Oregon TECH

Computer Systems Engineering Technology

CST 347 – Real-Time Operating Systems

Lab 01 – Introduction to FreeRTOS Name______________________________

Possible Points: 10 Points

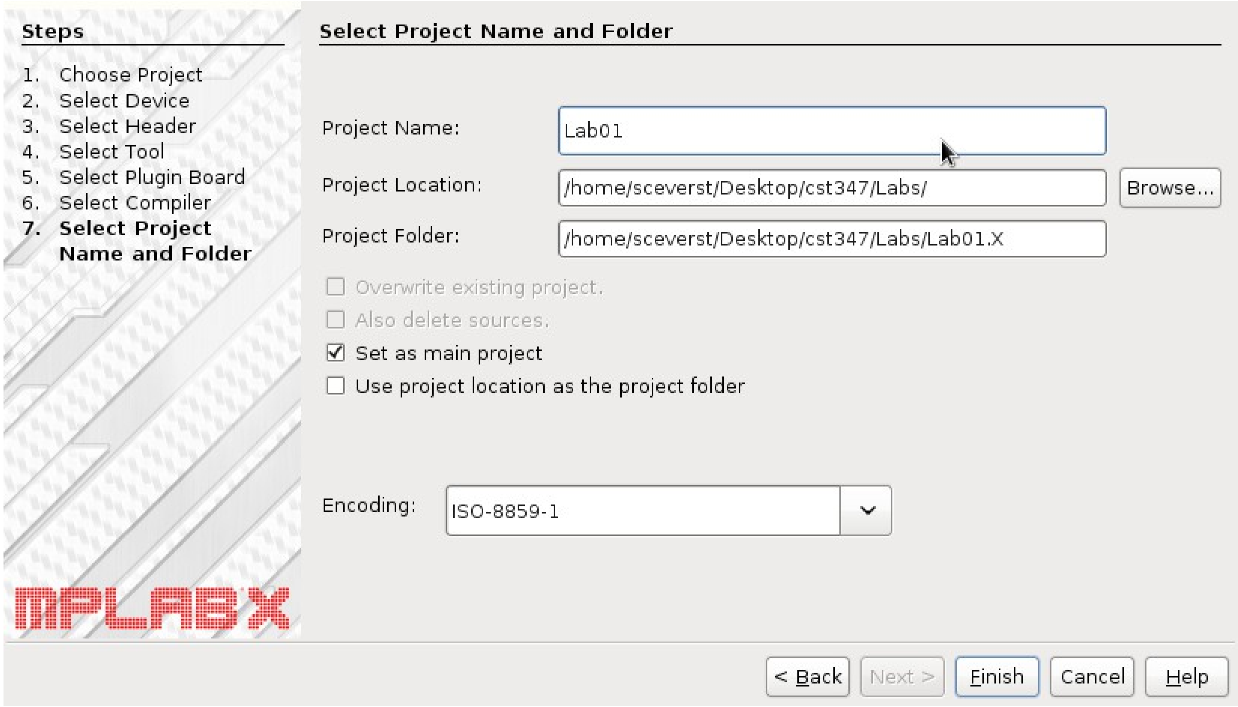
Instructions

Complete the following procedures. Zip up your final project and upload to blackboard.

Procedure

Part 1

1. Create a cst347/labs folder on your home directory of the lab machines.
2. Under cst347/labs create a directory called FreeRTOS
3. From the FreeRTOSV8.0.0.zip file (available on blackboard). Extract the following directories to cst347/labs/FreeRTOS.
    1. /FreeRTOSV8.0.0/FreeRTOS/License/
    2. /FreeRTOSV8.0.0/FreeRTOS/Source/
4. Using the MPLAB X wizard, create a project with the following properties:
    1. Microchip Embedded – Standalone Project
    2. Family: 32-bit MCUs (PIC32) – Device: PIC32MX360F512L
    3. Microchip Starter Kits: SKDE PIC32
    4. XC32 (v1.xx)
    5. Project Name: lab01
    6. Project Location: z:\cst347\labs\lab01
    7. Project Folder: z:\cst347\labs\lab01.X
    8. Check Set as main Project



Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. **Select Project Name and Folder**

Select Project Name and Folder

Project Name: Lab01

Project Location: /home/sceverst/Desktop/cst347/Labs/

Project Folder: /home/sceverst/Desktop/cst347/Labs/Lab01.X

☐ Overwrite existing project.

