10. Host Interface Bus

A host processor can communicate with the TMS34010 by means of an interface bus consisting of a 16-bit data path and several transfer-control signals. The TMS34010's host interface provides a host with access to four programmable 16-bit registers (resident on the TMS34010), which are mapped into four locations in the host processor's memory or I/O address space. Through this interface, commands, status information, and data are transferred between the TMS34010 and host processor.

A host processor may read from or write to TMS34010 local memory indirectly via an autoincrementing address register and data port. This optional autoincrement feature supports efficient block moves. The TMS34010 and host can send interrupt requests to each other. A pin is dedicated to the interrupt request from the TMS34010 to the host. To allow block moves initiated by a host to take place more efficiently, the host may suspend TMS34010 program execution to eliminate contention with the TMS34010 for local memory. DRAM-refresh and screen-refresh cycles continue to occur while the TMS34010 is halted.

This section includes the following topics:

10.1 Host Interface Bus Pins

The GSP's host interface bus consists of a 16-bit bidirectional data bus and nine control lines. These signals are described in detail in Section 2.

↔ **HD0–HD15**
form a 16-bit bidirectional bus, used to transfer data between the GSP and a host processor.

→ **$\overline{\text{HCS}}$** is the host chip select signal. It is driven active low to allow a host processor to access one of the host interface registers.

→ **HFS0, HFS1**
are function select pins. They specify which of four host interface registers a host will access (see Section 10.2).

→ **$\overline{\text{HREAD}}$** is driven active low to allow a host processor to read the contents of the selected host interface register, output on HD0–HD15.

→ **$\overline{\text{HWRITE}}$** is driven active low to allow a host processor to write the contents of HD0–HD15 to the selected host interface register.

→ **$\overline{\text{HLDS}}$** is driven low to enable a host processor to access the lower byte of the selected host interface register.

→ **$\overline{\text{HUDS}}$** is driven low to enable a host processor to access the upper byte of the selected host interface register.

← **HRDY** informs a host processor when the GSP is ready to complete an access cycle initiated by the host.

← **$\overline{\text{HINT}}$** transmits interrupt requests from the GSP to a host processor.

10.2 Host Interface Registers

The host interface registers are a subset of the I/O registers discussed in Section 6. The host interface registers can be accessed by both the GSP and the host processor. These registers occupy four 16-bit locations in the host processor's memory or I/O address map. One of these four locations is selected by placing a particular code on the two function select inputs, HFS0 and HFS1, as shown in Table 10-1. A 16-bit host processor will typically connect two of its low-order address lines to HFS0 and HFS1. An 8-bit processor typically connects two low-order address lines to HFS0–HFS1 and uses a third low-order address bit to enable either the upper or lower byte of the selected register by activating one of the byte select inputs, $\overline{\text{HUDS}}$ or $\overline{\text{HLDS}}$. In the second case, the registers occupy eight 8-bit locations in the host processor's memory map.

Table 10-1. Host Interface Register Selection

PORT	HFS1	HFS0	Selected Register	INTERNAL
	0	0	HSTADRL	16 BIT
	0	1	HSTADRH	16 BIT
	1	0	HSTDATA	
16 BIT	1	1	HSTCTL	2·16 BIT

HSTADRL and **HSTADRH** contain the 16 LSBs and 16 MSBs, respectively, of a 32-bit pointer address. A host processor uses this address to indirectly access GSP local memory.

The **HSTDATA** register buffers data that is transferred through the host interface between GSP local memory and a host processor. HSTDATA contains the contents of the address pointed to by the HSTADRL and HSTADRH registers.

The **HSTCTL** register is accessible to the GSP as two separate I/O registers, HSTCTLL and HSTCTLH, but is accessed by a host processor as a single 16-bit register. HSTCTL contains several programmable fields that control host interface functions.

NMI. Nonmaskable interrupt, bit 8. Allows a host processor to interrupt GSP execution.

NMIM. NMI mode, bit 9. Specifies if the context of an interrupted program is saved when a nonmaskable interrupt occurs.

CF. Cache flush, bit 14. Setting this bit flushes the contents of the GSP instruction cache. A host processor can force the GSP to execute new code after a download by flushing old instructions out of cache.

LBL. Lower byte last, bit 13. Specifies which byte of a register an 8-bit host processor will access first.

INCR. Increment address before local read, bit 12. Controls whether the 32-bit pointer in the HSTADR registers will be incremented before being used in a local read cycle that updates the HSTDATA register.

INCW. Increment address after local write, bit 11. Controls whether the 32-bit pointer in the HSTADR registers will be incremented after being used in a local write cycle that transfers the contents of the HSTDATA register to memory.

HLT. Halt GSP program execution, bit 15. A host processor can halt the TMS34010's on-chip processor by setting this bit to 1.

MSGIN. Message in, bits 0–2. Buffers a 3-bit interrupt message from a host processor to the GSP.

INTIN. Input interrupt bit, bit 3. A host must load a 1 into this bit to generate an interrupt request to the GSP.

MSGOUT. Message out, bits 4–6. Buffers a 3-bit interrupt message from the GSP to a host.

INTOUT. Interrupt out, bit 7. The GSP must load a 1 to this bit to send an interrupt request to a host processor.

10.3 Host Register Reads and Writes

Host interface read and write cycles are initiated by the host processor and are controlled by means of the *$\overline{\text{HCS}}$, $\overline{\text{HWRITE}}$, $\overline{\text{HREAD}}$, $\overline{\text{HUDS}}$*, and *$\overline{\text{HLDS}}$* signals. Host-initiated accesses of the register selected by the function-select code input on HFS0 and HFS1 are controlled as follows:

- While $\overline{\text{HCS}}$, $\overline{\text{HLDS}}$, and $\overline{\text{HWRITE}}$ are active low, the contents of HD0-HD7 are latched into the lower byte of the selected register.
- While $\overline{\text{HCS}}$, $\overline{\text{HUDS}}$, and $\overline{\text{HWRITE}}$ are active low, the contents of HD8-HD15 are latched into the upper byte of the selected register.
- While $\overline{\text{HCS}}$, $\overline{\text{HLDS}}$, and $\overline{\text{HREAD}}$ are active low, the contents of the lower byte of the selected register are driven onto HD0-HD7.
- While $\overline{\text{HCS}}$, $\overline{\text{HUDS}}$, and $\overline{\text{HREAD}}$ are active low, the contents of the upper byte of the selected register are driven onto HD8-HD15.

