ECEN 5823
Assignment 3 - Si7021 and Load Power Management
Fall 2020

Objective: Adding the Si7021 temp/humidity via the I2C bus and enabling / disabling the Si7021 to implement load power management.

Note: This assignment will begin with the completed Assignment #2 - Managing Energy Mode Assignment

Instructions:

1. Start by creating your assignment repository using the assignment link at https://classroom.github.com/a/12Z5XKre. You will not need to clone this from the GitHub repo you just created into a new local directory since you will be starting with your code from the previous assignment. Instead, follow these instructions and run these commands with git bash within your assignment directory. **It is recommended that you close Simplicity Studio before performing the git merge operations listed in the instructions above.** This will help avoid problems as the project file will be updated during the merge.
    a. On your local PC, duplicate the folder created in Assignment #2 to a new folder, name it something like ecen5823-assignment3-<username> where <username> is your GitHub username.
    b. Change your current directory to this new folder ecen5823-assignment3-<username> and execute the following git commands.
    c. git remote remove origin
    d. git remote add origin <url>
        i. Where <url> is the URL for the repository created with the link above
        ii. This adds the new submission repository created in the link above as your new origin.
    e. git remote add assignments-base https://github.com/CU-ECEN-5823/ecen5823-f20-assignments.git
        i. ecen5823-f20-assignments is a repo that contains branches with new files for each assignment
    f. git fetch assignments-base
    g. git merge assignments-base/assignment3
        i. This merges in new files for the current assignment into your local directory on your PC
    h. git push origin master

    i. Import ecen5823-assignment3-<username> into Simplicity Studio
        i. Remember to perform a **Clean…** immediately after you import the project into Simplicity Studio.
2. Configure logging as described below in the Appendix below and verify logging works with the logging build configuration. You do not need to attempt to implement the loggerGetTimestamp function yet. We will implement this in the next assignment.
3. Write a simple scheduler using the instructions provided in lecture as a reference. There are multiple ways to implement a scheduler. However you should ensure the only operations performed in your interrupt service routine are to:
    a. Service the interrupt, and
    b. Set an event in the scheduler
4. The timer module created in the previous assignment should be modified as follows:
    a. The interrupt period should be set to 3 seconds.
    b. LED access and functionality should be removed from the interrupt.
    c. The interrupt should set an event in the scheduler which will ultimately performs a Si7021 temperature measurement from the main loop using the code described in the next step.
    d. You should add a new function **timerWaitUs**(uint32_t us_wait) which blocks (polls) at least us_wait microseconds, using LETIMER0 tick counts as a reference, and which supports waits as long as those needed by the Load Power Management and I2C steps needed below. **Add code to range check the input parameter to ensure that the requested delay is not longer than the routine is capable of providing**.
        i. You may find it helpful to use the energy profiler and GPIO LED on/off functions to test the **timerWaitUs()** function.
5. Write routines which access the Si7021 I2C temp sensor built into the the base PCB using the instructions provided in lecture as a reference. The routines should include the following sequence:
    a. Turn on power to the Si7021
    b. Take temperature reading from Si7021
    c. Turn off power to the Si7021
    d. Log the temperature measurement to the serial port in degrees C.
    - **Any I2CSPM functions used should check return status and any unexpected return values should be logged.**
    - **Inline waits based on timerWaitUs() may be used to block as needed. We will replace this polling mechanism in the next assignment with a better low power solution.**
6. Anytime a temperature measurement is not being taken (timer interrupt is not occurring), the Blue Gecko should be sleeping in the deepest sleep state possible.

Deliverables:

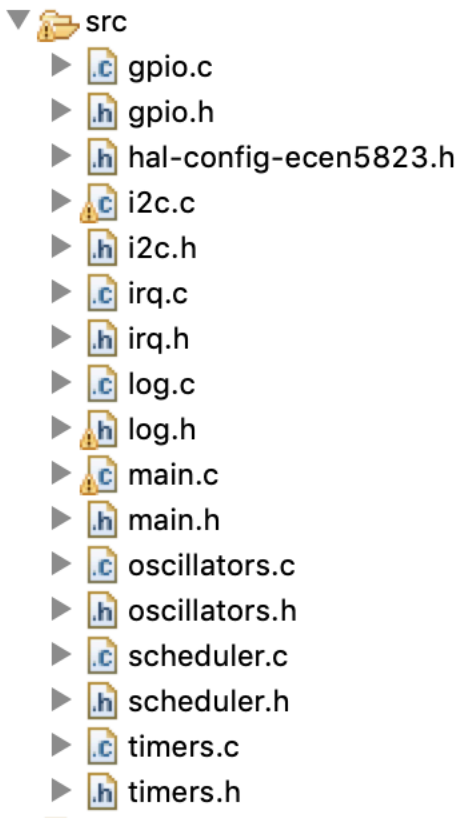
1. Answers to the Assignment3-I2CLoadPowerManagement.md file in the questions folder, included in your submission.
2. The completed program project to be run on the instructing team's computer for grading.
3. In Canvas, submit the URL to your individual github repository containing your submitted assignment code to complete your submission.

