ECEN 3xxx

Real Time Operating Systems Lab #3

Tasks

Spring 2020

Objective: Utilizing the Micrium RTOS, create a multi-task system that samples GPIO and the capacitive touch slider and converts the raw input signals to output signals visible on the LEDs.

This lab builds upon the code developed in Lab 2 Part I for sampling the pushbuttons and capacitive touch slider and driving the LEDs. Instead of polling from a main loop, separate tasks will be created to sample each input device and to drive the LEDs. Three tasks will be created:

1. Button Input Task – Samples the GPIO state of each pushbutton and stores the sampled state in a global variable for each pushbutton.
2. Slider Input Task – Samples the sensed position of the capacitive touch slider and stores the position in a global variable.
3. LED Task – Controls the lighting of LED0 and LED1 based on the sampled value of each pushbutton and the capacitive touch slider.

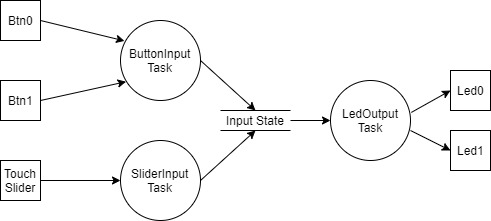


Figure 1: Task Diagram

The Micrium uC-Probe tool and the Segger SystemView tool will be utilized to analyze task execution and performance. An Idle Task will be created that places the device into low power mode when no other tasks or interrupts are running. The Energy Profiler will be used to compare the power consumption between the multi-task system (with and without the Idle Task enabling low power) and the two implementations from Lab 2.

References:

1. Micrium Documentation: <https://doc.micrium.com/display/OSUM50700/Micrium+OS+User+Manual>
   1. https://doc.micrium.com/display/TECHOV/Introduction+to+Kernels
   2. https://doc.micrium.com/display/TECHOV/Tasks
   3. https://doc.micrium.com/display/TECHOV/Time+Management
   4. https://doc.micrium.com/display/OSUM50700/Initialization+of+the+Kernel
   5. https://doc.micrium.com/display/OSUM50700/Kernel+Basic+Services
   6. https://doc.micrium.com/display/OSUM50700/Task+Management+Kernel+Services
   7. https://doc.micrium.com/display/OSUM50700/Kernel+Core+API
   8. https://doc.micrium.com/display/OSUM50700/Kernel+Task+Management+API
2. uC-Probe User’s Manual (available from the Help menu within the uC-Probe application)
3. Segger SystemView User Guide (available from the Help menu within the SystemView application)

Preparation:

1. Review the MicriumOS Documentation
2. Install the Micrium uC-Probe application from <http://micrium.com/tools/ucprobe/software-and-docs/> and review the documentation.
3. Install the Segger SystemView application from <https://www.segger.com/downloads/free-utilities/#SystemView>, install Segger J-Link from <https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack>, and review the documentation.
4. Download and run the SLSTK3402A\_micriumos\_blink Software Example to become familiar with MicriumOS initialization and task creation.

Procedure:

Part I – MicriumOS Integration and Task Creation

1. Make a copy of the project RTOS\_Lab2\_GPIO\_Timers\_Interrupts. Rename the project to RTOS\_Lab3\_Tasks.
2. Referring to the Software Example above:
   1. Add the Micrium source folders and include paths to the project.
   2. Add the API calls to main() as required for Micrium initialization and creation of the Startup task.
   3. Add the Startup task main function.
3. Define the stack and TCB variables for each of the three tasks.
4. Create the main function for each of the three application tasks (ButtonInput, SliderInput, and LedOutput).
5. Define the contents of each of the main functions for each application. Each main function should perform the following:
   1. Initialization of hardware that is “owned” by that task
   2. Infinite while loop
   3. Inside the while loop, add the call to the function that performs the processing that is specific to each task (e.g., sampling input or controlling LEDs); call OSTimeDly() to delay between each loop iteration.
6. Inside of the Startup task, add calls to OSTaskCreate() for each of the application tasks.
7. Build and run the project.
8. Set a breakpoint in the main function of each task to verify that each task is running.
9. Perform each of the functional tests from Lab2 to verify that the external behavior of the application is the same as from Lab2.

