Department of Engineering

EE 4710 Lab 7

Title: Instrumentation of Real-Time Systems

Objective: The student should understand how to add instrumentation code to a real-time system and to measure context switch overhead, task execution time and idle time.

Parts: 1-C8051FX20-TB Evaluation Board
1-USB Debug Adapter
1-DB-9 Serial cable (USB adapter cable is also ok)

Software: Silicon Laboratories IDE version 3.50.00 or greater. Keil compiler.

Background: In terms of software, the term “instrumentation” implies the addition of additional software code that aids analysis. In the case of real-time systems, this often means driving one or more unused port lines during certain times to determine the frequency, duration or phase of an event. For example, if a particular port line is driven high when a context switch starts and is driven low when the context switch concludes, we can use tools such as oscilloscopes or logic analyzers to determine when a context switch occurs and how long it takes.

Preparation: Write the title and a short description of this lab in your lab book. Make sure the page is numbered and make an entry in the table of contents for this lab.

In this exercise, we will modify the code from lab 6 so that port 2 can be used to output status information. First, modify your code from lab 6 by adding a global 1-byte variable called task_id_mask. This variable must be in the data (not idata or xdata) space and must be visible in context.asm. Modify your scheduler to set this variable to 1 when the background task is scheduled, 2 when the highest priority task is scheduled, 4 when the next highest priority task is scheduled and so on.

Add an instruction as early as possible in the timer 2 interrupt service routine to set P2 to 0x80 (80H). Add a second instruction as late as possible in the interrupt service routine to set P2 to the value of task_id_mask. This will cause P2.7 to be high during the context switch, P2.0 to be high during the execution of the background task, P2.1 to be high during the execution of the highest priority task and so on.

Find the longest non-preemptive critical section (which is usually in the code that releases semaphores). Just after interrupts are disabled, set P2.6 high and right before they are enabled again set P.6 low. This will cause P2.6 to be high during the longest non-preemptive critical section. (If there are multiple critical sections that are candidates for the longest critical section, instrument them too.)

Finally, with only the weak pull-ups of the 8051, the outputs will not switch fast enough, so somewhere in the initialization set P2MDOUT to 0xFF (0FFH).

After writing this code and compiling it without error, bring it to your scheduled lab session.

Procedure: Using an oscilloscope, verify that the signals on P2.0-P2.3, P2.6 and P2.7 are reasonable based on your understanding of the program.

Use the cursor feature of the oscilloscope to measure the maximum context switch overhead. (There may be jitter because the execution time of the scheduler is likely not constant. If so, measure the largest duration pulse.) Record this time in your lab book.

Repeat the procedure for the non-preemptive critical sections. Record the time of the longest critical section in your lab book.

Find the execution time and self-suspension time for the highest priority task. For this task, the execution time comes in two short bursts. We would model this task by summing the execution times and by treating the gap between them as self-suspension time. Add these bursts together to get the total execution time in the (500ms) period. Record this time and the self-suspension time in your lab book.

Find the execution time of the two periodic tasks. You will note that these tasks are interrupted by one or more unnecessary context switches. This happens when the scheduler gives the processor back to the interrupted task. These context switches exist to facilitate a round-robin scheduler, which our scheduler is not. Just include the times of the unnecessary context switches in the execution times of the tasks. Record these times in your lab book.

Estimate the utilization of the processor by measuring the time spent in (or more likely out of) the background (aperiodic) task (P2.0). Record the execution time of this task and your estimate of the utilization by the periodic tasks.

Now suppose you wish to send data faster. Use your measurements to find a period shorter than 500ms that will still allow all deadlines to be met. Modify your code to use that period (which may mean changing the timer 2 reload value). Recompile your code and run it. Verify that your program still works (but is faster). Connect the scope to P2.0 and record your observations. Demonstrate this to your lab instructor.

Affix all your source code to your lab book then write a summary or conclusion. Remember to sign or initial then date each page.