Use **git push origin master** (or equivalent GIT GUI operation) to push your results to the submission repository and **Ensure your repository linked at the GitHub Classroom link above includes the latest source and questions for your assignment before the due date.**
In Canvas, submit the URL to your individual github repository containing your submitted assignment code to complete your submission.

Approach/Guidance:

1. Create a design and UML diagram that illustrates the program flow. The UML diagram is not a graded item. However, it may greatly help you visualize what is happening, or is supposed to happen in your program.
2. Update the LETIMER0 #define for the required period.
3. Design and construct your scheduler based on guidance from in-class lectures
4. Update the main while(1) loop to receive events from your scheduler
5. Create the timerWaitUs() function in timers.c/.h
6. Enable logging and test it.
7. Create 2 new files i2c.c/.h to hold your I2C code. Develop the code for the I2C temperature measurement and load power management sequence.
8. Create 2 new files scheduler.c/.h to hold your scheduler code.

Files in your src/ directory should look like:



Appendix : Logging

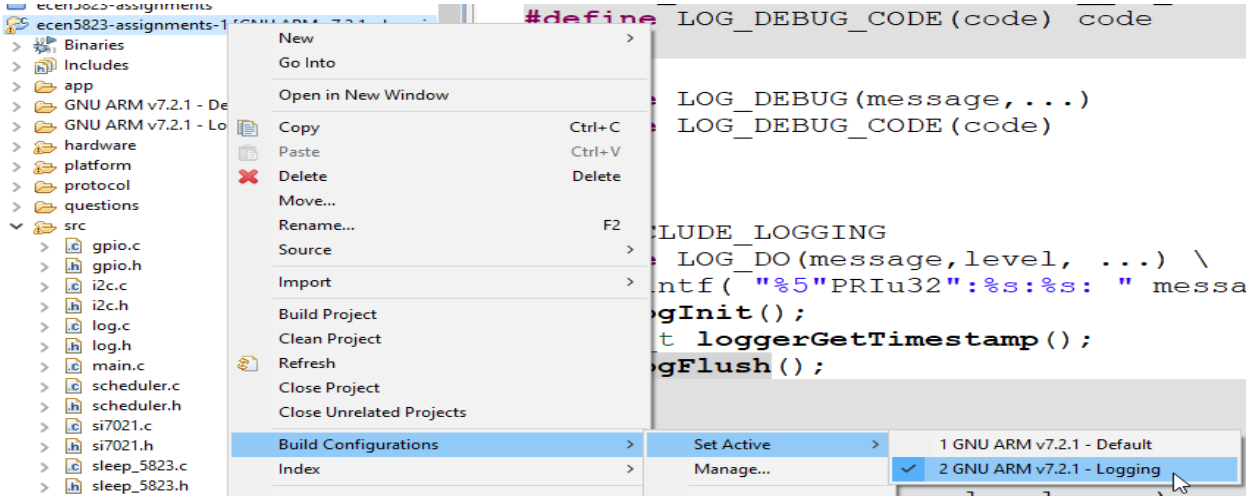
Course firmware includes a logging module (log.h/log.c) which helps you configure logging for your project. The Simplicity Studio IDE creates what SiLab's refers to as a Virtual Communication port i.e. a UART (Virtual Com Port or VCOM). This UART presents itself to the PC's system as a USB UART, which you can connect a terminal emulator to in order to view printf() output from your program. To incorporate in your project, #include log.h in your main.c file and call logInit() after initApp() and before any attempts to call LOG_INFO(), or LOG_ERROR() or similar.

Including or Removing Logging Prints

The project file is setup to easily include or remove logging prints by switching build configurations. In the “Default” build configuration logging prints are excluded from the build. In the “Logging” configuration, logging prints are included in the build and can be viewed with a Terminal Emulator.

