

Lab1: The Beginning

This lab is an introduction to the STM32 F3 Discovery board and the tools we will be using throughout the course.

Assigned: September 25, 2017

Due: Week of October 2, 2017

Part 1

Hello World!

In this section, you'll set up the tools and library code and build the existing demo project.

Tools

For building our projects we will use an embedded ARM toolchain paired with tools specific for interfacing with the Discovery board. The main programs we will be using are:

1. *arm-none-eabi-gcc*, the GNU C cross-compiler for embedded ARM processors.
2. *STLink*, tool to communicate with a ST microcontroller and program it.
3. *OpenOCD*, tool to remotely interface with microcontroller and allow remote debugging.

These tools are used by the provided Makefiles for each project. You should not ever have to use these tools directly, instead interfacing with them with `make program` or `make debug`.

The toolchain has been designed to be run on Ubuntu 14.04. For these labs we will be using a preconfigured virtual machine with all the tools installed. If you are using Mac or Linux install the latest version of [VirtualBox](#) to run the VM. On Windows use [VMWare Player](#). Ask the TAs for the prebuilt VM image and then follow the directions in the [Toolchain Tutorial](#).

Get the Code

You should have already cloned the Github repository to your local drive (if you used the prebuilt image this is in `~/green-electronics`). If not, use the following command to clone the repository `git clone https://github.com/ndanyliw/green-electronics.git`. Then enter the directory and you should find the Lab 1 starter code. You are encouraged to look around the directories especially the Lab 1 code and the provided libraries.

Connect the Hardware

At your station you should have a Discovery board as well as a breakout board. Plug the Discovery board into the breakout board and then connect the board to your computer. There are two USB connectors on the board - use the USB connector labeled 'STLink'. The board should power on and will begin running the last program it had loaded.

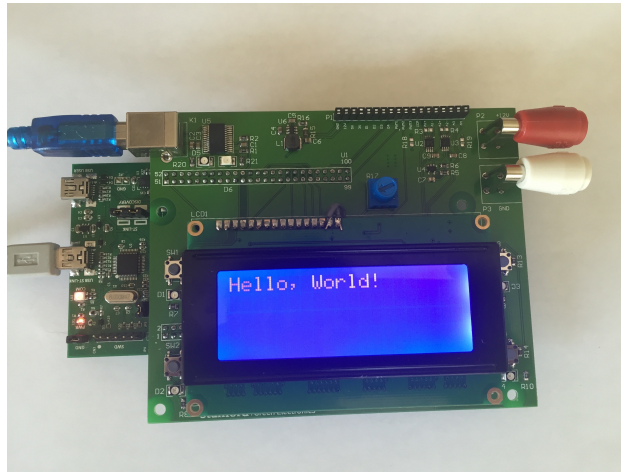


Figure 1: Connected Green Electronics Breakout Board

Program the Hardware

To program the hardware go to the project directory (`labs/lab1/hello_world`) and make sure the development board is plugged in and connected to the VM. To do this in VirtualBox go to “Devices→USB→STLink” For VMWare, go to “VM→Removable Devices” and make sure STLink is attached to the VM.

Then in a command prompt type `make` to compile the code and `make program` to upload it to the board.

The Hello World program demonstrates all the core functionality of the Green Electronics libraries. Take time to examine the source code (in the `src/` folder). Cycle through the different demonstrations using push-button 1 on the breakout board. The first mode demonstrates the use of timers to flash the Discovery board's LEDs. Press PBTN2 to change the speed. The next mode is the PWM demo which sweeps the brightness of half the LEDs (ones connected to PWM pins). The third mode is a demo of the STM32 ADCs. The conversions of four ADCs are printed over the serial port. The ADC pins are located on the breadboard header (P1). Connect the pins to a voltage between 0 and 3V and observe the readings change. The final demo is a USART echo program which will print whatever it receives back over the serial port.

A Note on the ADCs

There are four ADC pins broken out on the Green Electronics breakout board. They are labeled on the breadboard header (P1). Note that two channels are differential and pass through an instrumentation amp. If you wish to use the amplifier as a simple single-ended channel, ground the negative input. **These channels also have a built-in gain of 5.** All signals have an input voltage range of 0-3V.

Connecting the Serial Port

The Green Electronics breakout board has an on-board USB-Serial converter which can be used to communicate with the board. To communicate with the serial port, make sure the device is connected to the virtual machine (same process as described above) and launch a serial terminal. On the virtual machine, we use `picocom` to communicate with the board. To simplify connecting to the board, the green-electronics repository includes a script that handles all the settings for the serial port like adding the necessary line endings and setting the baud rate (115200). You will have to specify the serial port to connect to (`/dev/ttyUSBx`), where “x” is some integer index. List the available ports using `ls /dev/ttyUSB*`.

