ECEN 5823 Spring 2025
Assignment 3 - Si7021 and Load Power Management Part 1

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/C9uU_NR4. 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. Ensure you are setup to use SSH keys before proceeding, see the instructions from github at https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/ and be sure to use Git Bash For Windows or Cygwin if using a Windows environment. (Cygwin .ssh directory will be different than Git Bash.)

    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 push origin master
    f. Import ecen5823-assignment3-<username> into Simplicity Studio
        i. Remember to perform a **Clean…** immediately after you import the project into Simplicity Studio.
2. 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

c. Your interrupt service routine (ISR) should not know anything about your scheduler internal variables, data structures, bit fields / #defines. Create scheduler functions, that don't take any parameters, to call from your ISR that handle all of the scheduler internal data manipulations. Example: schedulerSetEventUF() (or make up whatever function name suits your implementation) - this would create an LETIMER0 underflow event for example. Implementing it this way is a great example of isolation of functionality: the ISR handles all LETIMER0 interrupt processing, and schedulerSetEventUF() handles all scheduler processing.

d. As per lecture material, design your scheduler to be able to store multiple events at the same time!

3. The timer implementation created in the previous assignment should be modified as follows:
   a. The interrupt period should be set to 3 seconds (use the underflow interrupt, UF bit - just so we're all doing it the same way).
   b. LED access and functionality should be removed from the interrupt service routine.
   c. The ISR should set an event in the scheduler which will ultimately perform a Si7021 temperature measurement from the main loop using the code described in the next step.
   d. You shall create 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 less than or longer than the routine is capable of providing, and choose a way to handle this exception.**

      (Hint: great coders always range their functions's inputs 👍 and range checking can add up to +10 extra credit points). We will replace this polling mechanism in **timerWaitUs()** in the next assignment with a better low power solution.
      i. You may find it helpful to use the energy profiler and GPIO LED on/off functions to unit test the **timerWaitUs()** function.
4. Write blocking (polling) 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 (this is an incomplete list):
   a. Turn on power to the Si7021
   b. Take temperature reading from Si7021
   c. Turn off power to the Si7021
   d. To have this code you develop better positioned for the next assignment, I recommend creating 1 function for steps a. and c. For step b. create 2 functions, one that sends the command to the Si7021 to take a temperature measurement, and one function that reads the result. 👍

   e. Log the temperature measurement to the serial port in degrees C, as an integer value.
   - **Any I2CSPM functions used should check return status and any unexpected return values should be logged.**
6. Anytime a temperature measurement is not being taken (timer interrupt is not occurring), the Blue Gecko shall be sleeping in **EM3**.
7. For the code you submit for grading, **the only LOG_***() call should be 1 call to print the temperature measurement.** No other log messages should be displayed in the terminal window unless an API function call returns an error. If an API routine returns an error, use LOG_ERROR() to report those conditions so you know something is not functioning properly; and so you can correct that(those) condition(s). **Also, no calls to CMSIS Systick routines are allowed.**
8. When taking screen captures of your Energy Profiler window, compile the code with any additional logging calls you may have added to help you debug your program disabled.

Questions:
Answers to the Assignment3-I2CLoadPowerManagement.md file in the questions folder, included in your submission.

Deliverables:

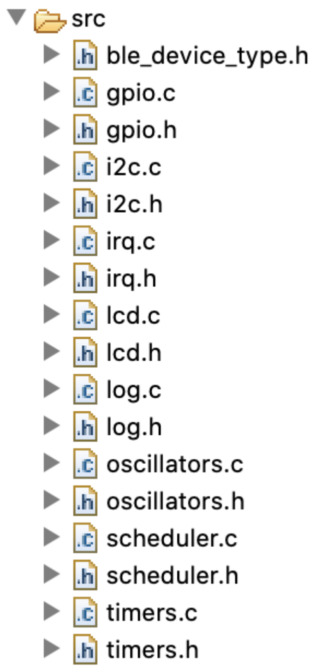
- Submission via **github classroom** and your repository setup via the classroom link at above. Use the following git command to submit your assignment:
  git push origin master
- Verify your ecen5823-assignment3-<username> repository contains your latest/ intended code and answers to the questions. It is your responsibility to push the correct version of your project/code. Use a web browser to verify the correct version of your project/code has been uploaded to GitHub.
- In **Canvas**, submit the URL to your github repository and in **GitHub** create a tag of your GitHub submission prior to the due date with the text of A3-final. The date and time of the tag creation is what we use to determine whether your submission was on time or late. These 2 steps will complete your assignment 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, but is required to get help at office hours.
2. Update the LETIMER0 #define for the required period.
3. Create 2 new files scheduler.c/.h to hold your scheduler code. Design and construct your scheduler based on guidance from in-class lectures.
4. Update the **app_process_action()** function to receive events from your scheduler.

