Harman
68332 Book

15.3.3 Interrupts and Latency

In this textbook, the designation *real-time* system refers to a system or product that must respond and generate some timely action in response to external events. Typically, a real-time system performs data acquisition and control under strict timing constraints. One of the primary parameters defining a real-time system is the time-related performance. The *performance* with respect to time is determined by the system response time to external events and other factors such as its data transfer rate. In this subsection, one key parameter that affects the system response time is taken to be the *interrupt latency*. The interrupt latency time is the time lag necessary to respond to an interrupt and pass control to the interrupt handling routine. The total interrupt response time for the product to recognize and service an interrupt is another parameter that determines system performance.[7]

Interrupt latency. The minimum interrupt *latency time* for the CPU32 is calculated by adding the times of the following contributions:

(a) completion of current instruction;
(b) acknowledgement and vector acquisition;
(c) stacking of format word, PC and SR;
(d) read of vector address from CPU vector table;
(e) fetch of first instruction and prefetch of next instruction.

This list of contributions to the latency follows from the discussion of interrupt handling in Section 11.7. Items (b) through (e) represent the interrupt *processing* by the CPU as the term is defined in Section 11.7. The entire interrupt cycle including interrupt processing, executing the interrupt routine, and restoring control to the interrupted program is usually called interrupt *handling* or interrupt *servicing*. Figure 15.5 shows the complete interrupt cycle and defines the various parameters of interest.

For a single interrupt source, the total execution time of the interrupt routine is the sum of the time for interrupt acknowledgement, the time for context switching, the time of execution of the instructions in the interrupt routine, and the time to restore the original context and return to the interrupted program. The *context switching* time can be defined as the time for interrupt processing by the CPU. In MC68332-based products, the interrupt processing includes the stacking of the format word, Program Counter and Status Register values and the time to fetch the interrupt vector address and pass control to the interrupt handling routine. The first instructions of the interrupt routine typically save register values and other data that must be restored after the interrupt routine completes.[8] After executing the instructions that perform the

[7] A number of other factors determine the overall response time of the system or product. The operating speed of the CPU and access time to external memory are two items that affect the response. Also, other characteristics than response time may be important in determining the system performance. Reliability, fault tolerance and similar factors are critical specifications for certain real-time systems. Such factors are not considered in this textbook.

[8] The *context switching* time is sometimes defined to include both the time for interrupt processing and the time taken to save registers and other data. In M68000 family processors, the saving of the contents of general-purpose registers is not automatic. Thus, instructions such as Move Multiple Registers (MOVEM) are required in the interrupt routine to save (and restore) general-purpose register contents.

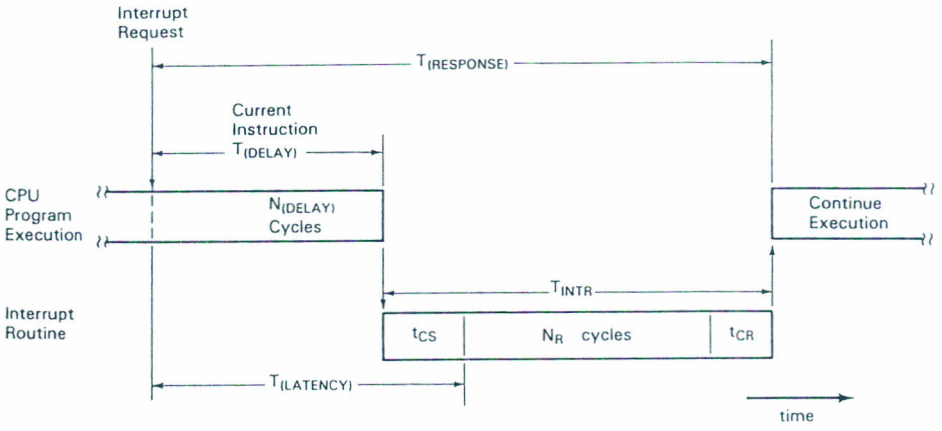

Figure 15.5 Interrupt timing parameters.

required action to service the interrupt, general-purpose register contents must be restored by program instructions. The context switch back to the interrupted program is accomplished with the Return From Exception (RTE) instruction described in detail in Section 10.2.