☐ Also delete sources.

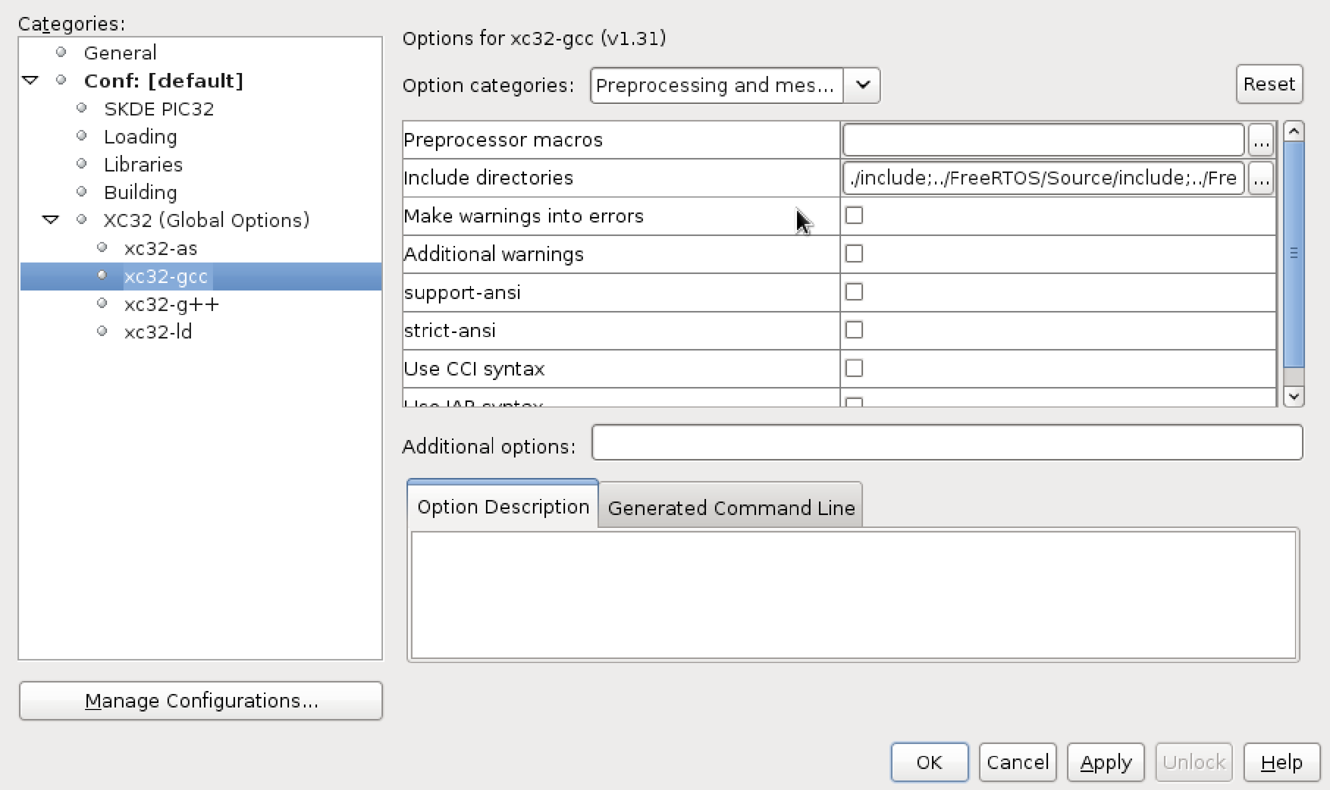
☑ Set as main project

☐ Use project location as the project folder

Encoding: ISO-8859-1

5. From the Lab01.zip file (available on blackboard). Extract the src and include directories to your cst347/labs/lab01.X folder. In your /cst347/labs/lab01.X/ folder you should have the following files: /cst347/labs/lab01.X/src/main.c and /cst347/labs/lab01.X/include/FreeRTOSConfig.h.
6. In summary, your folder structure should now be:
   - /cst347/labs/FreeRTOS/License
   - /cst347/labs/FreeRTOS/Source
   - /cst347/labs/lab01.X/include
   - /cst347/labs/lab01.X/src
7. In the Source Files logical folder, Add Existing Item…
   - \cst347\labs\lab01\src\main.c.
8. In the Source Files logical folder, create a FreeRTOS logical folder.
9. In the Source Files\FreeRTOS logical folder, Add Existing Item…files:
   - \cst347\labs\FreeRTOS\Source\list.c
   - \cst347\labs\FreeRTOS\Source\queue.c
   - \cst347\labs\FreeRTOS\Source\tasks.c
   - \cst347\labs\FreeRTOS\Source\portable\MemMang\heap_2.c
   - \cst347\labs\FreeRTOS\Source\portable\MPLAB\PIC32MX\port.c
   - \cst347\labs\FreeRTOS\Source\portable\MPLAB\PIC32MX\port_asm.S
10. In the Header Files logical folder, Add Existing Item…
    - \cst347\labs\lab01\include\ FreeRTOSConfig.h.
11. In the Header Files logical folder, create a FreeRTOS logical folder.
12. In the Header Files\FreeRTOS logical folder, Add Existing Item…files:
    - \cst347\labs\FreeRTOS\Source\include\croutine.h