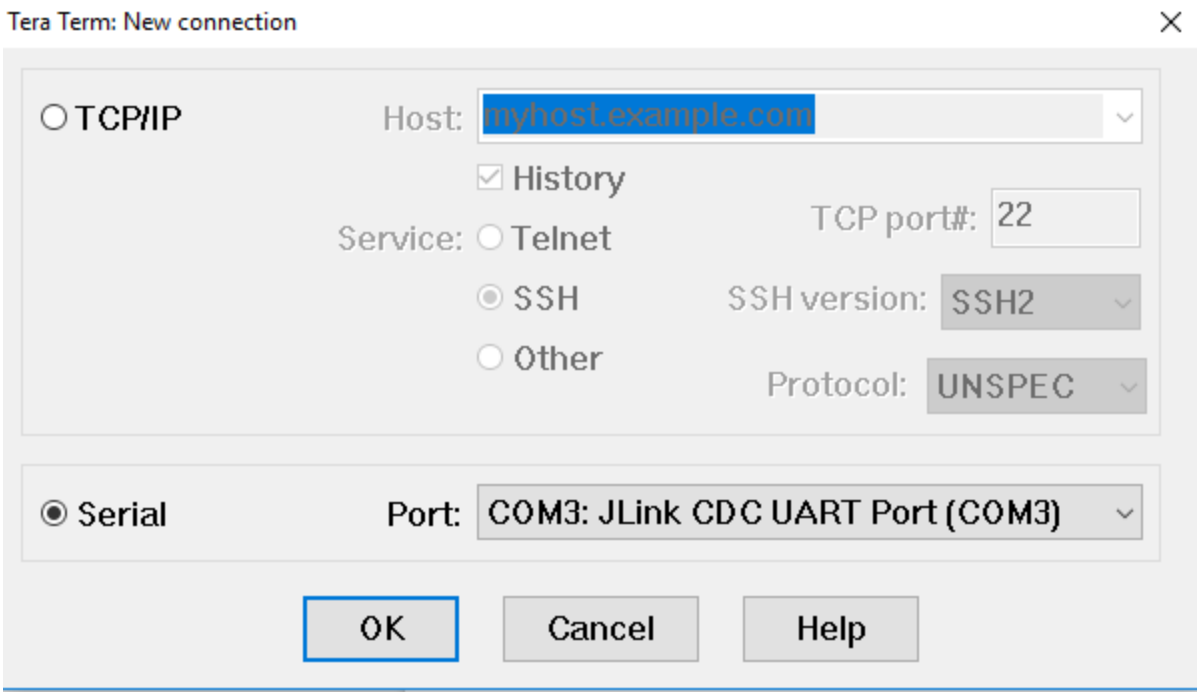
To create a new Build configuration that sets the overall (global) #define to enable logging, go to Build Configurations / Manage… and create a new build config. Go into project properties / Settings / GNU C Compiler / Symbols and add INCLUDE_LOGGING 1

To switch build configurations, right click on the project name in Simplicity Studio and select “Build Configurations-> Set Active”. Select the “Logging” or “Default” build configurations depending on whether you would like logging configured.