As this list indicates, at least three control signals *must* be active at the same time to initiate an access. The last of the three signals to become active begins the access, and the first of the three signals to become inactive signals the end of the access. A signal that begins or completes an access is referred to in the following discussion as the *strobe* signal for the cycle. Any of the signals listed above may be a strobe. Figure 10-1 shows a functional representation of the logic that controls the GSP's host interface.

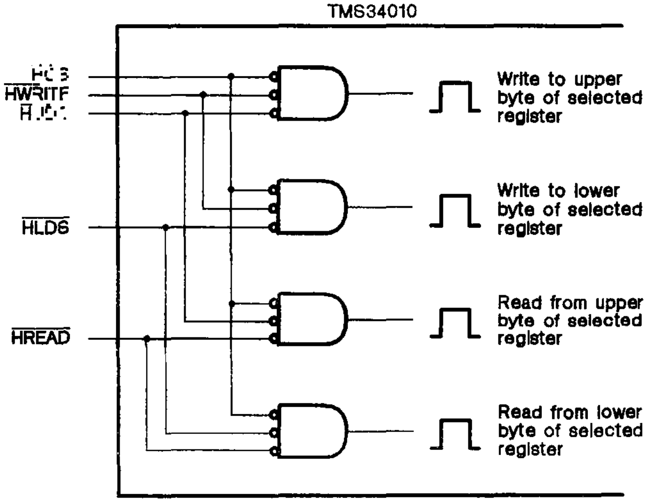
TMS34010
HCS
HWRITE
HUDS
HLDS
HREAD
Write to upper byte of selected register
Write to lower byte of selected register
Read from upper byte of selected register
Read from lower byte of selected register

Figure 10-1. Equivalent Circuit of Host Interface Control Signals

The designer must ensure that $\overline{\text{HREAD}}$ and $\overline{\text{HWRITE}}$ are never active low simultaneously during an access of a host interface register; this may cause internal damage to the device.

10.3.1 Functional Timing Examples

The functional timing examples in this section are based on the circuit shown in Figure 10-1.

- The $\overline{\text{HCS}}$ input is the strobe in Figure 10-2 and Figure 10-3.
- The $\overline{\text{HWRITE}}$ signal is the strobe in Figure 10-4.
- The $\overline{\text{HREAD}}$ signal is the strobe in Figure 10-5.
- The $\overline{\text{HUDS}}$ and $\overline{\text{HLDS}}$ signals are strobes in Figure 10-6 and Figure 10-7.