    - \cst347\labs\FreeRTOS\Source\include\FreeRTOS.h
    - \cst347\labs\FreeRTOS\Source\include\list.h
    - \cst347\labs\FreeRTOS\Source\include\portable.h

   - \cst347\labs\FreeRTOS\Source\include\projdefs.h
   - \cst347\labs\FreeRTOS\Source\include\queue.h
   - \cst347\labs\FreeRTOS\Source\include\semphr.h
   - \cst347\labs\FreeRTOS\Source\include\task.h
13. Open lab01 Project Properties using the File menu or the Dashboard “wrench” icon. Select the XC32 (Global Options) – xc32-as option on the left. Add the following path to the “Preprocessor Include directories” option field: (Double-click to manually enter path)
   - ./include
   - ../FreeRTOS/Source/include
   - ../FreeRTOS/Source/portable/MPLAB/PIC32MX
14. You must do the same in the xc32-gcc option as well. You will have to select



Options for xc32-gcc (v1.31)

Option categories: Preprocessing and mes...

Preprocessor macros	
Include directories	./include;../FreeRTOS/Source/include;../Fre
Make warnings into errors	□
Additional warnings	□
support-ansi	□
strict-ansi	□
Use CCI syntax	□

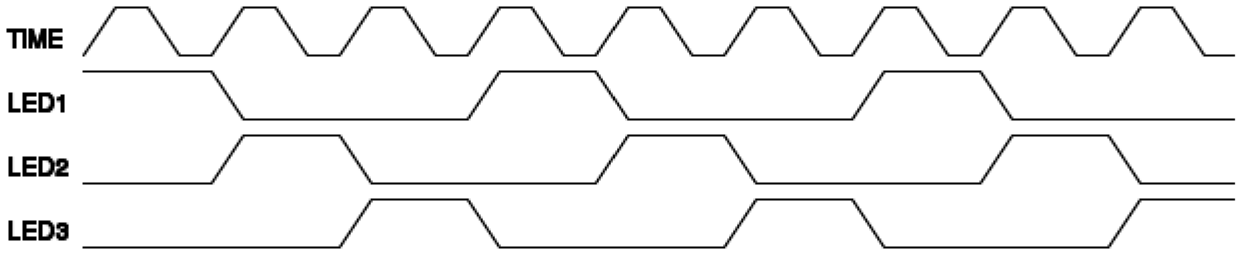
Additional options:

15. Compile the demo and run on the PIC32 starter kit. You should observe the LED's blinking at various rates.
16. Take a minute to go through main.c and see if you can figure out what is going on. Ask the instructor if you have any questions.
17. Make a backup of this project as we will use it as a base going forward....