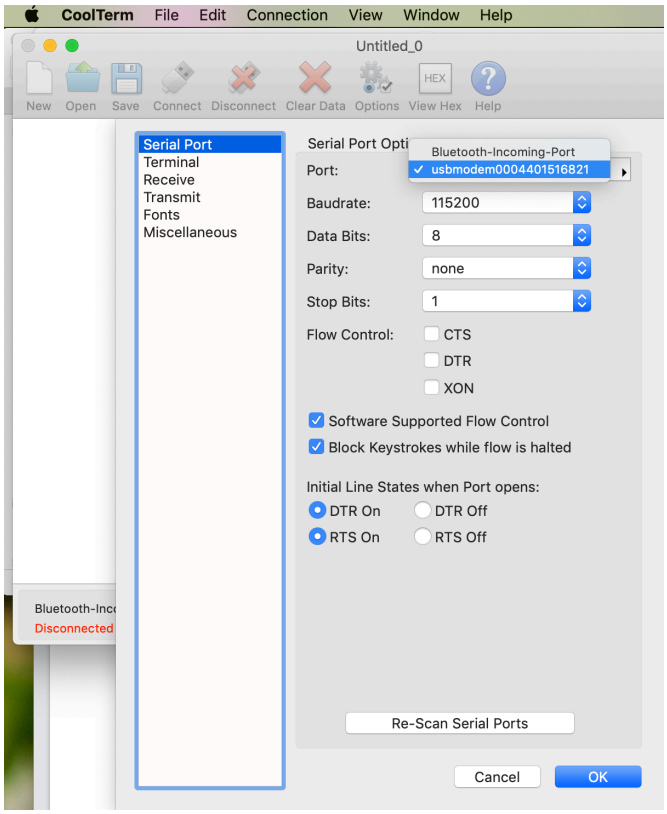


Configuring Terminal Emulator

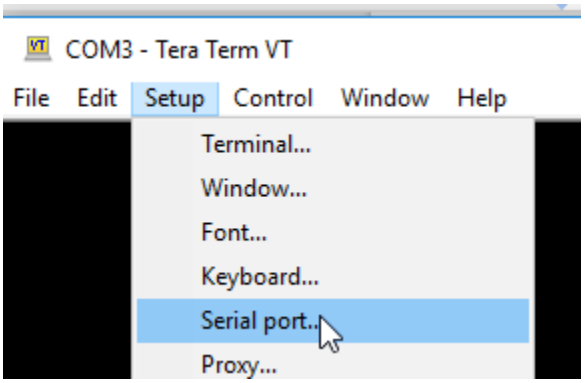
Configure Tera Term or similar terminal emulator program as follows, with the UART port labeled “Jlink CDC” as the serial port target. The COM number may be different, however you will see this port appear or disappear when the board is plugged/unplugged.

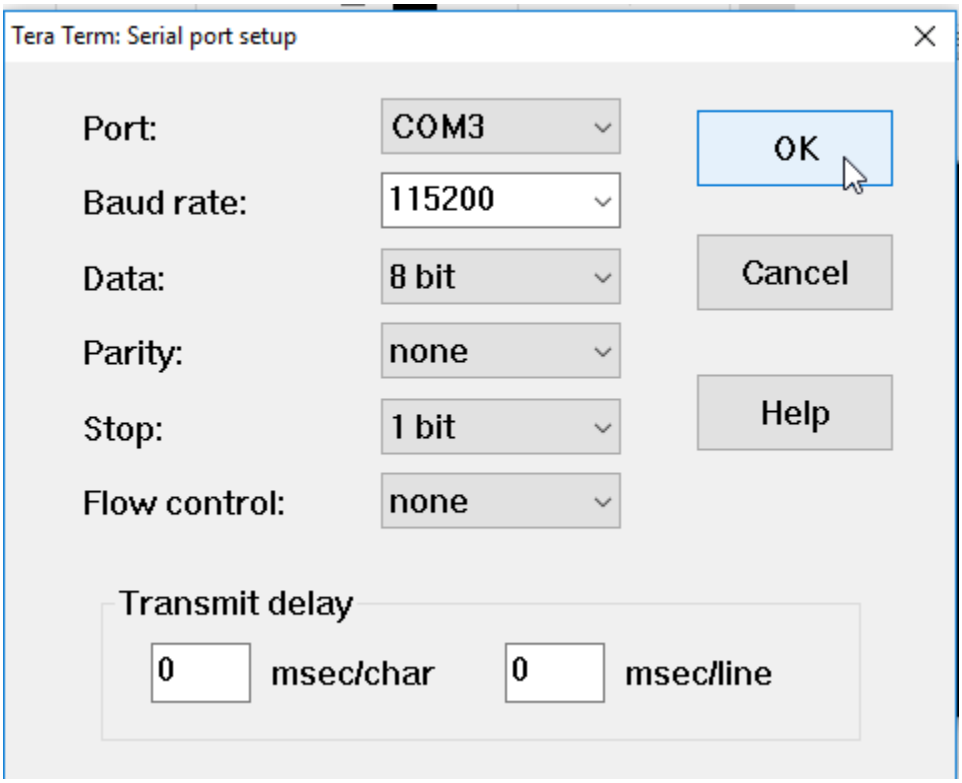


On a Mac using CoolTerm the USB UART shows up as:



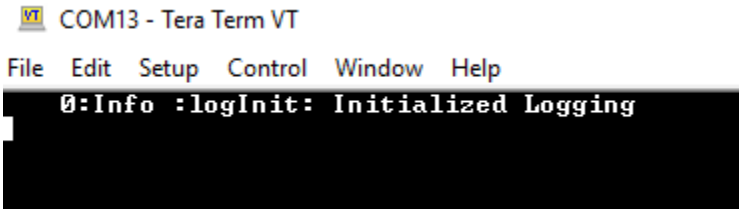
After enabling connection to the UART port, configure the appropriate COM port (as selected above) for **115200, 8-bit, no parity, 1 stop bit and no flow control**. See screenshots below for Tera Term.





Testing

When logging is initialized properly and terminal emulator is configured, you should see a message in the emulator



Optional: Flush before Sleeping

Add a call to logFlush() immediately before calling SLEEP_Sleep(). This will prevent bogus chars from being printed out on the terminal.

Optional: Incorporate Timestamp

To incorporate a timestamp into your logs, implement a function timerGetRunTimeMilliseconds() which returns the run time in milliseconds since power-up. This value will then be printed as the first number in your log message. You will also need to make sure any variables used by this function are initialized before calling logInit();