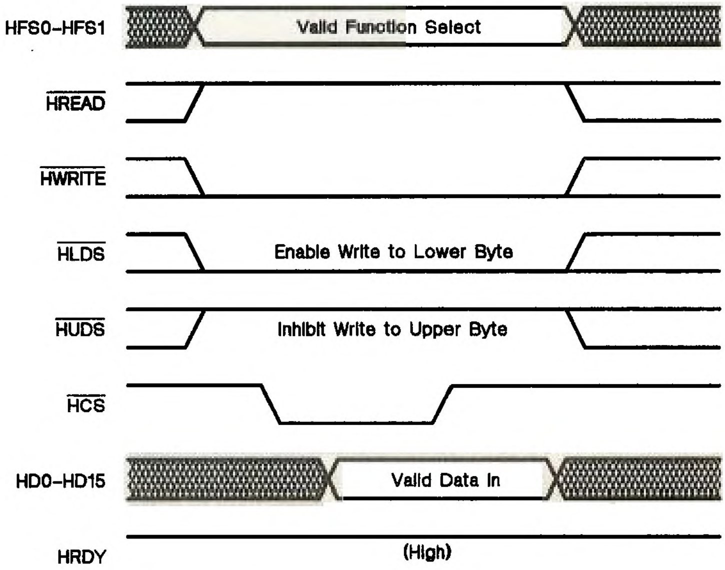

Figure 10-2. Host 8-Bit Write with $\overline{\text{HCS}}$ Used as Strobe

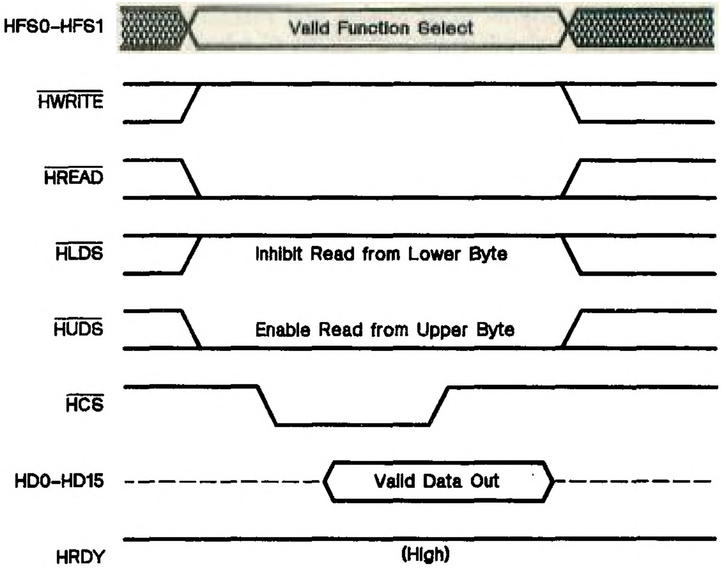

Figure 10-3. Host 8-Bit Read with $\overline{\text{HCS}}$ Used as Strobe

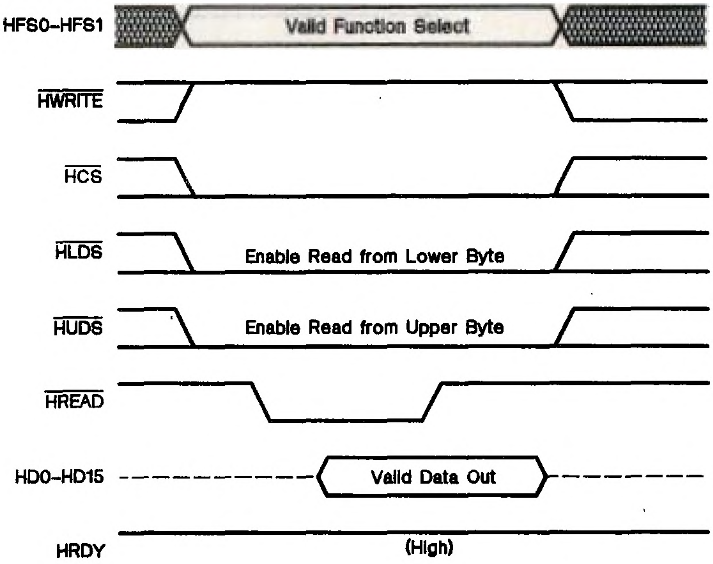

Figure 10-4. Host 16-Bit Read with $\overline{\text{HREAD}}$ Used as Strobe

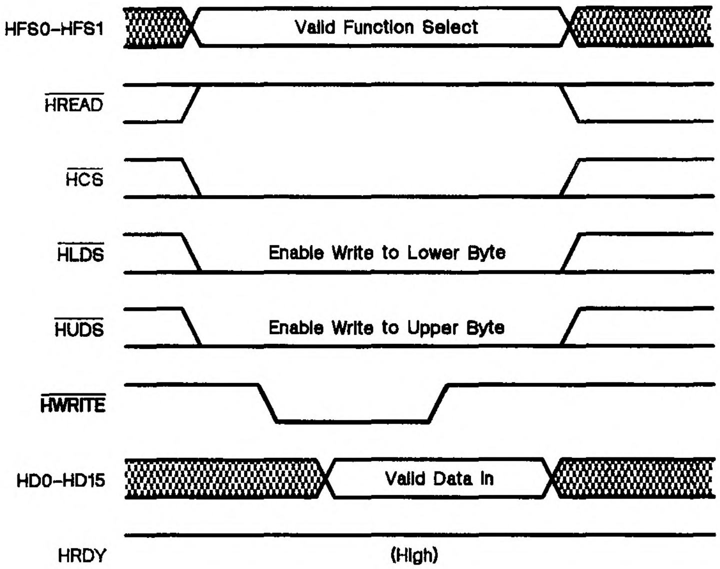

Figure 10-5. Host 16-Bit Write with $\overline{\text{HWRITE}}$ Used as Strobe

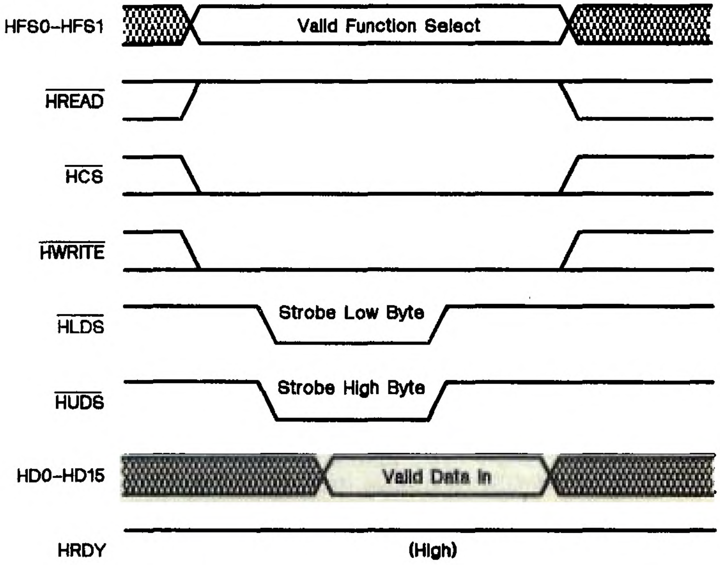

Figure 10-6. Host 16-Bit Write with $\overline{\text{HLDS}}$, $\overline{\text{HUDS}}$ Used as Strobes

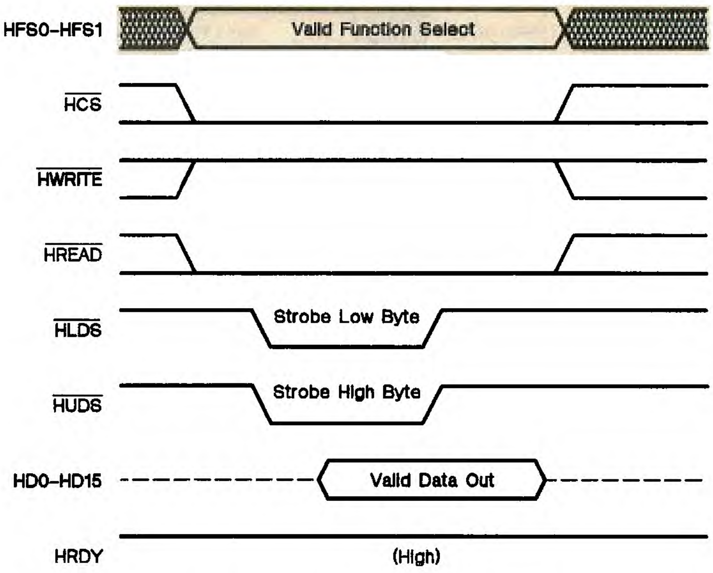

Figure 10-7. Host 16-Bit Read with $\overline{\text{HLDS}}$, $\overline{\text{HUDS}}$ Used as Strobes

10.3.2 Ready Signal to Host

The default state of the bus ready output pin, HRDY, is active high. HRDY is driven inactive low to force the host processor to wait in circumstances in which the GSP is not prepared to allow a host-initiated register access to be completed immediately.

HRDY is always driven low for a brief period at the beginning of a read or write access of the HSTCTL register. When the host attempts to read from or write to the HSTCTL register, HRDY is driven low at the beginning of the access, and is driven high again after a brief interval of one to two local clock cycles.

When the host processor performs certain types of host interface register accesses, a local memory cycle results. For example, in reading from or writing to the HSTDATA register, a read or write cycle on the local bus will result. If the host processor attempts to perform an access that will initiate a second local memory cycle before the GSP has had sufficient time to complete the first, the GSP will drive its HRDY output low to indicate that the host must wait before completing the access. When the GSP has completed the local memory cycle resulting from the previous access, it drives HRDY high to indicate that the host processor can now complete its second access.

A data transfer through the host interface takes place only when some combination of HCS, HREAD, HWRITE, HUDS, and HLDS are active simultaneously; however, the HRDY signal is activated by the HCS input alone. HRDY can be active-low only while the GSP is chip-selected by the host processor, that *is*, only when HCS is active low. A high-to-low transition on HRDY follows a high-to-low transition on HCS. The benefit of this mode of operation is that HRDY becomes valid as soon as HCS goes low, which typically is early in the cycle. HRDY is always driven high when HCS is inactive high.

A transient low level on the HCS input may cause a corresponding low pulse on the HRDY output. Systems that cannot tolerate such transient signals must be designed to prevent HCS from going low except during a valid host interface access.

In summary, the following rules govern the HRDY output:

1) If a high-to-low HCS transition occurs while the GSP is still completing a local memory cycle resulting from a previous host-indirect access, HRDY will go low. If the register selected is HSTDATA, HSTADRL or HSTADRH, HRDY will remain low until the local memory cycle is completed. If the register selected is HSTCTL, the HRDY output will remain low for one to two local clock periods.

