Software Timers for the 68000

By Joe Bartel, Hawthorne Technology

In most process control projects, and many interactive programs, there is a need to sequence and time operations. These are some techniques I have developed over the years and have converted into 68000 assembler code. A similar method will work with almost any other processor.

/m/mFor most projects there is a trade off between more software and more hardware to accomplish a given task. Using more software means the unit cost will be lower, but it also means there may be more developement costs involved. Software needs less board space than hardware does and it uses much less power. The purpose of a standard software item is to provide the same convenience that an LSI part like a UART or PIA does. You can use them and you don't have to design the parts each time. Also by reusing the same parts you can get more done with less effort and fewer bugs.

If the time interval is very short, less than a millisecond, then it is hard to use software instead of another hardware timer. If the time interval is longer than 100 msec and the resolution is 10 msec then there is no reason to use hardware. With a software timer you can have as many timed intervals as you want, but you only pay for one hardware timer. The internal timer on the 68681 used in the HT68k is an example of this.

When doing timing I like to divide the servicing of the timer interrupt from the actual action that is performed. The goal is to spend as little time as possible inside the interrupt service routine. This lessens the chance of losing an interrupt and makes more time available for the time critical services like serial input devices. To do this I use flags to indicate that an operation is to be performed. A non interrupt loop tests the flags and does the operations indicated.

```
IN LINE CODE:
INCT1   MOVE.W    COUNT1(A5),D0  ;TEST OLD
        BEQ.S     INCT2          ;IGNORE IF Z
        SUBQ.W    #1,COUNT1(A5)  ;DEC COUNTER
        BNE.S     INCT2          ;NO ACTION IF NZ
        ST        FLAG1+1(A5)    ;SET FLAG IF Z
        MOVE.W    LOAD1(A5),COUNT1(A5)    ;RELOAD COUNTER
INCT2   RTS

SUBROUTINE
        LEA       COUNTER1(A5),A0  ;POINT TO STRUCTURE
        BSR       SOFTCOUNT
        LEA       COUNTER2(A5),A0  ;#2
        BSR       SOFTCOUNT
        LEA       COUNTER3(A5),A0  ;#3
        BSR       SOFTCOUNT
        RTI
```

A soft counter is defined as a data structure in memory which has a counter that is counted down each time a timer interrupt occurs. There is a reload value to reload the count after it is counted down, and a flag that is set when the count has been counted down to zero. This is a flexible arrangement that can be used for one shot timers, continuous timers, or other timer types.

First, clear the flag to indicate that the timer is about to start. Then if it is to be continuous, set the reload value. If the time is to be a one shot timer, then the reload value should be zero. To start the timer, set the count to a nonzero value. The timer will then count down on each time tic until it reaches zero. At that time it will set the flag. To find out if the time has expired you check the flag at convenient intervals. Once the soft timer has been set and is runniing it will continue in the background almost like a hardware timer. For long time intervals in the millisecond or second region this is a very low cost way to have many timers. The single hardware timer on the ht68k can be used in this manner to provide a variety of timed intervals.

A common method of using the timer flags is to have a single very large loop for the main program. The loop is a series of tests of the various flags that have been defined. If a flag is set, the action routine that it was timing is done, and the flag is cleared. For physical events the accuracy will be good enough in most cases. As more timers are added, then the accuracy of the timing for each event may be less then when there are few timers. One approach to this part of the problem is to have the most time critical event flags at the start of the flag test loop. Instead of testing the next flag, a routine that takes a long time to execute can go back to the top of the test loop. This would insure that only one long routine gets executed each time through the scan loop. The same loop can also respond to flags set as the result of other kinds of interrupts or events.

An example of using a software timer is in a communication program. When a reply is expected the count value can be set to the maximum amount allowed, the flag cleared, and the reload value set to zero. The flag is tested and if any interval causes it to time out then the flag will be set. This can be used to abort a call if there has been no activity for a certain time period.

An example of using a periodic counter is a logging or reporting system. The counter and the reload value are set to the log interval. The flag is checked very often. When the flag is found set, a log record is generated and recorded. After the logging is done the timer flag is cleared and the wait for the next time proceeds.

A series of sequential events can be controlled by a single timer if a state variable is used, or a series of times can be used. When the flag for the first timer is set, the action for that flag is done. Then the flag for that event is cleared, and the timer for the next event in the sequence is started. Eventually the time for the first event is started and the cycle starts all over again. In this case each timer is acting like a one shot that triggers the next one shot in the series.

This has been a short discussion of how to time various events with only a single hardware timer. The techniques will work on any processor but are especially well adapted to the 68xxx family. With this type of programming it is possible to trade the faster speed of the 68000 for less complex, less expensive hardware. Also by having a simple generic timer routine that can be inserted like a hardware block it is possible to create larger, more complex programs without greatly increasing the time needed to debug the routines involved. By having the action part of the routine separated from the timer part, it is also easier to get less overall variation in timing, and almost eliminate time drift. ■