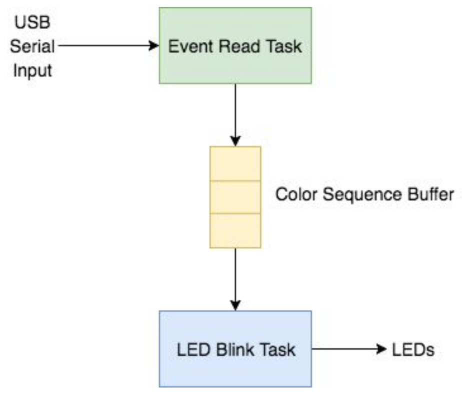
- How can you loop over this sequence?

Managing data sharing between tasks

Write a program in FreeRTOS that reads an input sequence of characters and flashes LEDs in that sequence. Define one character for each color combination you want. As shown in the block diagram below, design two tasks for your system. One task should read the input buffer sent from the keyboard and the other task should analyze the buffer and determine the corresponding light sequence.

For example: an input sequence of RGBW could toggle the LEDs in the following sequence: Red, Blue, Green, White.

- Can you use global buffers to store and share data between the two tasks?
- Why would you use/not use a global buffer to store data?
- Is there a better way to do this?



Part 3: Submissions

Submit a report in five pages or less answering all the questions by explaining them with relevant screenshots and/or flow charts. Reports are required to be in **Latex** format. Refer to this link to get started with latex on Overleaf. Do not forget to mention a **link to your github repo**. This lab requires **individual** report submissions.

Grading Rubric:

Part 1	Deploying your first sketch	- 10 Pts (Code + answers to questions)
	Making it interactive	- 10 Pts (Code + answers to questions)
Part 2	Deploying your first sketch with FreeRTOS	- 10 Pts (Code + answers to questions)
	Managing tasks and resources	- 20 Pts (Code + answers to questions)
	Managing data sharing between tasks	- 30 Pts (Code + answers to questions)

Latex Report writing and styling	- 10 Pts
Github Repo	- 5 Pts
ReadMe	- 5 Pts

Your github repo should ideally have five **.ino** files. The files should be titled as part1_1.ino, part1_2.ino, part2_1.ino, part2_2.ino, part2_3.ino. Ensure if your code is working before submitting the git link.