CS466 Lab 2 -- Hardware, Development Tools and Blinking the LED using FreeRTOS

Due by Midnight Saturday 2-13-2020.

Use provided lab format on Blackboard

Note: This is an individual lab, you are free to collaborate but every student must perform the lab and hand in a lab report. It will be critical that each student is able to develop to the target platform..

Overview:

This lab is an enhancement of lab 1 requirements using a multithreaded implementation.

Lab Preperation:

- ☐ Review your Lab1 problem and solution.. Does it meet all the requirements in step 9 below?
- ☐ Read the lab document CS466 Lab 2 Information.pdf, it supply's information required to get FreeRTOS up and running. Note that this a file from a previous years lab and may have slight differences from the blinkyRtos.c in the lab02 directory.
- ☐ Locate the FreeRTOS API documentation https://www.freertos.org/a00106.html and read about the xSemaphoreTake() and xSemaphoreGiveFromISR() FreeRTOS functions in detail. Take special care to understand the second parameter in xSemaphoreTake() and what it's behavior will be if this value is 0 or greater.
- ☐ Take a hard look at the blinkyRtos.c program. There are a lot of new concepts in it and it will be confusing at first.
  - Several driver library functions are used to simplify GPIO setup
  - Several #define macros are new that make the program read a little better.
  - There is an interrupt handler in place to detect the SW1 press. Study what the interrupt handler does
  - Look how the interrupt routine is registered.

Objectives:

- ☐ To discover that the RTOS helps organize and make your code more independent and flexible.
- ☐ To interact with a multithreaded program and synchronization with an asynchronous interrupt service routine.

Lab Work:

1. ☐ Perform a 'git pull' on the class repo to get the required lab02 directories.
2. ☐ Download the latest FreeRTOS Version (FreeRTOSv202012.00.zip when I updated this document) from https://sourceforge.net/projects/freertos/ unzip so that the FreeRTOS directory is a peer to your class and TivaDriver directory.
3. ☐ This Lab also starts to use the TI library code for the Tiva board when before we only accessed header files out of the directory.
4. ☐ Run the blinkRtos program and see how SW2 alters it's behavior.
5. ☐ Add code to read the status of SW1 and SW2. You will need to enable new GPIO pins for input and internal pull-up resistors. Do both switches follow the same rules (hint, hint).
   a. SW1 is done for you

   b. Study the existing ISR `_interruptHandlerPortF(void),` Look in the TivaDriver API document to see what the functions `GPIOIntStatus()` and `GPIOIntClear()` do.
   c. Add the GPIO setup for SW2: Look at how SW1 was configured using the TivaDriver code.
   d. You will still need to play the lock-register game
   e. It should be a pretty simple change to the existing program to verify that SW1 and SW2 are sensed and actionable.

6. ☐ Add two new FreeRTOS tasks to the system for each of the other two LED behaviors, start simple. Make the two nre tasks have higher priority then the heartbeat task. I define and start the tasks in the end of my main() function.

7. ☐ Normally a simple RTOS task has the overall structure shown in the file “CS466 Lab 2 Information.pdf”

8. ☐ See if you can make the semaphore_take blocking on the new normally inactive tasks so the they will happily wait forever.

9. ☐ Modify the code so that:
   a) The green LED always blinks at 1Hz
   b) By default the red and blue LED, are off.
   c) Pressing SW1 will cause the red led to flash 20 times at 15Hz
   d) Pressing SW2 will cause the blue led to flash 10 times at 13Hz
   e) This should work with any combination of SW1 and SW2
   f) Use only the button press as a trigger, the led should blink their required number of times no matter how long the button is pressed.

10. ☐ Use your self-check timing function to verify that the tasks are operating at the correct frequencies. What can you do if the frequencies are off?

11. ☐ Add two behaviors
   a. ☐ If you don’t press a button in 60 seconds red and 90 seconds for blue that the LED’s will flicker for 1 second to remind the user that they are available.
   b. Because sometimes hardware fails, check that the buttons are not stuck by asserting if either is pressed for more than 3 minutes. Assert!.

   Note: as you are developing these behaviors.. If you test at the actual times you’ll be sitting around for many many minutes waiting.. I suggest that you use a #define SPEED_TEST or some similar concept to reduce actual times during your implementation and all but final testing.. Imagine if the times were hours or weeks?

   Question: What is the purpose of the assert() macro? If an assert() fails, what is the most important thing that the program should do?

12. ☐ Questions:
   a. The program is larger, no question, Is it more understandable if you had to explain it to someone else? Why?

   b. The resource requirements of the second lab have increased. For the three major segments we discussed in class `[‘.text’, ‘.bss’, ‘.data’]` what happened?

   c. How many bytes of data did we use in the Flash ROM? How much is still available?