2) If the host is given a ready signal (HRDY high) to allow it to complete a register access that will cause a local memory read or write cycle, HRDY stays high to the end of the access. The access ends when the *strobe* for the cycle ends. The strobe ends when HREAD and HWRITE are both inactive high, or when HLDS and HUDS are both inactive high, or when HCS is inactive high, whichever is the first to occur. As soon as the strobe ends, a low level on HCS will allow HRDY to go low again. If the strobe is an input other than HCS, and HCS remains low after the strobe ends, HRDY can go low as a delay from the end of the strobe. If HCS is the strobe for the access, the access ends when HCS goes high, and HRDY can go low again as soon as HCS goes low again.

3) If HSTCTL is selected (FS0 = FS1 = 1) at the high-to-low transition of HCS, HRDY will go low as a delay from the fall of HCS, and will remain low for one to two local clock periods. To avoid a low-going pulse on HRDY when accessing a register other than HSTCTL, FSO-FS1 should be valid prior to the high-to-low transition of HCS.

Figure 10-8 and Figure 10-9 (page 10-10) show examples of host interface register accesses in which HRDY is driven low.

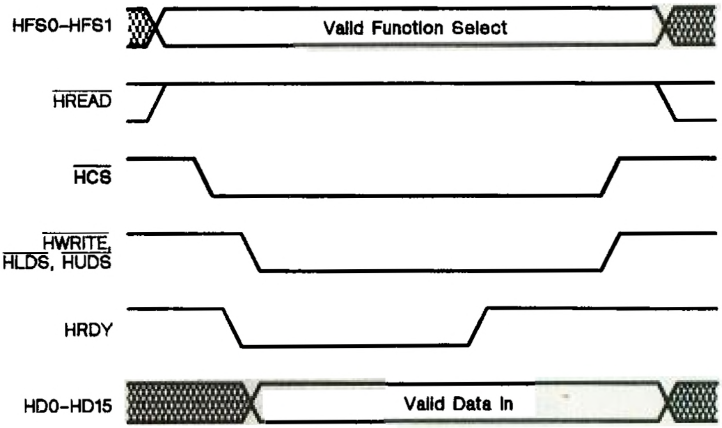

Figure 10-8. Host Interface Timing - Write Cycle With Wait

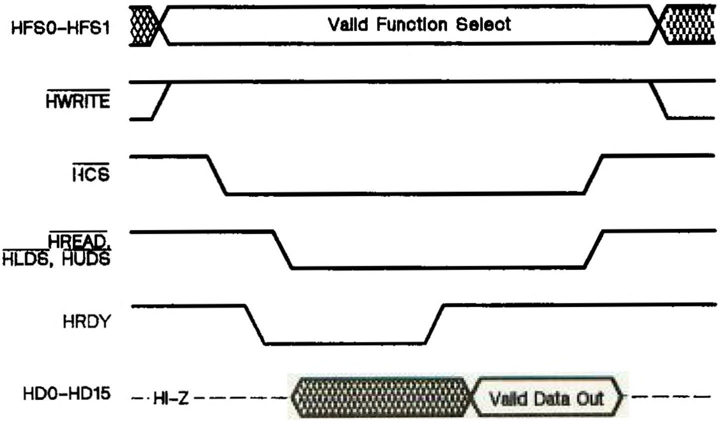

Figure 10-9. Host Interface Timing - Read Cycle With Wait

10.3.3 Indirect Accesses of Local Memory

The host processor indirectly accesses GSP local memory by reading from or writing to the HSTDATA register. HSTDATA buffers data written to or read from the local memory. The word in local memory that is accessed is the word pointed to by the 32-bit address contained in the HSTADRL and HSTADRH registers. The pointer address is loaded into HSTADRL and HSTADRH by the host processor before performing one or more indirect accesses of local memory using the HSTDATA register.

The four LSBs of HSTADRL are forced to 0s internally so that the address formed by HSTADRL and HSTADRH always points to a word boundary in local memory. Between successive indirect accesses of local memory using the HSTDATA register, the local memory address contained in the HSTADR registers can be autoincremented by 16. This allows the host processor to access a block of sequential words in local memory without the overhead of loading a new address prior to each access.

During a sequence of one or more indirect reads of local memory by the host, the GSP maintains in HSTDATA a copy of the local memory word currently addressed by the HSTADRL and HSTADRH registers. Reading from HSTDATA returns the word prefetched from the local memory location pointed to by the HSTADRL and HSTADRH registers, and causes HSTDATA to be updated from local memory again. Writing to HSTDATA causes the word written to HSTDATA to subsequently be written to the location in local memory pointed to by the HSTADRL and HSTADRH registers.

Two increment-control bits, INCR and INCW (contained in the HSTCTL register), are set to 1 to cause the pointer address in HSTADRL and HSTADRH to be incremented by 16 during reads and writes, respectively. In preparing to use the autoincrement feature, the appropriate increment-control bit, INCR or INCW, is loaded with a 1, and the HSTADRL and HSTADRH registers are set up to point to the first location of a buffer region in the local memory.

- When **INCR** is set to 1, a read of HSTDATA causes the address in HSTADRL and HSTADRH to be incremented *before* being used in the local memory read cycle that updates HSTDATA.

- When **INCW** is set to 1, a write to HSTDATA causes the address in HSTADRL and HSTADRH to be incremented *after* being used in the local memory read cycle that writes the new contents of HSTDATA to local memory.

Loading the pointer address automatically triggers an update of HSTDATA to the contents of the local memory word pointed to. No increment of HSTADRL and HSTADRH takes place at this time regardless of the state of the increment bits. Each subsequent host access of HSTDATA causes HSTADRL and HSTADRH to be automatically incremented (assuming INCR or INCW is set) to point to the next word location in the local memory. In this manner, a series of contiguous words in local memory can be accessed following a single load of the HSTADRL and HSTADRH registers without additional pointer-management overhead.

10.3.3.1 Indirectly Reading from a Buffer

Figure 10-10 illustrates the procedure for reading a block of words beginning at local memory address *N*. Assume that the INCR bit in the HSTCTL register is set to 1 and the LBL bit in HSTCTL is set to 0.