The parameters in Figure 15.5 are defined as follows:

$T_{(\text{DELAY})}$: time between interrupt request and processing
t_{CS} : time for interrupt processing (context switch)
$T_{(\text{LATENCY})}$: time between interrupt request and
: execution of the interrupt routine
T_{INTR} : time for interrupt handling and context switching
t_{CR} : time to restore context
$T_{(\text{RESPONSE})}$: total time for interrupt servicing.

The time for interrupt servicing, T_{INTR}, includes both the context switch time (t_{CS}) and the time t_{CR} to return control to the CPU program.

Neglecting instruction overlap and special instructions such as RESET and RTE, the CPU instructions take from 2 to 64 clock cycles (DIVS.L) to execute. To this time must be added the number of cycles to compute the effective addresses of any operands. Depending on the number of clock cycles to read and write memory, the interrupt processing may be delayed from 2 clock cycles to more than 70 cycles plus any wait states (1 clock cycle each) added because of slow memory. The additional delay before the interrupt routine executes depends on the memory access time as the CPU performs stacking, fetching of the vector address from the CPU vector table and fetching of the first instruction. Section 14.5 defines the number of system clock cycles for various types of CPU memory accesses. CPU accesses to the on-chip RAM or external accesses using fast termination take two clock cycles. Most external memories

require at least three clock cycles for a CPU read or write cycle if no wait states are necessary.

Example 15.4

Figure 15.5 shows the complete interrupt cycle. Assume that the present instruction takes $N_{(\text{DELAY})}$ clock cycles to complete before interrupt processing begins. The latency time until the first instruction of the interrupt routine begins execution is

$$T_{(\text{LATENCY})} = T_{(\text{DELAY})} + t_{\text{CS}} \tag{15.1}$$

For a system clock frequency of $f_{(\text{system})}$ Hertz, the clock cycle time is

$$t_{\text{CYC}} = 1/f_{(\text{system})} \text{ seconds.}$$

Since $T_{(\text{DELAY})} = N \times t_{\text{CYC}}$ for the current instruction to complete, the latency time from Equation 15.1 is thus

$$T_{(\text{LATENCY})} = N_{(\text{DELAY})} \times t_{\text{CYC}} + t_{\text{CS}} \tag{15.2}$$

For the MC68332, timing estimates for the interrupt latency can be obtained from the instruction timings given in the *MC68332 User's Manuals*. For this example, 70 cycles will be assumed as the worst case delay for the current instruction to complete. The interrupt processing time t_{CS} includes 4 memory read cycles for interrupt acknowledgement (1 cycle), reading the vector address from the CPU vector table (2 cycles) and fetching the first instruction from memory. This requires 39 clock cycles if each memory access requires 3 clock cycles. Substituting these example number of cycles in Equation 15.2 for a clock period of 59.6ns (16.78 MHz) yields the result

$$T_{(\text{LATENCY})} = (70 + 39) \times 59.6\text{ns}$$

or 6.5μs.

According to Motorola's data for instruction timing, the RTE instruction takes 30 clock cycles to restore control to the interrupted program. Thus, the time the CPU program is interrupted would be

$$T_{\text{INTR}} = (39 + N_{\text{R}} + 30) \times t_{\text{CYC}} \text{ seconds}$$

where N_{R} is the number of clock cycles required by the interrupt routine. The calculation of the number of cycles required by the RTE instruction is based on the four read cycles required to restore the 4 words of information placed on the stack during context switching and two additional read cycles to fetch the first two words from the interrupted program. It is assumed here that each read cycle requires 3 clock cycles. The other clock cycles taken by the RTE are for internal processing.

Interrupt priorities. When multiple interrupt sources are present, interrupt priority must be established. The purpose is to assure that the most important interrupts are serviced within a predefined time regardless of other events including lower priority interrupt requests. In many products, interrupt servicing for one interrupt may be interrupted by another, higher priority interrupt. To determine whether the

product's timing specifications can be met, the product designer must estimate the latency and the execution time of the interrupt handler for each interrupt source.

