Lab 02

Due Jan 24 by 11:59pm **Points** 10

Instructions

Complete the following procedures. When finished make sure your code is checked into gitlab (gitlab.cset.oit.edu). You will also need three LEDs (with current limiting resistors) and two pushbuttons to complete this lab.

Procedure

This lab will expose a few more FreeRTOS kernel functions. Last time we used the “task create” function to spawn a single task that blinked the LED every **500 ms**. Our task was capable of taking a parameter as a void *, But we just sent NULL for this hello world intro.

In this lab, you will need to wire up additional LEDs and buttons to the EXT3 header of the board. I will list what pins will be used for what. You will then create multiple tasks, delete tasks, and suspend/resume tasks. There are many ways to model the functionality for this lab, you are required to follow the lab specification specified here. You may also only use the FreeRTOS functions provided as part of the lab description.

Application Function

taskSystemControl

This task will poll the status of the three switches and act accordingly. Each of the switches will cause the behavior listed below:

EXT_SW1

The first time **EXT_SW1** is pressed, **EXT_LED1** will start flashing. The second press will start **EXT_LED2**, and finally, the third press will start **EXT_LED3** flashing. Any further pushes will do nothing. This will be accomplished by doing an xTaskCreate() for the next LED. Unlike last time, the task creation must take a parameter (the LED number) and it also must accept back the handle to the task, Last time both of these parameters were **NULL** as they were not needed. Look at the API reference for xTaskCreate for the parameter order. We need the parameter to tell the LED function which LED it will be blinking. The Handle (one for each LED task created) is necessary to either suspend or delete the task as shown in other button pushes. The LED task will then toggle the appropriate LED and then do a vTaskDelay(500 / portTICK_PERIOD_MS);.

EXT_SW2

Each time **EXT_SW2** is pressed, the highest LED flashing will stop. LEDs will stop until all of them have been stopped on subsequent presses of **EXT_SW2**. After all LEDs have stopped flashing, additional presses will do nothing. This will be accomplished by deleting the highest flashing LED task. To delete a task, the task handle created during the task creation is used along with the API function vTaskDelete. After deleting a task **EXT_SW1** can restart it, so if all three LEDs were blinking, then EXT_SW1 did nothing. If **EXT_SW2** is pressed then **EXT_LED3** would stop blinking, **EXT_SW1** would once again start this task blinking.

SW0 (Onboard Switch)

A press of **SW0** will “freeze” al current flashing LEDs. If fewer than three LEDs are currently flashing, **EXT_SW1** will have the opportunity to cause any higher LEDs to start flashing as described above. A second press of **SW0** will “unfreeze” the frozen LEDs. This is accomplished by suspending all currently running LED tasks on the “first” button press. The second button press will then resume all created LED tasks. We will use to API functions to accomplish this task; vTaskSuspend and vTaskResume.

The polling of the switches will happen in sequence and actions taken accordingly. After all three switches have been polled, the task will perform a vTaskDelay for **100ms**.

taskHeartbeat

The heartbeat task will be responsible for toggling the onboard LED every second. This will give you a visual clue that the FreeRTOS system is still running.

Hardware Considerations

The input will come from the onboard switch (don’t have to do anything for this one) and two external switches wired to the EXT3 header. Outputs will be the onboard LED (again already wired in) and three LEDs wired into the EXT3 header. The pins on the header and corresponding pins on the processor are given in the following table. Make sure and wire current limiting resistors for the LEDs.

Pin on EXT3 Header	Pin on the Processor	Function
5	PD28	EXT_LED1
6	PD17	EXT_LED2
9	PE1	EXT_LED3
10	PD26	EXT_SW1

15	PD30	EXT_SW2
19	-	GND
20	-	VCC

Initialization for the LEDs is pretty straight forward. You just need to call the ASF function ioport_set_pin_dir(PIN_NUM, DIRECTION);. This will set the GPIO pin as an input or output. There are defines for this already: IOPORT_DIR_OUTPUT and IOPORT_DIR_INPUT. The buttons are a little more complicated. You have to set the pin direction as with the LEDs, but you also need to setup the mode of the pin and its sense. This will accomplish turning on internal pull-ups and some other stuff. The two functions needed for this are: ioport_set_pin_mode(PIN_NUM, MODE) and ioport_set_pin_sense_mode(PIN_NUM, SENSE).

The MODE will be defined as follows: (IOPORT_MODE_PULLUP | IOPORT_MODE_DEBOUNCE). This will turn on pull-up resistors and debounce for the switches. The SENSE will be defined as follows: (IOPORT_SENSE_RISING). This sets the switch to detect the Rising edge.

You will want to create some defines for all these external devices. This should be in a file called myDefines.h.

Final Considerations

The FreeRTOS API calls that can be used with this lab are:

- xTaskCreate()
- vTaskDelete()
- vTaskSuspend()
- vTaskResume()
- vTaskDelay()

The API reference can be found at **FreeRTOS API Reference** **(https://www.freertos.org/a00106.html)**.

You should end up with a total of 5 tasks by the time it is done. You will have the main control task, three LED tasks controlled by the buttons (One for each external LED), and the heartbeat task. Even though there are five tasks, there should only be 2 functions for the tasks as all the LEDs and heartbeat will call the same function with the appropriate LED number passed in.

You will also need to right an LED driver (ledDriver.c and ledDriver.h) to control the various LEDs. Your driver can call the ASF functions though.

The LED driver should have the following functions:

- void initializeLEDDriver(void)
  - Responsible for initializing the LEDs, this should be called from the Hardware Init Function in main.
  - It will setup the ports for the LEDs and set them to the OFF state.
- uint8_t readLED(uint8_t uiLedNum)
  - returns the current state of the given LED
  - uiLedNum Defined as:
    - 0 = Onboard LED
    - 1 = EXT_LED1
    - 2 = EXT_LED2
    - 3 = EXT_LED3
  - uint8_t setLED(uint8_t uiLedNum, uint8_t uiLedValue);
    - Sets the LED to a specified value (ON/OFF)
  - uint8_t toggleLED(uint8_t uiLedNum);
    - Toggles the LED
      - If on turns off
      - If off turns on

Next you will need to design a button driver as well. It should be similar to the LED driver with functions of initializeButtonDriver and readButton.

If there are any questions please ask.