- In Figure 10-10 *a*, the host processor loads the 32-bit address *N* into HSTADRL and HSTADRH.
- The loading of the second half of the address into HSTADRH causes the GSP host interface control logic to automatically initiate a read cycle on the local bus. This read cycle, shown in Figure 10-10 *b*, transfers the contents of memory address *N* to the HSTDATA register.
- In *c*, the host processor reads the HSTDATA register, fetching the data previously read from address *N*.
- The read of HSTDATA by the host processor causes the GSP to automatically increment the contents of HSTADRL and HSTADRH by 16, as shown in *d*.
- The contents of the new address are read into HSTDATA, as shown in Figure 10-10 *e*. This data will be available in HSTDATA the next time it is read by the host processor.

The process shown in *c* through *e* repeats for every word read from GSP local memory.

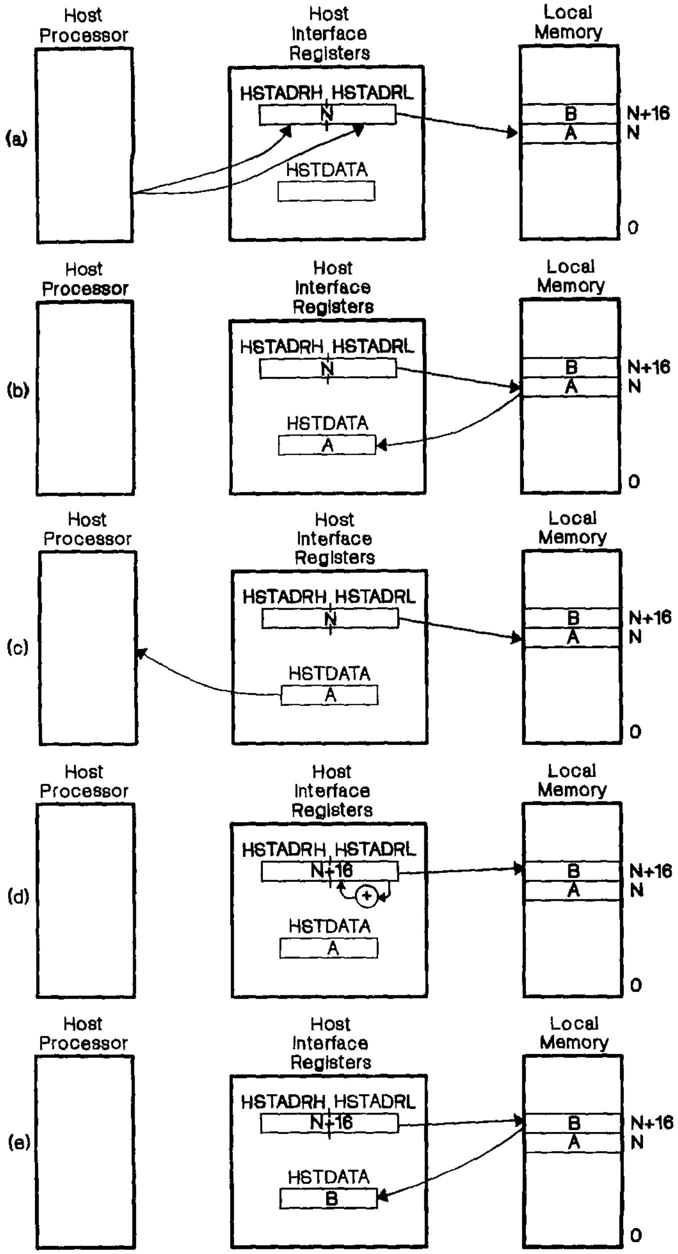

Figure 10-10. Host Indirect Read from Local Memory (INCR=1)

10.3.3.2 Indirectly Writing to a Buffer

Figure 10-11 illustrates the procedure for writing a block of words to GSP local memory. The block begins at address *N*. Assume that the INCW bit is set to 1 and the LBL bit is set to 0.

- In Figure 10-11 *a*, the host processor loads the 32-bit address *N* into HSTADRL and HSTADRH.
- The loading of the second half of the address into HSTADRH causes the GSP host interface control logic to automatically initiate a read cycle on the local bus. This read cycle, which takes place in Figure 10-11 *b*, fetches the contents of memory address *N* into HSTDATA.
- The data loaded into this register will not be used, however. Instead, the host processor writes to the HSTDATA register in Figure 10-11 *c*, overwriting its previous contents.
- In response to the host's write to HSTDATA, the GSP automatically initiates a write cycle to transfer the contents of HSTDATA to the local memory address *N* as shown in *d*.
- Following the write, the GSP automatically increments the address in HSTADRL and HSTADRH to point to the next word, as shown in *e*. At this point the host interface registers are ready for the host processor to write the next word to HSTDATA.

The process shown in *c* through *e* repeats for every word written to GSP local memory.

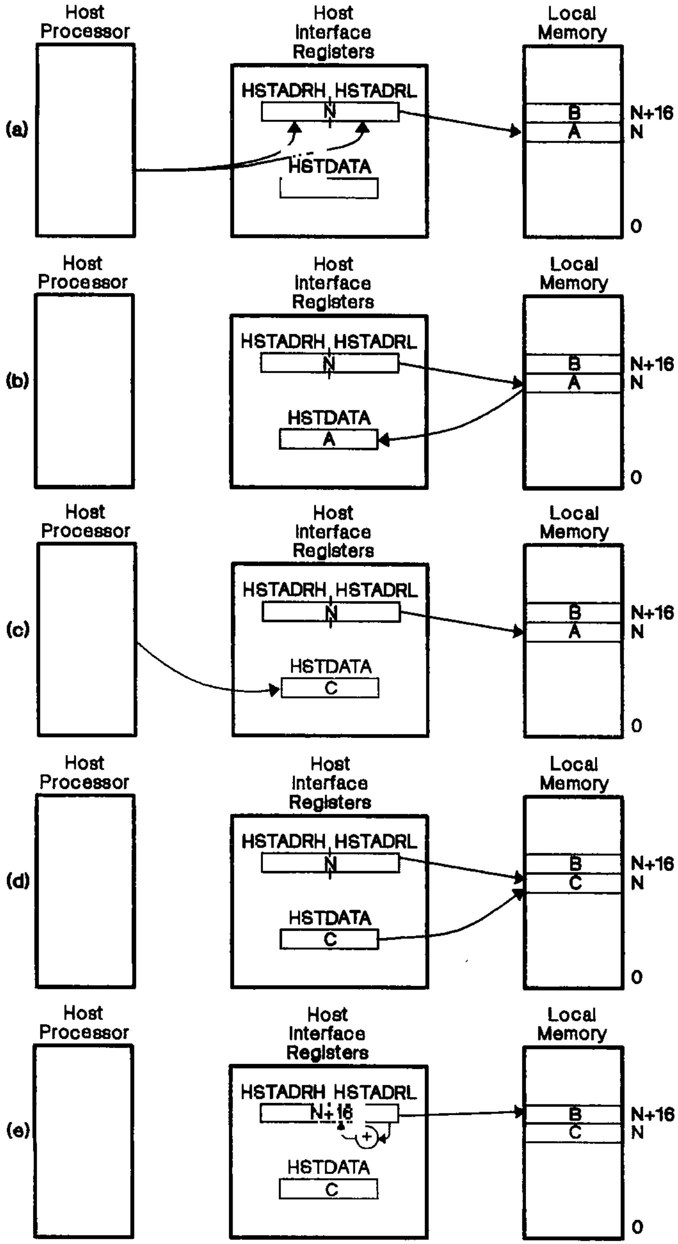

Figure 10-11. Host Indirect Write to Local Memory (INCW=1)