5. Create the **timerWaitUs()** function in timers.c/.h. Develop unit test code for your timer. This is a special build of your software that shall throughly exercise your timer function with small (**10ms**), medium (**100ms**), and large values (**1 sec)**, plus values less than and greater than your routine can handle so you test your range checking code. In your unit test code you can use the LEDs and observe the current profile in the Energy Profiler to verify your **timerWaitUs()** function. Many students mess up the design and implementation of their **timerWaitUs()** function, and subsequently fail to detect those errors because they fail to perform unit testing. 90% of the I2C transfer failures that occur in A3 and A4 are due to errors in the implementation of the **timerWaitUs()** function. In industry you will be writing many unit tests. This is time well spent: Unit test your **timerWaitUs()** function! Run your unit test in all 4 energy Modes as we will be limiting the MCU to no lower than EM2 starting with A5!
6. Create 2 new files i2c.c/.h to hold all of your I2C code. Develop the code for the I2C temperature measurement and load power management sequence using the I2CSPM_***() functions.

Files in your src/ directory should look like:


```
src
  ble_device_type.h
  gpio.c
  gpio.h
  i2c.c
  i2c.h
  irq.c
  irq.h
  lcd.c
  lcd.h
  log.c
  log.h
  oscillators.c
  oscillators.h
  scheduler.c
  scheduler.h
  timers.c
  timers.h
```

Appendix : Logging

Course firmware includes a logging module (log.h/log.c) which helps you configure printf functionality for your project. The Simplicity Studio IDE creates what SiLabs refers to as a Virtual Communication Port i.e. a UART (Virtual Com Port or VCOM in SiLabs terminology). This UART presents itself to the PC's system as a USB based UART, which you can connect a terminal emulator to in order to view printf() output from your program. To incorporate in your project, do the following:

- Do include these lines at the top of each .c file that you want to call the LOG_***() functions from:
    - **#define** INCLUDE_LOG_DEBUG 1
    - **#include** "src/log.h"
- Do not #include "src/log.h' in any other .h file!!!

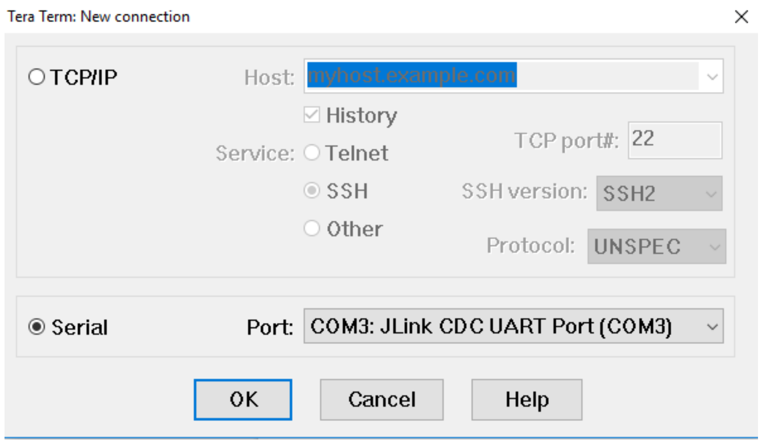
If you then want to disable all logging for a particular .c file, then just comment out the line:

- // **#define** INCLUDE_LOG_DEBUG 1

Configuring A Terminal Emulator

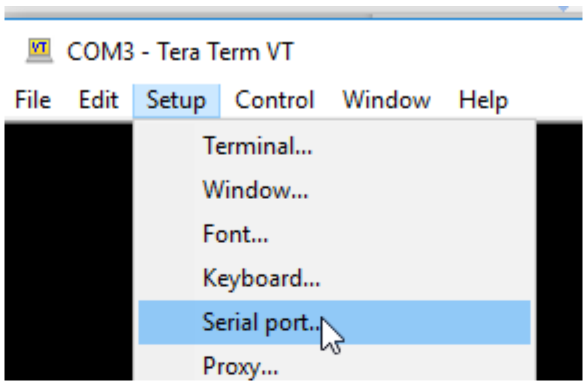
On Windows configure Tera Term or similar terminal emulator program as follows, with the UART port labeled "Jlink CDC" as the serial port target. The COM number may be different, however you will see this port appear or disappear when the board is plugged/unplugged.