To launch the serial port run the following from the “green-electronics” directory:

```
scripts/board <device-name>
```

This will start `picocom`. To exit the serial terminal type `CTRL-A`, `CTRL-X`.

Using the Debugger

The STM32 Discovery allows you to use GDB remotely as code executes on the board. This is invaluable when debugging your code and worth spending some time to become familiar with. If you are not familiar with GDB check out some tutorials [here](#).

To launch the debugger run `make debug`. This will connect to the board and start with the program execution halted. Work through the following examples to explore what can be done to debug programs running on the board.

1. Place a breakpoint at `main()` — `b main`
2. Continue execution of the program — `c`
3. Break program execution — `CTRL-C`
4. Place a breakpoint at `toggle_led()`
5. See what called `toggle_led` by stepping up and down the stack frame — `up` or `down`
6. Remove the breakpoint at `toggle_led` — `clear toggle_led`
7. Set a new breakpoint at `my_adc_callback()` and continue execution.
8. Print out the new conversion results — `p val`
9. Print out the contents of `data_buf1`, the ADC1 buffer that is automatically populated after an ADC conversion.
10. Occasionally you need to directly examine the contents of memory at a specific location. Examine the contents of `buf` — `x/4uh buf` (print the first 4 memory elements as unsigned integers, half-word (16 bit) width)
11. Feel free to continue stepping through the program using the `next` or `step` commands.
12. Exit the debug session using `quit`.

Part 2

Stopwatch

Write a new program that works as a stopwatch:

1. Elapsed time is shown on the LCD in the format minutes:seconds.milliseconds (initially 00:00.000)
2. One button starts the stopwatch
3. One button stops the stopwatch
4. One button resets the stopwatch

To make a new project copy the `project_template` folder located in the `green-electronics` directory into your Lab 1 folder and rename it (`cp -r /home/green-electronics/green-electronics/project-template/ /home/green-electronics/green-electronics/labs/lab1/stopwatch`). Then you can place your code in the `src/` and `inc/` folders. You may have to update the path to the EE155 libraries in the Makefile. Make sure that the Project Template Makefile points to the `green-electronics` directory in its `EE155_HOME` variable. If not, change it to the proper path.

Implementation Hints: Take a look at the `ge_timer.h` to see what functions are available to you when creating a timer for your stopwatch and how it is used in the Hello World demo. If you're curious, take a look at `ge_timer.c` to see exactly what the functions do. The library header and source files are located in `~/green-electronics/libraries/` in the `inc/` and `src/` folders respectively.

Use `timer_register` and `timer_start` to register a function to be called every millisecond. Update the current time in this function and display it on the LCD.

For buttons, take a look at `ge_gpio.c`. Look at the Hello World program for an example on reading in a digital input.

Poll the buttons periodically to check if each button has been pressed. See the Hello World demo for examples of how to use the user interface and timer parts of the library.

Pin definitions for hardware on the board can be found in the `ge_pins.h` file.

When formatting strings for printing it is often not necessary to reinvent the wheel. Take a look at `snprintf()` and format strings to store a generated string in a character buffer and then send that to the LCD.

Part 3

Pulse-Width Modulation

Write a new program that produces a PWM signal on one of the timer output compare pins which is accessible on the header at the top of the controller board:

1. One button increases the duty cycle
2. One button decreases the duty cycle
3. Display the current PWM level on the LCD

Choose an output pin on the header. Connect an LED from the output pin to ground via a 330 resistor to limit the current to less than 10mA. Choose a PWM frequency which will cause the LED to appear to dim without flickering. Use one of the timer PWM modes rather than turning the output on and off in software.

Observe the PWM signal with an oscilloscope. Demonstrate that you can make the output constantly low, constantly high, and a wide range of duty cycles in between. Measure the PWM frequency and verify that it is the same as what you calculated for your chosen timer configuration.

Implementation Hints: Look at the included PWM library for helper functions for setting up the pin. Look at `ge_pins.h` and `ge_pwm.h` to see what pins are available for PWM control. Note that the library only implements a small subset of the PWM capabilities of the STM32F303. For more complicated operations take a look at the user manual to see what is possible.

Part 4

Extras

These are not required for this lab, but will give you a head start on functionality that we'll use in the future.

Voltmeter

Use the analog-to-digital converter to measure a voltage. Be careful about the range of safe input voltages on the microcontrollers pins: 0-3V only! We'll look at ADCs in more detail in lab 2. Take a look at the Hello World demo to see how the ADCs are accessed.

Signoffs

Stopwatch

1. Start, stop, and reset buttons work properly. Resetting while running has reasonable and predicted behavior.

Pulse-Width Modulation

1. Adjust duty cycle with buttons
2. Show PWM signal on the oscilloscope. The duty cycle matches the displayed value.

3. The PWM frequency is as predicted.
4. Demonstrate 0-100% duty cycle.