10.3.3.3 Combining Indirect Reads and Writes

If the HSTDATA register in Figure 10-11 is read by the host processor following step *e*, the value returned will be the value that the host previously loaded into the register. The host must read HSTDATA a second time to access data from GSP local memory. This principle is illustrated in Figure 10-12, which shows how the host interface performs when a write is followed by two reads. For this example, INCW=1 and INCR=0.

- In Figure 10-12 *a*, HSTADRL and HSTADRH together point to location *N* in the GSP's local memory. The host processor is shown writing to HSTDATA.
- In *b*, the data buffered in HSTDATA is written to location *N* in memory.
- The address registers are incremented in *c*.
- In *d*, the host processor reads the HSTDATA register, which returns the value that the host loaded into the register in step *a*.
- Reading HSTDATA causes a memory read cycle to take place in *e*, which loads the value from memory address *N*+16 into HSTDATA.
- In *f*, a second read of HSTDATA by the host processor returns the value from memory address *N*+16.

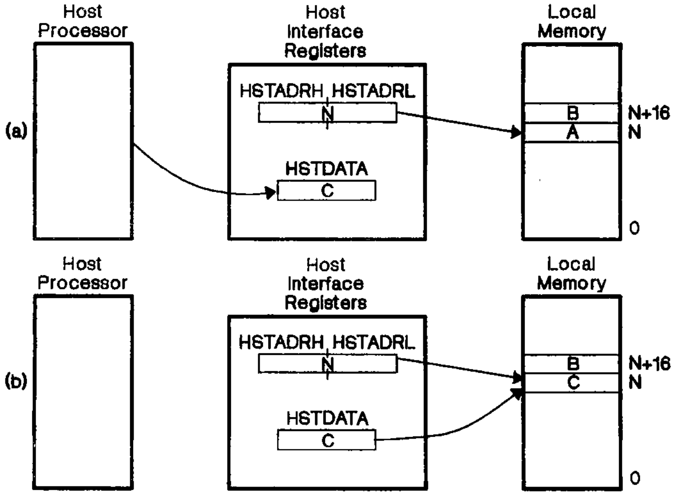

Figure 10-12. Indirect Write Followed by Two Indirect Reads (INCW=1, INCR=0)

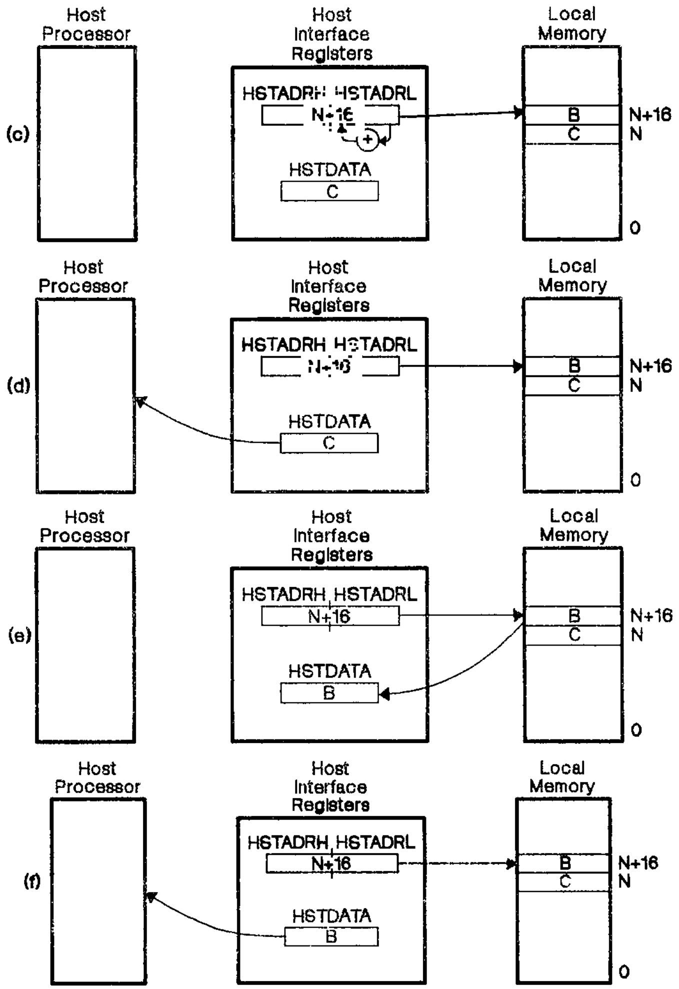

Figure 10-12. Indirect Write Followed by Two Indirect Reads (INCW=1, INCR=0) (Concluded)

10.3.3.4 Accessing Host Data and Address Registers

When the TMS34010 internal processor accesses the HSTDATA, HSTADRL, or HSTADRH register, no subsequent cycle occurs to transfer data between HSTDATA and local memory. Also, the address in HSTADRL and HSTADRH is not incremented, regardless of the state of the INCR and INCW bits.

The host processor can indirectly access any register in the GSP's internal I/O register file by first loading HSTADRL and HSTADRH with the address of the register, and they writing to or reading from HSTDATA.

No hardware mechanism is provided to prevent simultaneous accesses of the HSTDATA, HSTADRL and HSTADRH registers by the host processor and by the GSP internal processor. Software must be written to avoid simultaneous accesses, which can result in invalid data being read from or written to these registers.

