Lab 1 - Blinky

By now you should have a high level familiarity with the Crazyflie. You should be able to fly the quadrotor with the game controller and understand the basics of quadrotor flight. In this lab we will zoom in to the low level side of the project from where we build our own firmware from the ground up.

Your job is to write software from scratch to flash the green LED on the Crazyflie at 1 Hz. The green LED should be alternatively on and off for 0.5 second.

- You are forbidden to use any non-open source or paid software other than your operating system (in case of Windows and Mac OS) and virtual machine (in case of VMWare Fusion). All software we provide are open source and free.
- You are forbidden to use any of the Crazyflie source code.

Follow the following instructions to obtain resources and information needed.

Learning about the Crazyflie MCU

Please read the section titled "Learning about the Crazyflie MCU" in the resources page here, obtain the datasheet and reference manual of the MCU, and the standard peripheral library provided by the vendor. You will need them for this lab and most future ones. You can use this official standard peripheral library.

Locate the LED

You can find its whereabouts here.

You now have all the info you need to write your blinkey software.

Prepare the Programming Hardware

In this lab and future ones, you will use a JTAG programmer to write, flash, and debug software for the Crazyflie. Please read the section titled "Preparing the Programming Hardware" in in the resources page here. Again, please be very careful since incorrect connection can fry the Crazyflie.

Developing Your Software

In the first homework, you already set up the cross-development environment with all the necessary tools installed. There are multiple options to write, flash, and debug your project. This can be done at the command line or in and IDE such as Eclipse. For this lab we will outline how to develop from the command-line. This will provide insight as to what goes on behind the scenes of an IDE. You are free to write your code however you prefer. If you have never edited code under Linux, read the "Linux and its Ubuntu Distribution->Editing Code" section in the resources page.

Compile Your Code

You are allowed to use this working Makefile, which assumes that you have in the same folder the following:

- **blinky.c**

- **Makefile**
- **lib/STM32F10x_StdPeriph_Lib_V3.5.0/**
- **stm32_flash.ld** <--- copy this from: lib/STM32F10x_StdPeriph_Lib_V3.5.0/Project/STM32F10x_StdPeriph_Template/TrueSTUDIO/STM3210B-EVAL/stm32_flash.ld
- **stm32f10x_conf.h** <--- copy this from: lib/STM32F10x_StdPeriph_Lib_V3.5.0/Project/STM32F10x_StdPeriph_Template/stm32f10x_conf.h

You should comment out all non-essential libraries in stm32f10x_conf.h to speed up compilation times. As you project grows, uncomment the necessary libraries (and update the makefile).

Look over the makefile carefully. You should understand the structure and ALL of the options. You should look up each of the CFLAGS here. Executing 'make all' should produce the compiled binary (blinky.elf). 'make clean' will remove all the compilation output. We will cover the last two options of the makefile (flash and openocd) next.

Flash Crazyflie

To download (or flash) the compiled binary to the Crazyflie MCU, we will be using Open On-Chip Debugger (OpenOCD) that was installed in your first homework. Take a look at there documentation here. In your OpenOCD installation directory take a look at interface/busblaster.cfg and target/stm32f1x.cfg. OpenOCD requires you to have some sort of hardware connecting your development computer to the crazyflie. As mentioned above you will use the BusBlasterv4 to connect to the JATG interface of the Crazyflie.

To write your elf to flash memory be executing 'make flash'. If all went well you should have the green LED blinking every second!

Testing/Debug

Things usually never work on the first try. To debug your blinky project we will take advantage of GNU GDB. Official documentation is here and cheatsheets are always handy.

To help figure out what went wrong we will use GDB. This is a powerful debugger that will allow you to set breakpoints, step through your code, print variables, look at the stack pointer and pretty much every memory address, and so on. To get started run 'make openocd' to start openocd (it should find the cortex_m3) . In a separate terminal execute 'arm-none-eabi-gdb -ex "target remote localhost:3333" blinky.elf' to start the debugger. You can start poking around and find what is wrong. Find a good cheat sheet and learn the commands. Here are a few useful examples

- **monitor reset** --- monitor commands are passed to openocd
- **monitor halt**
- **file blinky.elf**
- **load blinky.elf** --- this will load blinky into SRAM
- **info registers** --- lists SP PC and other useful registers
- **x 0x40002C00** --- check to see if the watchdog is enabled (WWDG is at this memory location. It is highly recommended to print PG 34 from the datasheet
- **br blinky.c:10** --- set a breakpoint on line 10
- **continue**
- **step**

Test your Understanding: Lab Report

The Lab Report should be submitted electronically (in letter size PDF, 11pt for main text) via email with all team members copied. The email should be titled exactly "[ELEC424] Lab 01 Blinky". The email should attach a zip file that include all the sources and Makefile needed to produce the binary. The email should include a link to a YouTube video that shows your Crazyflie blink for about 30 seconds.

The report should contain the following information.

Credit Assigment

Explain the role of each team member; break down the 100% credit to team members.

Answers to Technical Questions

Please answer the following questions concisely.

1. Can you get the red LED to flash? Why is this harder? This is a good example of some of the challenges you will face with embedded devices and why the reference manual/STM-examples are key.
2. Can you explain how your main function starts execution?
3. What are the bootmode selection pins?
4. How are they wired on the Crazyflie?
5. What role does the linker script (stm32_flash.ld) play?
6. What does the ENTRY command in the linker script do?
7. What role does the startup file (startup_stm32f10x_md.s) play?
8. What happens when you push the reset button?

Up Next...

Now you have all the tools to develop MCU firmware. Although we have only flashed an LED, spinning a motor or reading a sensor is just a few lines of code away when you leverage the STM peripheral library! We recommend that you start exploring the rest of the peripherals and the Crazyflie hardware.