Obtaining Accurate Timings in the Presence of Clock Interrupts.

Lory D. Molesky

January 6, 1992

When one wishes to determine experimentally how long a section of code takes to execute on a 136A board, the Z8036 clock is used. If not performed carefully, timing measurements taken in the presense of clock interrupts can include the cost of the ISR (interrupt service routine). Excluding the time of the clock ISR is particularly important when one is timing code of short duration. This application note addresses this problem and describes techniques to obtaining accurate measurements in the presense of clock interrupts. These techniques are currently in use on the SP (system processor) board of the Spring kernel.

On the SP, the Z8036 clock interrupts the OS every 10 ms. The interrupt service routine called re-enables the clock, then updates the software clock locally (on the SP), and remotely (on all the AP's). This interrupt service routine has a worst case execution time of 24.5 us. under no bus or memory contention by the AP's [1]. When performing timing experiments of which the sampled intervals are always less then 10 ms., we can avoid including the execution time of the clock ISR. This can be accomplished by filtering out samples which *might* include the execution of the clock ISR. The endpoints of a sample is defined by a `start` value of the clock and a `finish` value of the clock. The `elapsed` time is defined as `start - finish` (the timer counts down from 20000 ticks to zero, a tick is .5 usecs). Given these definitions of `start, finished` and `elapsed`, the precise definition of this filter follows:

- if `elapsed` is negative, the sample is excluded.
- if `start` >= 19996 ticks, the sample is excluded.

This definition of a filter is useful since it is independent of the sample time, given the sample duration is less than the period (10 ms.) Point 1 is fairly obvious. If `elapsed` is negative, The timer has counted down from 20000 to past 0 and has thus expired, and most likely the ISR computation time has been included in the sample. Point 2 is not so obvious. This behavior has been detected experimentally, and can be explained by a maximum delay of 4 ticks between the time when the countdown timer reaches 0, initiating the interrupt sequence, and the actual time which the ISR is called.

[1] All timings are performed with the cache off.

The following code illustrates the use of three macros for use on the SP. READCLOCK_FAST(arg1) performs a read of the Z8036 clock, assigning its argument to that value (in units of half micro-seconds). The call to STORECLOCK_FAST(arg1,arg2) marks the end of the timing interval; arg2 is assigned to the elapsed time (in half usecs). VALID_READ(arg1,arg2) is the filter which ensures that the execution time of the ISR was not included in the measurement.

```
   {
      short start_cd;         /* Starting value of the countdown timer */
      int elapsed_half_usecs; /* Number of half usecs */
      int invalid = 1;

      while (invalid) {
         READCLOCK_FAST(start_cd);
         /* Put code to time here. */
         STORECLOCK_FAST(start_cd,elapsed_half_usecs);
         if (VALID_READ(start_cd,elapsed_half_usecs)) {
            /* elapsed_half_usecs
            invalid = 0;
         }
     }

#define READCLOCK_FAST(CD) \
*(char *) CIO_CT2CS = 0xCE; \
CD = *(short *) CIO_CT2CCM;

#define STORECLOCK_FAST(START_CD,ELAPSED_HALF_USECS) \
{ \
    short cd;\
    short finish_cd; \
    READCLOCK_FAST(finish_cd); \
    ELAPSED_HALF_USECS = START_CD - finish_cd; \
}

#define VALID_READ(START_CD,ELAPSED_HALF_USECS) \
(ELAPSED_HALF_USECS > 0 && START_CD < (MAX_USEC - 4))
```