10.3.3.5 Downloading New Code

The TMS34010 host interface provides a means of efficiently downloading new code from a host processor to GSP local memory. The host initiates this operation through the following process:

- Before downloading, the host interrupts and halts the GSP by writing 1s to the HLT and NMI bits in the HSTCTL register. The host processor should then wait for a period of time equal to the TMS34010 interrupt latency. (GSP hardware will reset the NMI bit if the nonmaskable interrupt is initiated before the halt occurs.)
- The code is then downloaded using the auto-increment features of the host interface registers.
- After downloading the code, the host should flush the cache as described in Section 5.4.5, Flushing the Cache (page 5-26).
- The nonmaskable interrupt vector is written through the host port to location >FFFF FEE0 so that the new code will begin execution at the vectored address.
- The NMI bit in the HSTCTL register should be set to 1 to initiate a nonmaskable interrupt. At the same time, the NMIM bit in the HSTCTL register should be set to 1. If the host does not need the current context to be stored on the stack, or if the nonmaskable interrupt was taken in the first step, the NMIM bit should be set to 1. Otherwise, NMIM should be set to 0.
- The host restarts the GSP by writing a 0 to the HLT bit in the HSTCTL register.

Setting the HLT and NMI bits to 1 simultaneously reduces the worst-case delay (compared to setting HLT only). NMI latency is the delay from the 0-to-1 transition of the NMI bit and the start of execution of the first instruction of the NMI service routine. Halt latency is the delay from the 0-to-1 transition of the HLT bit and the time at which the GSP actually halts (see Section 10.3.4). The maximum NMI latency may be much less than the halt latency if a PIXBLT, FILL, or LINE instruction is in progress at the time of the NMI or halt request. An NMI request will interrupt instruction execution at the next interruptible point, but a halt request is ignored until the executing instruction completes or is interrupted. When NMI and HLT are set to 1 *simultaneously,* the GSP will have halted before beginning execution of the first instruction in the NMI service routine. Therefore, the delay from the setting the NMI and HLT bits to the time that the GSP actually halts is simply the NMI latency.

10.3.4 Halt Latency

The TMS34010 may be halted by a host processor via the HLT bit in the HSTCTL register. The delay from the receipt of a halt request to the time that the TMS34010 actually halts is the sum of five potential sources of delay:

1) Halt request recognition
2) Screen-refresh cycle
3) DRAM-refresh cycle
4) Host-indirect cycle
5) Instruction completion

In the best case, items 2 through 5 cause no delay. The minimum delay to due item 1 is one machine state.

- The **halt request recognition** delay is the time required for the setting of the $\overline{\text{HLT}}$ bit to be internally synchronized after the low-to-high transition of the HRDY pin.

- The **screen-refresh** and **DRAM-refresh cycles** are a potential source of delay, but in fact occur rarely and are unlikely to delay a halt.

- The likelihood of a delay caused by a **host-indirect cycle** is small in most instances, but this depends largely on the application. It would only occur if the host had written to the data register just prior to writing to the HLT bit. The delay due to a single host-indirect cycle is two machine states, assuming no wait states.

- The instruction completion time refers to the time required for an instruction that was already executing at the time the halt request was received to complete. Note that the TMS34010 halt condition is entered only on **instruction** boundaries. This means that a PIXBLT, FILL, or LINE instruction that is already in progress will run to completion before the GSP halts.

Table 10-2 shows the minimum and maximum times for each of the five operations listed. The halt latency is calculated as the sum of the numbers in the five rows. In the best case, the halt latency is only one machine state. The worst-case latency is six machine states plus the delays due to host-indirect cycles and instruction completion. Table 10-3 shows instruction completion times for some of the longer instructions. However, a PIXBLT, FILL, or LINE instruction may take longer than the times shown in Table 10-3, depending on the size of the pixel array or line specified. Table 10-3 also shows the instruction completion time for a JRUC instruction that jumps to itself – the GSP may be executing this instruction if the software is simply waiting for a halt.

Table 10-2. Five Sources of Halt Delay

Operation	Latency (In States)	
	Min	Max
Halt recognition	1	2
Instruction completion	0	See Table 10-3
DRAM-refresh cycle	0	2 See Note 2
Screen-refresh cycle	0	2 See Note 2
Host-indirect cycle	0	See Note 1

Notes: 1) The latency due to host-indirect cycles depends on both the hardware system and the application. The delay due to a single host-indirect cycle is two machine states, assuming no wait states.

2) DRAM-refresh and screen-refresh cycle times assume no wait states.

Table 10-3. Sample Instruction Completion Times

Instruction	Worst-Case Instruction Completion Time (In States)	
	SP Aligned	SP Not Aligned
DIVS A0,A2	43	43
MMFM SP,ALL	72	144
MMTM SP,ALL	73	169
PIXBLT, FILL, and LINE	See Note 1	See Note 1
Wait: JRUC wait	1	1

Notes: 1) The worst-case instruction completion time is equal to the instruction execution time less one machine state.

2) The SP-aligned case assumes that the SP is aligned to a word boundary in memory.

10.3.5 Accommodating Host Byte-Addressing Conventions

Processor architectures differ in the manner in which they assign addresses *to* bytes. The GSP host interface logic can be programmed to accommodate the particular byte-addressing conventions used by a host processor.

This ability is important in ensuring software compatibility between 8- and 16-bit versions of the same processor, such as the 8088 and 8086 or the 68008 and 68000. The 8088 transfers a 16-bit word as a series of two 8-bit bytes, low byte first, high byte second. The 68008 transfers the high byte first, and low byte second.

The HSTCTL register's LBL bit is used to configure the GSP host interface to accommodate different byte-accessing methods. The host interface is configured to operate according to the following two principles:

1) First, when a host processor with an 8-bit data bus reads from or writes to the HSTDATA register, it will access the high and low bytes of the register in separate cycles. The GSP will not initiate its local memory access until both bytes of HSTDATA have been accessed.

2) Second, when HSTADRH and HSTADRL are loaded by the host, the GSP must not initiate its read of the local memory until the complete pointer address has been loaded into HSTADRL and HSTADRH.

When LBL=0:

- A local memory read cycle takes place when the host processor reads the high byte of HSTDATA, or writes to the high byte of HSTADRH.
- A local memory write cycle takes place when the host processor writes to the high byte of HSTDATA.

When LBL=1:

- A local memory read cycle takes place when the host processor reads the low byte of HSTDATA, or writes to the low byte of HSTADRL.
- A local memory write cycle takes place when the host processor writes to the low byte of HSTDATA.

When the host processor is an 8088, for example, the GSP is typically configured by setting the LBL bit of the HSTCTL register to 0. When configured in this manner, the GSP expects the HSTADRL register to be loaded first, and HSTADRH loaded second. Furthermore, the high byte of the HSTADRH register is expected to be loaded after the low byte. When LBL is set to 0, a local read cycle is initiated when the upper byte of the HSTADRH register is written to by the host processor. This permits the lower byte of HSTADRH to be loaded first without causing side effects.