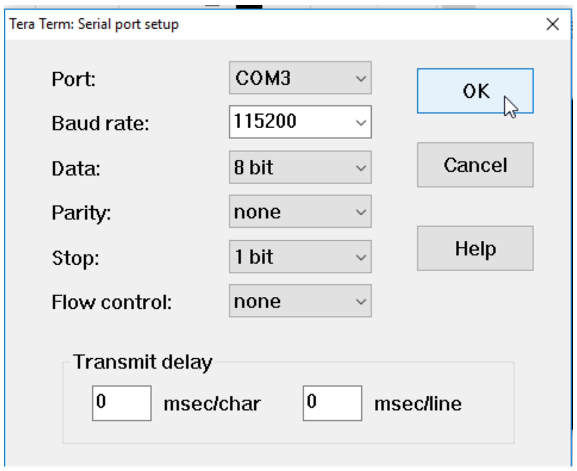
After enabling connection to the UART port, configure the appropriate COM port (as selected above) for **115200, 8-bit, no parity, 1 stop bit and no flow control**. See screenshots below for Tera Term.


Tera Term: New connection

TCP/IP Host: myhost.example.com
History
Service: Telnet / SSH / Other   TCP port#: 22   SSH version: SSH2   Protocol: UNSPEC

Serial Port: COM3: JLink CDC UART Port (COM3)




Tera Term: Serial port setup

Port: COM3

Baud rate: 115200

Data: 8 bit

Parity: none

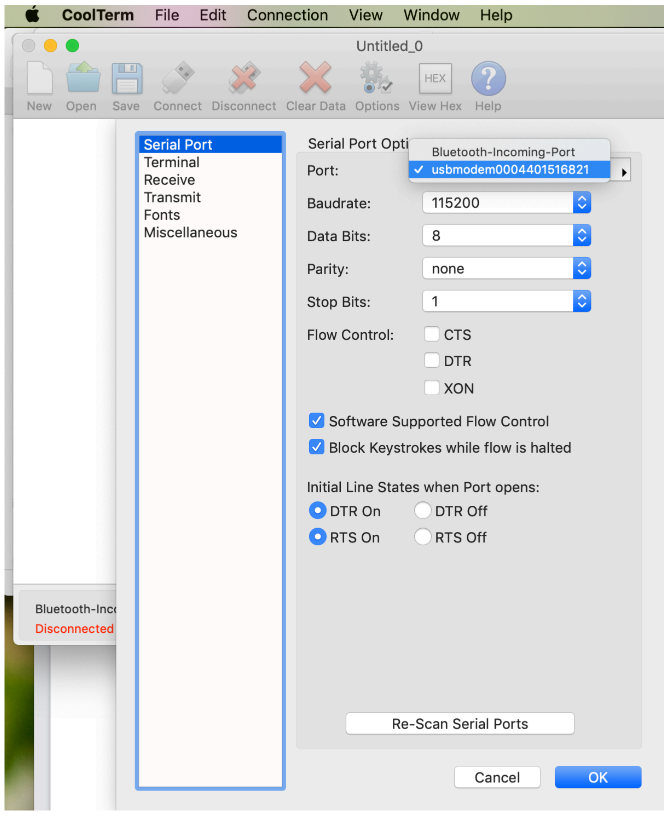
Stop: 1 bit

Flow control: none

Transmit delay

0 msec/char 0 msec/line

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


CoolTerm File Edit Connection View Window Help
Untitled_0
New Open Save Connect Disconnect Clear Data Options View Hex Help
Serial Port
Terminal
Receive
Transmit
Fonts
Miscellaneous
Bluetooth-Inc
Disconnected

Serial Port Opti

Port: Bluetooth-Incoming-Port / ✓ usbmodem0004401516821

Baudrate: 115200

Data Bits: 8

Parity: none

Stop Bits: 1

Flow Control: ☐ CTS ☐ DTR ☐ XON

☑ Software Supported Flow Control

☑ Block Keystrokes while flow is halted

Initial Line States when Port opens:

