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

Due by Midnight Sunday 2/4/2024.

Use provided lab format on Canvas

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:

Lab Preperation:

- ☐ Review your Lab1 problem and solution.. Does it meet all the requirements in step 9 below?
- ☐ Update (git pull) your class repo, be sure to commit any changes that you have first.
- ☐ Take several minutes to look at the CMakeLists.txt file and how the Pico support libraries work into our lab2 build with FreeRTOS
- ☐ Locate the FreeRTOS API documentation at http://www.freertos.org/a00106.html and read about the xTaskCreate(), 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.
- ☐ You may want to hunt down some simple semaphore examples.
- ☐ Take a detailed look at the provided lab2.c program.
  - The main program sets up the LED hardware in the same manner.
  - The heartbeat() function will blink the LED at 1Hz at startup.
  - Look at main and notice how the example registers a new 'Task' and sets it to point to the heartbeat() method.
  - 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.
  - Notice that we now have 'printf()' calls. How was that enabled?
  - Notice theat there is commented out semaphore code and an example SW1 handler for some aid.
- ☐ Take a detailed look at the config file FreeRTSOConfig.h
- ☐ Take time to look at the CMakeLists.txt file.

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.
- ☐ Use of printf() to redirect serial type data over a USB port

Lab Work:

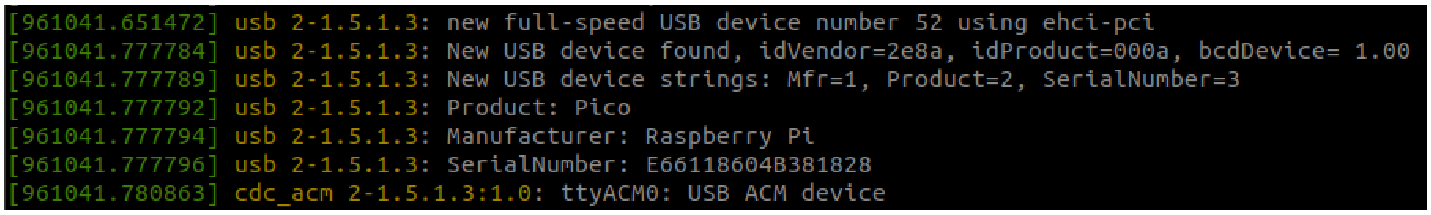
1. ☐ Perform a 'git pull' on the class repo to get the required lab02 directories.

2. ☐ Follow the instructions in the cs466_s24 README.md file to add the FreeRTOS environment to your directory Tree. Note that I used FreeRTOSv202212.01 in the build files. I have this on a USB stick in Lab as the download from https://github.com/FreeRTOS/FreeRTOS/releases/download/202212.01/FreeRTOSv202212.01.zip can take a long time.

3. ☐ Run the lab2 program and see how SW1 (Which I have assigned to pard pin 22, GPIO17) alters its behavior.

   LAB-Questions:
   a. Describe the program behavior
   b. Is it consistant, discuss your result and probable causes.

4. ☐ After the program is running it should show up as a device in /dev.
   a. On Ubuntu you can run the command `dmesg` or `sudo dmesg` after the program starts to determine if the pico has registered iusself as a serial device. On my wortkstation the interesting out (at the end of my 'dmesg' output) looks like;



```
[961041.651472] usb 2-1.5.1.3: new full-speed USB device number 52 using ehci-pci
[961041.777784] usb 2-1.5.1.3: New USB device found, idVendor=2e8a, idProduct=000a, bcdDevice= 1.00
[961041.777789] usb 2-1.5.1.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[961041.777792] usb 2-1.5.1.3: Product: Pico
[961041.777794] usb 2-1.5.1.3: Manufacturer: Raspberry Pi
[961041.777796] usb 2-1.5.1.3: SerialNumber: E66118604B381828
[961041.780863] cdc_acm 2-1.5.1.3:1.0: ttyACM0: USB ACM device
```

