C S4680-101_EMBEDDED SYSTEMS (SPRING 2021)

Grad Project (ACX 4): x_delay - Delay a thread by specified number of "system ticks"

Background

Many real-time and embedded system programs must perform a function on a periodic basis, or a sequence of operations with specified delays between operations in the sequence. A temperature control application must monitor temperature changes on a periodic basis; a solenoid based actuator usually requires one voltage to be applied to "pull in" the solenoid and another to "hold" it in place--the pull phase might last 50 milliseconds and then the voltage is changed to the "hold" phase.

There are not always enough hardware resources (timers) to dedicate to each timed activity in an embedded system. Providing some centralized service for this "delay" requirement is often a more efficient way to handle these situations, though it may be at the cost of precision. Implementing this capability in ACX requires:
(1) A periodic interrupt source
(2) At least one delay counter variable for each thread that needs to perform timed delays
(3) An ISR that handles the periodic interrupt source and updates the delay counters each time it is invoked
(4) A function that initializes a thread's delay counter and sets its status to "delayed".

Periodic Interrupt Source

We will use the Atmega2560 Timer0 to generate a periodic interrupt that we will call the "system tick". Choosing the interrupt frequency is an important design decision in an embedded system. Since the time overhead for servicing the interrupt will be approximately constant (once designed), a shorter interrupt period will result in this time consuming a larger percentage of the processor's available cycles compared with a longer period. However, the shorter period will provide higher timing resolution than a longer period. This tradeoff must be made carefully since, once chosen, it will affect all threads that use the timing features. If possible, we would like to keep the system tick overhead under 1% of CPU cycles. For example if the processor runs at 16 MHz (assuming 16 million instructions per second) then it will execute 16000 instructions every millisecond. If our system tick is set for 1 msec (every 16000 instructions) then it would be best if the ISR consumes less than 160 cycles (1% of 16000). If our tick occurs every 5 msec, then the ISR can consume up to 800 cycles and be within a 1% budget. The Atmega2560 runs at 16MHz, therefore at a 1 msec system tick rate an ISR that consumes 160 cycles would incur a 1% CPU overhead.

Thread Delay Counters

Each thread will need a delay counter as part of its "state" stored in the kernel. In your acx.c file you should have already created an array of delay count values, one per thread. This array should be named x_thread_delay. The **x_thread_delay** array should be of **type uint16_t** to allow for maximum positive range for a 16-bit integer. These counters will be updated by the Timer0 ISR when a thread is delayed.

Timer0 ISR

When a Timer0 interrupt occurs, the Timer0 interrupt handler should decrement each thread delay counter. If the resulting counter value becomes zero, the x_delay_status bit corresponding to that thread should be cleared to zero, thus removing the delay blocking condition from that thread. Add this functionality to your Timer0 ISR in your **acx.c** file.

The x_delay Function

The **x_delay** function causes a thread to place itself in a "blocked" condition for a specified number of "system ticks". If there are other READY threads, then one of them will be selected by the scheduler to be placed into execution. Write the x_delay function so that it:

- copies the delay value into the calling thread's delay counter
- sets the x_delay_status bit corresponding to the calling thread's ID, then
- calls x_yield to initiate thread rescheduling

NOTE: Since the x_thread_delay counters are shared by both ISR and non-ISR code, access to them must be made "atomic"--that is, we don't want the ISR to interrupt writes to a multi-byte variable, therefore we must disable interrupts around updates to these counter variables. Use the ATOMIC_BLOCK macro. Also, since they are updated by ISRs, it will be necessary to declare them **volatile** to avoid having the C optimizer eliminate accesses to them.

Details and Testing

- Make sure you have created the thread delay counter array and named it **x_thread_delay**
- Implement the Timer0 ISR so that it correctly decrements the delay counters and clears the x_delay_status bits as needed
- Implement the x_delay function

Test your thread delay support by creating a test program in which you create two threads where each thread blinks an LED at a different rate. For example, let the first thread blink an LED every 1.5 seconds and let another blink a second LED every 500 msec using your x_delay function rather than the library software delay utility _delay_ms.

Submission

Demonstrate your working program to your instructor and upload your updated project acx.zip file at the prompt.

Submission status

Submission status	Submitted for grading
Grading status	Not graded
Due date	Tuesday, April 20, 2021, 11:55 PM
Time remaining	Assignment was submitted 5 hours 22 mins early
Last modified	Tuesday, April 20, 2021, 6:32 PM
File submissions	ACX04.zip April 20 2021, 6:32 PM
Submission comments	Comments (0)

◄ Grad Project (ACX 3): x_new - Create a new thread

Jump to...

Grad Project (ACX 5): Complete Remaining ACX functions ►