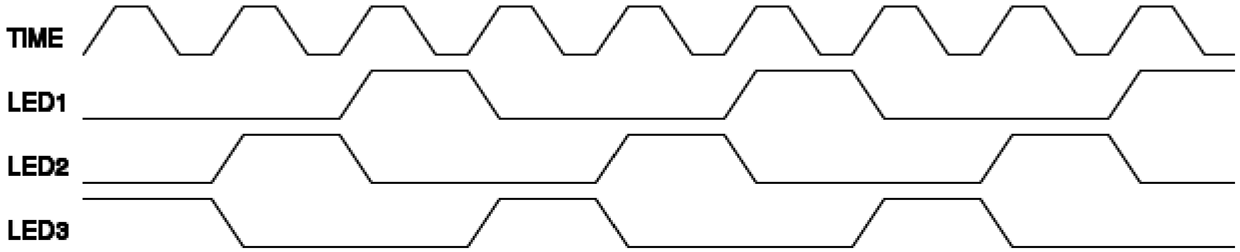
Part 2

1. You will now modify this demo project to do the following:
   1. Create a new set of files called leddrv.c and leddrv.h and add them into your project (These should be stored in the include and src directories of your project)
   2. In the led files write an LED “driver” with the following functions:
      1. *uint8_t initializeLedDriver(void)*
         1. This function will setup the ports for the LEDs and set them to the OFF state. Returns a 0 for success, any other value for failure.

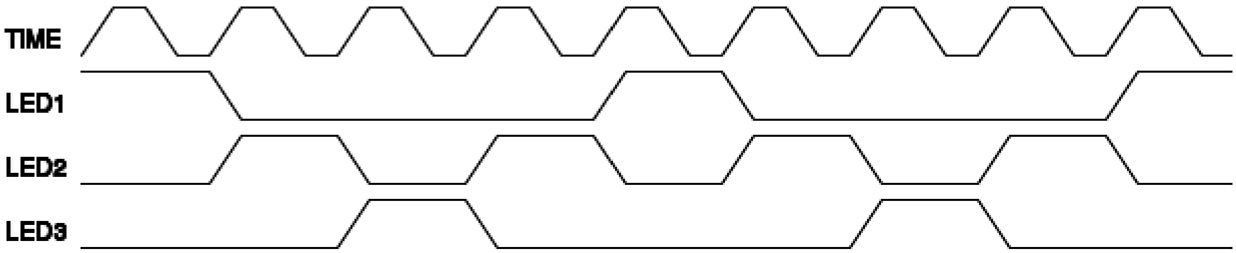
   2. *uint8_t readLed(uint8_t ledNum)*
      1. This function will return the current state of the given LED Number. 0 for off, 1 for on, any other number is an error condition
      2. ledNum is defined as:
         1. 0 = LED1 (RD0)
         2. 1 = LED2 (RD1)
         3. 2 = LED3 (RD2)
   3. *uint8_t setLED(uint8_t ledNum, uint8_t value)*
      1. Sets ledNum to a state of OFF or ON depending on value. If value is 0 turn OFF LED, any other value will turn ON LED. Returns a 0 for success or any other number for failure
   4. *uint8_t toggleLED(uint8_t ledNum)*
      1. This function will toggle the current state of the LED. If the LED is OFF it will turn it ON, if LED is ON it will turn it OFF. Returns 0 for success, any other value for error.
3. Create another set of files called myTasks.c and myTasks.h and add them to your project (These should be stored in the include and src directories of your project)
   1. You will also have to modify your main.c file to include this new header file myTasks.h and ledDrv.h
   2. You will instantiate new tasks in *main()* and call the *InitializeLedDriver()* from *prvSetupHardware()*
4. Write a task that uses your ledDriver to perform three different LED light patterns as follows:
   1. Strobe down continuously



   2. Strobe up continuously



   3. Bounce up and down continuously



5. The task will use two parameters to set the LED ON/OFF time and pattern. You will create the parameter structure similar to the example provided.
6. Demonstrate each of these patterns to the Lab Instructor