[✓ DTR On] DTR Off

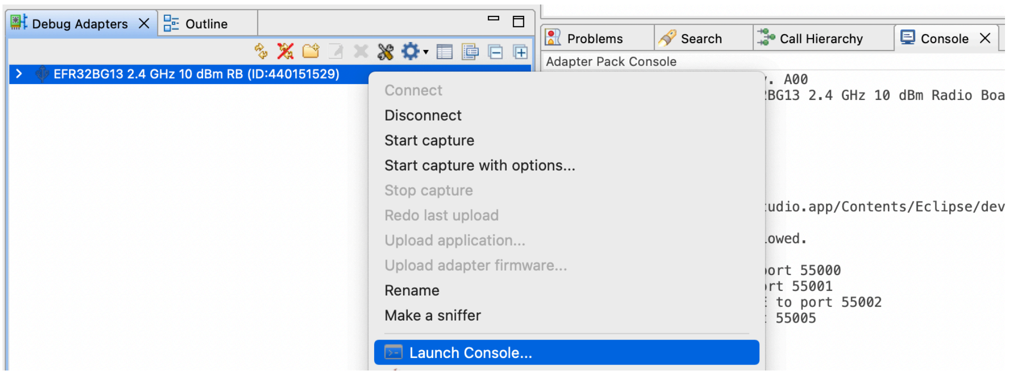
[✓ RTS On] RTS Off

On a linux system using a terminal window:

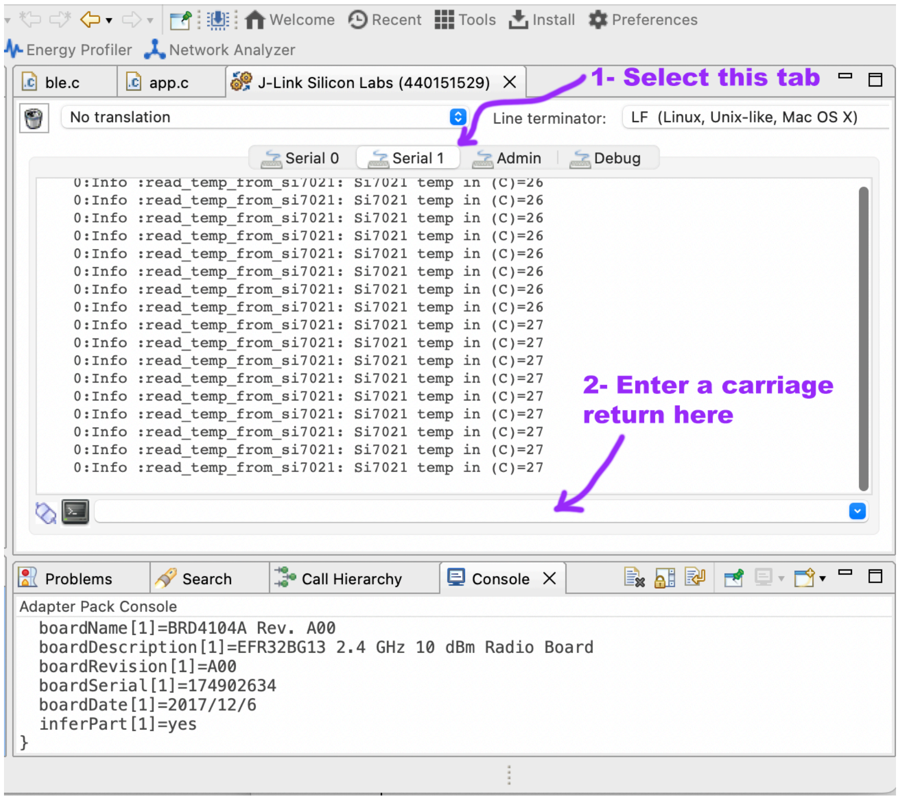
```
% ls -l /dev/*usbmodem*
crw-rw-rw-  1 root  wheel  0x9000003 Feb 10 18:08 /dev/
cu.usbmodem0004402367371
crw-rw-rw-  1 root  wheel  0x9000002 Feb 10 18:08 /dev/
tty.usbmodem0004402367371

% screen /dev/cu.usbmodem0004402367371 115200
```

Viewing VCOM output within Simplicity Studio



```
. A00
BG13 2.4 GHz 10 dBm Radio Boa
udio.app/Contents/Eclipse/dev
owed.
ort 55000
rt 55001
 to port 55002
 55005
```



```
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=26
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
0:Info :read_temp_from_si7021: Si7021 temp in (C)=27
```

Adapter Pack Console

```
   boardName[1]=BRD4104A Rev. A00
   boardDescription[1]=EFR32BG13 2.4 GHz 10 dBm Radio Board
   boardRevision[1]=A00
   boardSerial[1]=174902634
   boardDate[1]=2017/12/6
   inferPart[1]=yes
}
```