Several factors can delay the execution of an interrupt routine. For example, the interrupt at the requested CPU level could be disabled when the interrupt request occurs. Lower level interrupt requests are disabled when a higher priority interrupt routine is executing. The lower priority interrupt must wait to be recognized until interrupts are enabled at its priority level. An interrupt request can also be delayed if the interrupt levels are deliberately disabled by a CPU program. The CPU program may disable all interrupts (except level 7 which is not maskable in M68000 family processors) to protect a critical region of code as discussed later. Another possibility is that a higher priority exception such as a bus error is being processed when the interrupt request occurs. Section 11.8 explains exception priorities. An interrupt is simply one of the many possible exceptions recognized by the MC68332 CPU.

Example 15.5

Figure 15.6 summarizes the timing involved for an interrupt request and execution of the interrupt service routine. The worst-case total response time as seen by an external device is defined as $T_{(\text{RESPONSE})}$. This response time is the important parameter to use in determining the overall timing for the system or product. The response time is the sum of the delay time before interrupt processing begins and the total time required for the interrupt handling routine. This worst case estimate of response time is used because the action that actually services the interrupt request may occur anywhere in the interrupt routine, including during the execution of the last instruction.

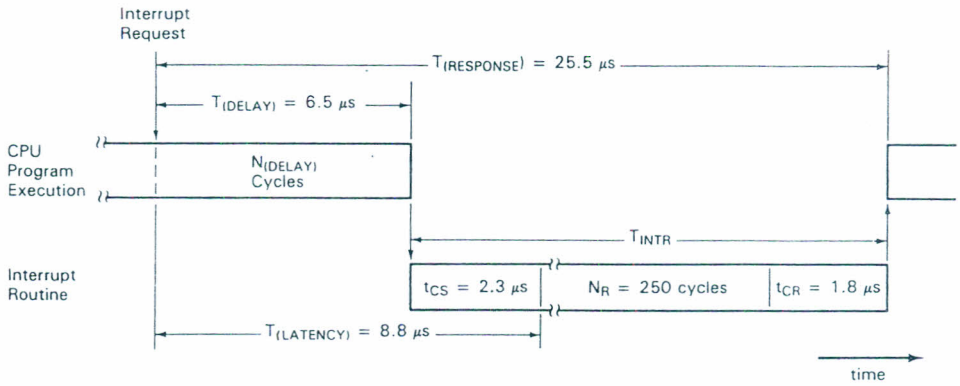

Figure 15.6 Interrupt timing.

In Figure 15.6, the latency time $T_{(\text{LATENCY})}$ discussed in Example 15.4 includes the delay time $T_{(\text{DELAY})}$ until the interrupt is recognized and the context switching time t_{CS}. If no higher priority interrupt routines are executing, the delay is the time required for the current instruction to complete execution as described in Example 15.4.

When interrupts are disabled at the requested level by a higher priority interrupt routine or for any other reason, $T_{(\text{DELAY})}$ represents the time between the interrupt request and the time that interrupts are reenabled at the requested level. The total time from the interrupt request until control is returned to the interrupted program is

$$T_{(\text{RESPONSE})} = T_{(\text{DELAY})} + T_{\text{INTR}}. \tag{15.3}$$

This time includes the time to restore the context, t_{CR}.

As an example, assume the following values have been determined:

$$T_{(DELAY)} = 6.5\mu s$$
$$t_{CS} = 2.3\mu s$$
$$t_{CR} = 1.8\mu s.$$

If the interrupt routine executes 50 instructions that average 5 clock cycles each, the total number of cycles is $N_R = 250$. The clock frequency, $f_{(system)}$ is assumed to be 16.78MHz.

The latency until the interrupt routine begins execution is

$$T_{(LATENCY)} = T_{(DELAY)} \ (6.5\mu s) + t_{CS} \ (2.3\mu s)$$

or $8.8\mu s$ in this example.

The response time is therefore

$$\begin{aligned} T_{(RESPONSE)} &= T_{(DELAY)} + t_{CS} + N_R \times t_{CYC} + t_{CR} \\ &= 6.5\mu s + 2.3\mu s + 250 \times 59.5ns + 1.8\mu s \\ &= 25.5\mu s \end{aligned}$$

using where t_{CYC} is the period of the 16.78MHz system clock.