10.4 Bandwidth

One measure of the performance of the host interface is its data rate, or bandwidth. The bandwidth is the number of bits per second that can be transferred through the host interface during a block transfer of data to or from GSP memory. Assume that the host interface address register is programmed to autoincrement. The maximum data rate through the host interface can be expected to approach the bandwidth of the GSP's memory. For example, assume a 50-MHz GSP and a memory requiring no wait states. The memory cycle time is about 320 nanoseconds (bandwidth = 50 megabits/second). The host's access cycle time at the host interface is somewhat longer than this due to certain additional delays inherent in the operation of the GSP's internal host interface logic. Also, the throughput of the host interface may depend on whether or not the GSP is halted.

The bandwidth is calculated as the width of the host data path (16 bits) times the frequency of access cycles through the host interface. Given a continuous series of word accesses, with successive accesses occurring at regular intervals, what is the minimum interval between host accesses that the interface can sustain without having to send not-ready signals to the host? (The GSP drives its HRDY output low temporarily to inform the host when the GSP is not yet ready to complete the host's current access.)

First, when the GSP is halted, the host interface should support continuous accesses occurring at regular intervals no less than about 400 nanoseconds apart. As long as the host attempts to maintain a throughput no greater than this limit, delays due to not-ready signals will occur rarely, if at all. The bandwidth for this case is calculated in Table 10-4 *a* as approximately 40 megabits per second. This value can be expected to vary slightly with system-dependent conditions such as the frequency of DRAM-refresh and screen-refresh cycles.

When the GSP is running, the host interface should support continuous accesses occurring at regular intervals no less than approximately 550 nanoseconds. The bandwidth for this case is calculated in Table 10-4 as approximately 29 megabits per second. This value varies slightly with conditions such as the frequency of DRAM-refresh and screen-refresh cycles, and also with the characteristics of the program being executed by the GSP.

Table 10-4. Host Interface Estimated Bandwidth

Assumptions	Approximate Throughput
GSP halted 50-MHz GSP No wait states	$\frac{\text{16 bits/transfer}}{\text{400 ns/transfer}}$ = 40 megabits/s
GSP running 50-MHz GSP No wait states	$\frac{\text{16 bits/transfer}}{\text{550 ns/transfer}}$ = 29 megabits/s

10.5 Worst-Case Delay

In some applications, designers must determine not only the effective throughput of the host interface, but also the delays that can occur under worst-case conditions. These conditions occur too rarely to affect overall throughput, but the important consideration here is not how often they occur, but that they can occur at all. First, with the GSP halted, the worst delay is given by the formula $(6 + 2N)T$, where N is the number of wait states per GSP memory cycle, and T is the local clock period (nominally 160 nanoseconds for a 50-MHz GSP). Second, with the GSP running, the worst delay is given by the formula $(9 + 4N)T$. The derivation of these formulas, summarized in Figure 10-13, may be helpful in illustrating the mechanisms of the host interface.

$2T$	Synchronization delay
$(2 + N)T$	Screen-refresh cycle
$+ (2 + N)T$	DRAM-refresh cycle
$(6 + 2N)T$	Worst-case delay (total)

(a) Worst-Case Delay with GSP Halted

$2T$	Synchronization delay
$(1 + N)T$	GSP CPU read
$(2 + N)T$	GSP CPU write
$(2 + N)T$	Screen-refresh cycle
$+ (2 + N)T$	DRAM-refresh cycle
$(9 + 4N)T$	Worst-case delay (total)

(b) Worst-Case Delay with GSP Running

N = Number of wait states per memory cycle
T = Local clock period (nominal 160 nanoseconds for 50-MHz device)

Note: These are worst-case delays and have negligible effect on performance. The case shown in *a*, for example, could be expected to occur less than once per thousand (0.1 percent of) host accesses in a typical system.

Figure 10-13. Calculation of Worst-Case Host Interface Delay

Consider case *a*, in which the GSP is halted, first; the worst-case delay is calculated as the sum of the three delays. The first of these delays is the time required to internally synchronize the host interface cycle to the GSP local clock. The host's signals are generally not synchronous to the GSP local clocks. A signal from the host must therefore be passed through a synchronizer latch (part of the GSP on-chip host interface logic) before being used by the GSP. The delay through the synchronizer is from one to two local clock periods ($1T$ to $2T$), depending on the phase of the host clock relative to the GSP's local clock. The second and third delays in Figure 10-13 represent the time needed to perform a screen-refresh cycle followed by a DRAM-refresh cycle. The arbitration logic internal to the GSP assigns these two types of cycles higher priorities than host-requested indirect accesses. (Screen refresh has a higher priority than DRAM refresh.) Thus, a host access requested at the same time as one of these cycles must wait. The worst-case assumption is that a screen-refresh cycle is generated internal to the GSP on the same clock edge at which the request for the host access arrives. Furthermore, a DRAM-refresh cycle is requested during this same clock edge or during the next 1 + *N* clock edges. An equivalent delay occurs in the case in which a DRAM refresh and host access are requested on the same clock edge (the DRAM refresh wins), and a screen refresh is requested on a later clock edge before the host access can begin. This case is not shown in Figure 10-13, but the delay in this instance is also (6 + 2*N*)*T*. In a typical system, DRAM-refresh cycles consume about 2 percent of the available memory bandwidth, and screen-refresh cycles take about 1.5 percent (using VRAMs). The probability of either sequence of events is therefore very small (less than one in a thousand, assuming *N* = 0; that is, no wait states), and the performance degradation due to these unlikely events is negligible.

Now consider the case in which the GSP is running. Host accesses are of higher priority than GSP instruction fetches and data accesses, but still of lower priority than DRAM-refresh or screen-refresh cycles. The worst-case delay is calculated as the sum of the five delays indicated in Figure 10-13 *b*. This assumes that the GSP begins a read-modify-write operation on a memory word (this is performed as a read cycle followed by a separate write cycle) just one clock before the GSP receives the host access request. The GSP CPU read cycle is actually (2 + *N*)*T* in duration, but since it begins one clock before the host access is requested, only (1 + *N*)*T* is left in the cycle. The GSP's local memory controller treats a read-modify-write operation as indivisible; once the read has started, no other request can be granted until the write completes. The write cycle is (2 + *N*)*T* in duration. Again, assume that sometime before the write cycle does complete, screen-refresh and DRAM-refresh cycles are also requested. The probability of this case is somewhat more difficult to calculate than that of Figure 10-13 *a*, since the frequency of read-modify-write operations is very program dependent. This sequence of events rarely occurs, however.