Lab-03: Writing a Simple Scheduler

# Introduction

In this lab you will create a scheduler on your Nucleo-476RG that will execute three tasks at predetermined rates. To do this you will create a scheduler. A scheduler is a set of code that keeps a queue of possible processes/threads/tasks (we’ll say “**task**” here) and decides which of these should be executed at what time on the processor.

In order to keep this simple you will not be implementing a pre-emptive scheduler. A pre-emptive scheduler is a scheduler that can interrupt a task in the middle of execution, save the context (Program Counter/Stack Pointer/Register States), run the scheduler to decide with task should have time on the processor, and then restore that task, then continuing running it.

In this non-pre-emptive scheduler, each task runs and will continue until it is done completed, then the scheduler will determine which task should run next. This scheduler allows the processes to be started at different intervals and seemingly run in parallel. In the scaffolding code provided, there are three tasks written to be run at setup (times can be easily changed:

1. **D1 task**  every 700 mSec
2. **D4 task** every 1500 mSec
3. **Serial Port (UART) Write** every 2000 mSec

**Idle task** runs in-between

A diagram of a task

Description automatically generatedWhen the scheduler runs it will look for any task that should be run (according to its defined start time).

The scheduler will choose the first one in the queue. When that task is run, the scheduler checks to see if it’s time for the next one in the queue, and will run that, … , etc. until none are left ready to run. The scheduler will then go into an idle task.

Note that you’re not writing a true scheduler with the ability to ‘suspend’ a given task and then swap to another. This scheduler simply starts the tasks on a pre-defined schedule. Each of the tasks must start and then return to the scheduler. We’re not saving task state (“task control block”) information or doing anything like a context switch. For this reason, the tasks have been written to be a simple LED toggle: Toggle the LED and return and wait for the scheduler to come back and toggle again.

# Implementation

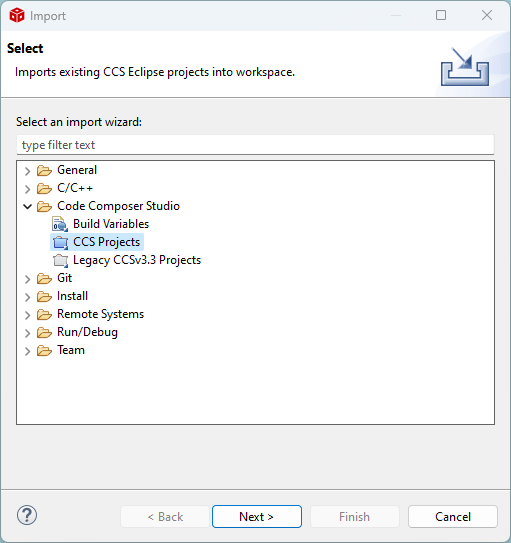
Like all schedulers, a ‘slice’ of time to run a task and look for the next task has to be provided. This is done here, like most, with a Timer Interrupt. The Timer is built into the MSP432, and referenced by the SDK as the SysClk-Timer. This is set up for you in the code provided. The scheduler here will simply use a global variable, “millis” which is updated with the absolute number of milli-seconds since the processor was turned on. The SysClk interrupt service routine updates it every “slice” (about every millisecond).

Most of the scaffolding code for this lab is provided in the CodeComposer Studio Project, that can be downloaded (HERE), and imported via the ZIP file, as done in the previous lab.

# Setup

## Insure your CodeComposer Studio runs

(Current version is 12.1.0) – This should’ve been running from Lab-02.



Import the pre-built project examples for this lab into your clean workspace.

1. From CCS, File/Import/CCS Projects
2. Import from a ZIP file ([Lab-03-Simple-Scheduler.zip](https://drive.google.com/file/d/1FJI8qDuLB8TPOf2SBRILZPq3rERjYYYS/view?usp=share_link)) (note – external link)
3. Import the Project

Your Project Explorer tab of CCS should have the new project: Graphical user interface, text, application

Description automatically generated

Build this project. There should be NO errors.

(Note – there’s nothing to DEBUG yet, since your code isn’t completed.

If it doesn’t BUILD, get help from the T/A or instructor.

# Lab Activities

Complete the following steps.

1. Text, letter

   Description automatically generatedWrite the remainder of the Scheduler\_Dispatch() code (about line 150 in scheduler.c).

This will be completed by creating a loop to go through the task queue and seeing if it’s time to run the task being inspected. Remember to take into consideration the ‘suspended’ field in deciding if it should be run or not.

Operation will be verified as you see the RED and RGB\_BLUE lights blinking at their appropriate rates.

1. Identify the terminal on your PC/Mac Connection is probably:
   1. Text

      Description automatically generatedPC: Device\_Manager, COMx:
   2. Mac: ls /dev/tty.usb\*:

*(something like)*

/dev/tty.uysbmodemM432100054

1. Connect to the serial port,with baud=115,200. no parity, no flow-control
   1. PC: program like PuTTY
   2. Mac: terminal window with

screen /dev/tty.uysbmodemM432100054 115200

Text

Description automatically generated

Output streaming milli value 🡺

If you have difficulty seeing the serial communication coming thru a terminal emulator, contact a T/A or Instructor.

1. Verify that the **SW-1** toggles the suspend of the red\_task process. Pressing the SW-1 should start/stop the RED LED from blinking. Note that it stops at the same state (on or off).
2. Add an interrupt **SW-2** interrupt to do the same with the blue\_task. This is a second ISR, that does the same thing for task[1].
3. Submit a short video of the board working.
4. Answer the questions in the “[Lab-03](https://byui.instructure.com/courses/212324/quizzes/3766207)” in iLearn.