CST 347 – Lab 2 – Multiple Tasks and Task State Changes
Spring 2013 – Instructors Claude Kansaku / Troy Scevers

This lab will expose a few more RTOS kernel functions. Last time, we used the "task create" function to spawn a single task that sequenced the three PIC32 STARTER KIT LEDs in one of three patterns. A void * parameter reference was provided to the task at creation which held the pattern indication and the delay time in milliseconds.

In this lab, you will **create** multiple tasks, **delete** tasks, and **suspend/resume** tasks. Even if there are many ways to model this function, you are required to follow the program architecture specified here. You are only to use the RTOS functions provided as part of this Lab 2 Description.

Application Function:

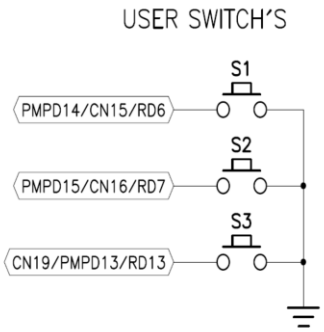
taskSystemControl: This task will poll the status of the three switches and act accordingly. Each of the switches will cause this behavior:

- SW1: The first time SW1 is pressed, LED1 will start flashing. The second press will start LED2, and finally, the third press will start LED3 flashing. Any subsequent presses will not do anything. This will be accomplished by doing a taskCreate for the next LED. Unlike last time, the task creation call must take back the task handle, which was the last parameter in the call. It was set NULL last time as the handle was not needed. However, in this function, the handle (one for each LED task created) will be necessary to delete the task. The task will toggle the appropriate LED and then do a taskDelay of 500 ms. This wil be done using an RTOS call versus the busy wait that was implemented in the last lab.
- SW2: Each time SW2 is pressed, the highest LED flashing will stop. LEDs will stop until all of them have stopped. After all have LEDs have stopped flashing, subsequent 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.
- SW3: A press of SW3 will "freeze" all current flashing LEDs. If fewer than three LEDs are currently flashing, SW1 will have the opportunity to cause any higher LEDs to start flashing as described above. A second press of SW3 will "unfreeze" and frozen LEDs. This is accomplished by suspending all currently running LED tasks on the "first" button press. The second button press will resume all created LED tasks.

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

Hardware Considerations:

The input will come via the three STARTER KIT switches SW1 (RD6), SW2 (RD7) and SW3 (RD13) as shown in the Figure to the right. The switches are raw and simply ground the respective Port D pin when pressed. They do not have external pullups and they not debounced. Both of these issues must be addressed by the software.




The first issue of pullups is handled by enabling the PIC32-provided internal pullups that are part of the Change Notification (CN) peripheral. Only the pin references that specify a CNx notion have this capability, such as those pins used for the STARTER KIT switches as illustrated in the Figure. You will notice that RD6 is associated with CN15, RD7 with CN16, and RD13 with CN19. These CN pins will need to have their pullups enabled. This PIC32 C environment provides a macro that performs the enable function in a similar way to interacting with the Port bits themselves. The following macro statement should be written into the prvSetupHardware() function.

```
ConfigCNPullups(CN15_PULLUP_ENABLE | CN16_PULLUP_ENABLE | CN19_PULLUP_ENABLE);
```

The second issue of debounce. The debounce is implemented with the three stage strategy:

- Read the switch
- Delay 10 ms
- Read again to verify change

The FreeRTOS API calls you will use are:

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

The API documentation can be found in Blackboard.