Considering the overall system timing, a CPU program would be interrupted for T_{INTR} seconds or $19.\mu s$ in this example. The action to be taken by the interrupt routine to respond to the interrupt request would not occur until after the interrupt routine begins executing at least $t_{(LATENCY)}$ seconds after the request. A worse case estimate of the response time would be $t_{(LATENCY)}$ plus the total time of the interrupt routine or $25.5\mu s$ in our example.

Interrupt frequency and density. During normal operation of a product, some interrupt sources may interrupt randomly in time and other sources may request interrupts periodically. For example, an interrupt request from the TPU in response to an input capture might occur at a random time. On the other hand, the SCI may periodically interrupt when data are being transferred as discussed in Subsection 12.3.2. For interrupt-controlled input at 9600 Baud, for example, the CPU must respond to each SCI interrupt and read a character from the SCI receiver register approximately every millisecond.

The product designer must assure that every interrupt that can occur within a given time interval is serviced and that no interrupt service is delayed beyond the maximum allowable response time for the interrupt. Thus, during the specification of interrupt timing, the following conditions must be considered:

(a) the worst case interrupt density;

(b) the worst case delay time for any interrupt.

The *interrupt density* determines the percentage of CPU time taken up by interrupt service routines. The value is the ratio of the sum of the interrupt routine times divided by an arbitrary time period associated with the product's operation. For example, if a control system performs data acquisition, processing and control actions periodically, the cycle time, $T_{(CYCLE)}$, for the complete sequence could be taken as the

periodic reference time for the product. If the i^{th} interrupt routine requires $T_{\text{INTR}i}$ seconds to switch context and complete execution, the interrupt density is calculated as

$$\frac{T_{\text{INTR1}} + T_{\text{INTR2}} + \cdots + T_{\text{INTR}n}}{T_{(\text{CYCLE})}} < 1.0. \tag{15.4}$$

When the inequality of Equation 15.4 is satisfied, a product designer can be sure that the interrupt routines do not consume all of the CPU execution time during $T_{(\text{CYCLE})}$. The time left in the cycle for normal CPU processing is obviously $T_{(\text{CYCLE})}$ minus the sum of the times taken by the interrupt routines.

Example 15.6

Another calculation based on Equation 15.4 will determine whether the periodic interrupt with the minimum time period can be serviced. Assume that a low priority interrupt at level 3 interrupts every 500 μs. The time $T_{(\text{CYCLE})}$ in Equation 15.4 becomes 500μs. Further assume that level 4 and level 5 interrupts occur during the time $T_{(\text{CYCLE})}$ and that the total times taken by the routines are as follows:

$$T_{\text{INTR3}} = 20\mu\text{s}$$
$$T_{\text{INTR4}} = 20\mu\text{s}$$
$$T_{\text{INTR5}} = 20\mu\text{s}.$$

According to Equation 15.4, the interrupt density is

$$\frac{20\mu\text{s} + 20\mu\text{s} + 20\mu\text{s}}{500\mu\text{s}} = .12$$

or 12%. Thus, 86% of the time between the fastest interrupt requests is available for CPU execution of other programs.

Once assured that all interrupts can be serviced and the CPU can execute its main program, the designer must calculate the maximum delay time associated with any interrupt. If an interrupt at level i, $(1 \leq i \leq 7)$, must be serviced within a time T_{Pi}, the maximum delay time for this interrupt must satisfy the equality

$$T_{(\text{DELAY})} < T_{Pi} - T_{\text{INTR}i} \tag{15.5}$$

when the term $T_{\text{INTR}i}$ represents the total time to service the interrupt. As in Example 15.4, $T_{\text{INTR}i}$ includes the context switch time, the execution time for the interrupt routine and the time to restore context and return to the interrupted program. As expected, the equation states that the maximum allowable delay time is equal to the minimum time between interrupt requests for this interrupt source minus the time for the interrupt routine. Even if the interrupt routines are very short and the interrupt density is low, a low-priority interrupt that interrupts frequently might not be serviced in time if the delay is excessive.