Part II – Using Micrium µC-Probe to analyze the system

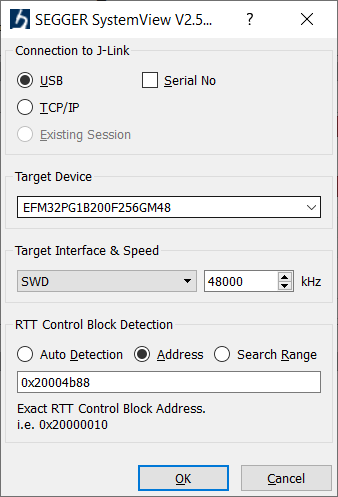
1. Add a copy of cpu\_cfg.h to your project that can be modified locally
2. Follow the instructions in Appendix B of the uC-Probe User’s Manual for adding Kernel Awareness to your project (B-1-5 Micrium OS Kernel Requirements).
3. Rebuild the project and run the project from the debugger
4. Launch the Micrium uC-Probe application
5. Create a new project in uC-Probe and add the Micrium OS Kernel screen to the project
6. Click Run (make sure the firmware application is also running and not stopped at a breakpoint)
7. Go to the Task(s) tab in uC-Probe
8. Verify that each of the application tasks is listed (ButtonInput, SliderInput, and LedOutput) and that the Context Switch Counter is incrementing, showing that the tasks are running.
9. In addition to the tasks that your application created, what other tasks are also running? Describe the purpose of each one (refer to MicriumOS documentation).
10. Record the CPU Usage and Stack Usage of each task.
11. Determine the period of each task.
12. Save a screenshot of the application, showing the Task(s) tab.

Part III – Using Segger SystemView to analyze the system

1. Add the SystemView source folder to your project
2. Set OS\_CFG\_TRACE\_EN to DEF\_ENABLED in your local copy of os\_cfg.h
3. Make a local copy of SEGGER\_SYSVIEW\_Conf.h. Increase SEGGER\_SYSVIEW\_RTT\_BUFFER\_SIZE to 16 KiB. Leaving the default value of 4 KiB may cause event overflow errors resulting in loss of recorded data.
4. Rebuild the project and run the application from the debugger.
5. Launch the Segger SystemView application

(stop the uC-Probe application if it is still running).

1. Lookup the address of the RTT Control Block from the generated .map file (symbol name \_SEGGER\_RTT)
2. Click the record button (green arrow). You should see the following dialog box displayed:



1. Under RTT Control Block Detection, enter the address of \_SEGGER\_RTT found in the .map file and click OK.
2. If you see a message about SystemView overflow events recorded, then you will need to increase the buffer size by adjusting SEGGER\_SYSVIEW\_RTT\_BUFFER\_SIZE as described above.
3. You should see data being updated on the screen.
4. After several seconds, press the stop button (red box) to stop the recording.
5. Save a screenshot of the SystemView application.
6. Expand the Timeline to zoom in on a section where which each of the application tasks is running. Analyze the events in the event log to observe when each task is made ready and run.
7. What do you observe about the scheduling of each task? How do interrupts and the priority of other tasks affect scheduling? Does the scheduling of each of the application tasks that you created seem optimal?
8. Record the frequency and min/max run time of each task

Part IV – Idle Task and Low Energy Mode

1. Start the Energy Profiler and record the average power consumed at nominal conditions (no buttons pressed, slider not touched). Compare this value with the values recorded in Lab 2.
2. Disable the default Idle Task by setting OS\_CFG\_TASK\_IDLE\_EN to DEF\_DISABLED in os\_cfg.h
3. Create a new task to replace the default Idle Task. Set its priority to the lowest of all other tasks, including kernel tasks.
4. Inside the Idle Task, create an infinite while loop that calls EMU\_EnterEM1().
5. Build and run the application. Verify with SystemView that the new Idle Task is running at the lowest priority of all other tasks.
6. Start the Energy Profiler and record the average power consumed at nominal conditions (no buttons pressed, slider not touched). Compare this value with the value recorded in Step 1.

Grading:

1. 25 points equals “100%” for this lab.
2. All project files and source code must be submitted, such that results can be duplicated by the grader. ZERO score if all project files and source code are not submitted.
3. Basic Construction:
   1. Project builds without warnings and successfully executes, with each of the three application tasks running: 6 pts
   2. Idle task is placing the system into low energy mode: 2 pts
   3. All functional tests from Lab 2 pass: 2 pts
4. uC-Probe Measurements:
   1. Recorded CPU Usage, Stack Usage, and Period of each task with screenshot: 2 pts
5. Segger SystemView Measurements:
   1. Recorded observation of scheduling behavior with screenshot: 2 pts
   2. Frequency and min/max run time of each task: 2 pts
6. Power Measurements:
   1. Analysis and comparison of power consumption of tasking method both with/without placing the system into lower power mode from the Idle task, and comparison with each implementation in Lab 2: 3 pts
7. Good Coding:
   1. Header documentation has been provided for all functions: 2 pt
   2. Header documentation follows Doxygen guidelines: 2 pt
   3. Appropriate use of comments inside functions: 2 pt
8. Bonus points (will add to the above-listed 25 pts if the lab is turned in on time—which can push your lab#2 grade above 100%)
   1. Describe the effect on scheduling when the following changes are made. Use Segger SystemView to observe the change in scheduling of each task. In your observation, look for changes in when each task is run relative to the others and the delay between when a task is made ready until it actually runs.
      1. Relative priority of each application task is changed (2 pt)
      2. Relative frequency of each application task is changed (2 pt)
      3. Processing time spent by each task is proportionately increased up to the point that the Idle Task consumes < 10% of the CPU (2 pt)

Point Deductions for Late Submission: -5 points per day