Lab 2 - Manual Scheduling

Buy now you have a solid development environment set up and have finished Lab 1 Blinky. In this lab we will advance our understanding in 3 areas. First, we will no longer be using the SystemInit() function provided by system_stm32f10x.c. You will now write your own "init" function to be used for all future labs. Second, with the aid of the reference manual (RM), you will learn how to drive the Crazyflie motors. Lastly, we will challenge your ability to manually schedule the execution of multiple tasks according to their associated priorities. By doing so you will begin to realize some of the benefits of using a real-time operating system.

- 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.

Part 1 System Init

As mentioned above now on you will not be allowed to reference system_stm32f10x.c in your project. Since our linker is hard coded to call SystemInit() we need to comment out line 94 from startup_stm32f10x_md.s. Your program will need to properly configure the clock tree in your own init function. You will need to read and understand chapter 7 in the RM. We suggest printing out "Figure 8 Clock Tree" and highlighting the DEFAULT path (i.e state just after powerup) for SYSCLK and mark the default frequency for each branch/CLK. In a separate color highlight your target path for SYSCLK and calculate/mark the target frequencies for each branch/CLK. This will account for the majority of the registers you will need to configure in your init function. Feel free to use STM Library and examples.

You should configure the Crazyflie's SYSCLK to run at 72 MHz by using the HSE source.

What to report

In you lab report briefly explain the clock path from start to finish (crystal to SYSCLK), including explanations for all your major register configurations.

Part 2 Motors

Before you begin motor development, take a moment to consider how to test your motors safely and not damage your Crazyflie (and yourself). We suggest fixing your Crazyflie to the Crazyflie box with a rubber band or string. You can also remove the propellers and replace with a small strip of electrical tape. Eye protection is recommended as a precaution.

You will need to use the same resources as in Lab 1 to find out how the motors are connected and how to configure the PWMs for each motor. Test your motors with a slow speed.

Once you have verified the motors work properly please send a quick YouTube video of each of the quadrotor's motors safely spinning for 1 second one at a time.

Code with Style

We recommend designing functions that will abstract away the low level details and provide a simple interface to drive each motor (Recall System Design Principle I: Modularization). You will likely want to re-use this code in your final project where things will get significantly more complicated. Take care to write clean code with concise comments.

What to report

In your lab report describe how you determined your PWM frequency. Also describe how you would calculate CCR1 values for motors off, %50, and %100.

Part 3 Manual Scheduling

Now you will need to develop software for the Crazyflie that execute a set of tasks We have provided a static library that contains a handful of simulated functions. You can download the static library here. See lab2.h for details about the simulated functions, their priorities (the smaller the higher), and execution frequency requirements. You need to update your makefile to include the static library. You can do this by adding liblab2.a AFTER your main.c. You will also need to include lab2.h in the c file where the simulated functions are used.

These functions are to simulate tasks that are time sensitive. Each function has to be executed within a given interval or time sensitive data will be lost. Each function has a given priority that corresponds with how critical it is (the smallert the Priority, the more critical). For example checkEmergency() is more important that logDebugInfo(). Your goal is to execute each function at least one time during its designated interval. You may assume that all simulated functions, except for calculateOrientation(), execute without significant delay. calculateOrientation() simulates heavy floating point math and will take some time to complete. To test your scheduling we have provided a function, updatePid(), that will check if you have met the time constraints. If you are successful, this function will return a struct with speeds for each motor for you to use with your code in Part 2. You need to set the motors to these speeds immediately after updatePid() returns.

For safety reasons updatePid() will only return speeds in the range of 0-1 (off-on) and you will need to scale it to a safe speed. This is necessary since a "slow speed" on one PWM configuration may be a "high speed" on an different PWM configuration.

If you have correctly scheduled the tasks below your motors will spin in a recognizable pattern. If you have not met the specified scheduling requirements your motors will not spin.

To make things even more challenging, you will also need to blink the green LED at 0.5Hz and red LED at 0.25Hz.

What to report

In your lab report describe how a RTOS would make scheduling easier. What complications could arise if the high priority tasks had heavy computational requirements?

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 02 Scheduling". 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 running the software from Part 3: motors spin in a defined pattern periodically and LEDs blink at the required frequency.

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 questions under "What to report" in all three parts of the Lab.

Comments for Source Code

The submitted source code must be commented to demonstrate your understanding of how it works. (i) No magic number. All numbers used for programming registers must be descriptive, human readable #define's. #define's must have comments. (ii) Provide high-level comments that explain how your code works.