   b. From the commanf output I see that my workstation has set the ISB device up as /dev/ttyACM0 but by default this device will only be avilable if you are running as root.
   c. Linux has a service `udev` that allows running configuration whenever devices are connected. Using the `idVendor=2e8a, idProduct=000a` information you can construct a udev file and register it with the udev process that will make the `/dev/ttyACM0` device available to users
   d. Creatre a file /etc/udev/rules.d/95-pi-pico-udev.rules that contains the following two lines (the second line is long and wraps below.. Don't wrap it in the udev file.);

```
# raspberry pi-pico
ATTRS{idVendor}=="2e8a", ATTRS{idProduct}=="000a", MODE="0666",
ENV{ID_MM_DEVICE_IGNORE}="1", ENV{ID_MM_PORT_IGNORE}="1"
```

   e. After creatign the udev file you can reboot your linux or it's much easier to just retart the udev service using the command;
      sudo udevadm control --reload-rules
      sudo udevadm trigger
   f. Restart your pi-pico board and check that you have read-write access to /dev/ttyACM0. A ls command should return the following;

```
$ ls -l /dev/ttyACM0
crw-rw-rw- 1 root dialout 166, 0 Jan 24 17:58 /dev/ttyACM0
```

   g. If the protections are not 666 (rw-rw-rw) for the device you have an issue somewhere.

5. ☐ use your serial program of choice to listen to /dev/ttyACM0. I use kermit but any will work. The serial port should be setup as 115200-n-8-1. I have put a kermit executable and setup file in the lab directory, thses will priobably only work on Ubuntu 20.04 but the source can be found an recompiled pretty easily for other platforms.

```
$ kermit /home2/miller/kermACM0
Connecting to /dev/ttyACM0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
```

```
Type the escape character followed by C to get back,
or followed by ? to see other options.
--------------------------------------------------------
hb-tick: 125
hb-tick: 125
hb-tick: 125
…
```

6. ☐ Remove the code that changes the heartbeat frequency if SW1 pressed.
7. ☐ Add code to cause pressing either SW1 (GPIO17) or SW2 (GPIO16) to fire the interrupt routine for you.
    a. SW1 is done for you, add setup and interrupot connection for SW2
    b. Create two semaphores (variables) that are static globals at the module level.
    c. In main() before you create the threads assign the semaphore objects to the result of xSemaphoreCreateBinary();
    d. In the ISR, if SW1 is pressed give a semaphore (e.g. semSW1). If SW2 is pressed give a different semaphore (e.g. semSW2)

8. ☐ 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.
    a. Generally RTOS task implemntations should not return.
    b. The task is generally characterized by
        i. An infinate loop
        ii. A blocking or delay call
        iii. Do something meaningful

    In my SW1 receive task I satisfy all three of the above requirements

```
void sw1_handler(void * notUsed)
{
    while (true)
    {
        xSemaphoreTake( _semBtn, portMAX_DELAY);
        printf("sw1 Semaphore taken..\n");
    }
}
```

9. ☐ Using only interrupt detection of SW1 and SW2 press, modify the code so that:
    a. If no buttons are pressed, the green LED continuously blinks at 1.0Hz
    b. Pressing SW1 will cause the led to flash 20 times at 15Hz
    c. Pressing SW2 will cause the led to flash 10 times at 13Hz
    d. If both buttons are pressed the LED should flash continuously at 5 Hz.
    e. For single buttons, 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. ☐ Notice that we FreRTOS tasks use vTaskDelay in place of the non-RTOS delay_ms/delay_us. What is the best delay resolution that you can attain.

11. ☐ By now you have probably noticed that a single button press may cause mutliple interrupt occurnaces. Put code in the ISR that will add a 25ms guard time after a button press to ignore bounces that may occur when you press the button.

12. ☐ If you don’t press a button in 60 seconds, cause the LED to flicker at 20Hz for 1 second to remind the user that they are available.