user or multitasking system. At the end of the interrupt service routine, the RTE instruction restores the status register (which includes the user/supervisor flag and the interrupt mask) to its state prior to the interrupt.

The 68008 8-bit bus version of the 68000 is available in two packages: a 48-pin DIP and a 52-pin chip carrier. Due to pin limitations, the DIP version has only two interrupt inputs: $\overline{\text{IPL1}}$ and IPL0/$\overline{2}$. $\overline{\text{IPL0}}$ and $\overline{\text{IPL2}}$ are connected together internally to produce the IPL0/$\overline{2}$ input. Thus, there are only four possible interrupt levels: 0 (no interrupt), 2, 5, and 7 (nonmaskable interrupt). As illustrated in the design example in Sec. 3.10, this is more than adequate for many applications.

3.7 BUS ARBITRATION

The microprocessor is normally the only device that drives the address and control buses. However, there are situations in which another device drives these buses, and the microprocessor's bus drivers must then be disabled. The most common example is direct memory access (DMA), in which a DMA controller takes over the buses to perform high-speed data transfers (typically between a peripheral and memory). Other examples are systems with coprocessors that need access to the buses, and multiple microprocessor systems in which more than one microprocessor share the same buses. Normally the microprocessor controls the buses and is thus the *bus master*. When another device such as a DMA controller takes control of the buses, it becomes the *temporary bus master*.

(DMA is described in detail in Chap. 5, and multiple processor systems are described in Chap. 10.)

3.7.1 Basic Bus Request/Bus Grant Logic

Most microprocessors include a *bus request* input to allow other devices to request that the microprocessor not use the buses. When this input is asserted, the microprocessor completes the instruction in progress and then stops instruction execution, disables its bus drivers, and asserts the *bus grant* output. Bus grant is monitored by the other device(s) capable of controlling the bus to determine when the microprocessor has released the buses. Bus grant is also used to disable external bus buffers, if present. The device that requested the buses then controls them for as long as is needed. It then negates bus request, and the microprocessor takes control of the buses and resumes normal operation.

Many microprocessors use this basic two-wire bus control scheme, although different names are used. The 8085 and 80186 call these signals HOLD and HLDA (hold acknowledge). The Z80 calls them $\overline{\text{BUSRQ}}$ (bus request) and $\overline{\text{BUSAK}}$ (bus acknowledge). The 6802 calls them $\overline{\text{HALT}}$ and BA (bus available). The 6802 does not place its address bus in a high-impedance state, however, so external three-state drivers are required. This is not unreasonable, since most systems using DMA or multiple processors are likely to need address bus buffers anyway to provide increased bus drive.

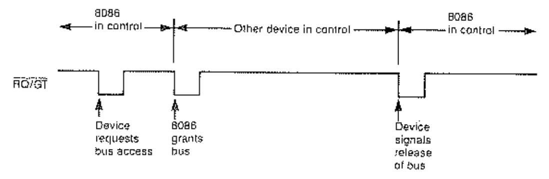
8086 in control
Other device in control
8086 in control
$\overline{\text{RQ}}/\overline{\text{GT}}$
Device requests bus access
8086 grants bus
Device signals release of bus

Figure 3.28. The 8086 bus request/bus grant operation

3.7.2 The 8086 Request/Grant Operation

The 8086 uses two different techniques to implement bus request/bus grant functions. In minimum mode, HOLD and HLDA signals as previously described are used. In maximum mode, a different approach is used. Rather than using one pin for request and another for grant, one pin serves both functions. There are two request/grant pins, $\overline{\text{RQ}}/\overline{\text{GT0}}$ and $\overline{\text{RQ}}/\overline{\text{GT1}}$, which are bidirectional. Each operates in the same manner, although $\overline{\text{RQ}}/\overline{\text{GT0}}$ has higher priority.

Control of the buses is requested by asserting one of the $\overline{\text{RQ}}/\overline{\text{GT}}$ inputs for one clock period, as shown in Fig. 3.28. Note that the line is not held asserted but only pulsed. Open-collector or three-state drivers are used, so that when the line is not asserted it floats to the inactive state. The 8086 then issues the bus grant by outputting a pulse on the same pin. To terminate the sequence, the temporary bus master pulses the same $\overline{\text{RQ}}/\overline{\text{GT}}$ pin again to indicate to the 8086 that the buses are now available.

While this approach complicates the logic required to perform bus arbitration, it reduces the number of lines required from two to one. This allows the 8086 to provide two separate request inputs in maximum mode and allows the 8087 math coprocessor to request and be granted use of the bus using only one pin.

For those applications that don't need this feature and don't want the complications, the 8086's minimum mode provides the simpler two-wire control approach. The two $\overline{\text{RQ}}/\overline{\text{GT}}$ pins are replaced with HOLD and HLDA pins, identical to those used by the 8085 and 80186.

3.7.3 The 68000 Bus Arbitration

The 68000 uses a three-wire approach to bus arbitration. Figure 3.29 shows the bus exchange operation. As usual, a bus request signal (called $\overline{\text{BR}}$) indicates to the microprocessor that another device wants access to the bus. The microprocessor responds by asserting $\overline{\text{BG}}$ (bus grant), but this bus grant functions differently than the signals previously discussed. $\overline{\text{BG}}$ is asserted as soon as the 68000 has internally synchronized the $\overline{\text{BR}}$ input. The buses are not actually available until the completion of the cycle, so the device requesting bus control must wait not only for $\overline{\text{BG}}$ to be asserted but also for the address and data strobe signals to be negated.

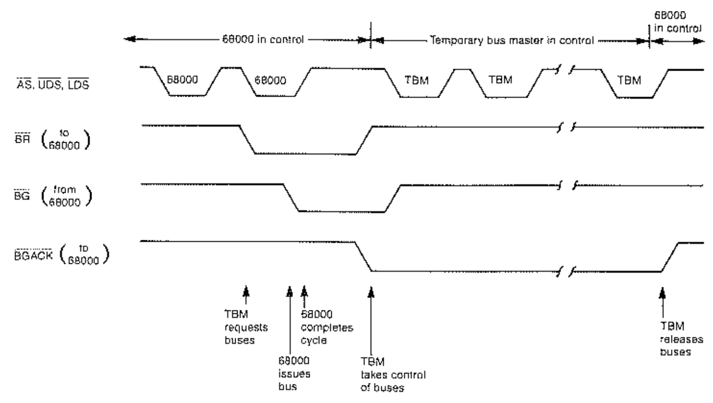
68000 in control
Temporary bus master in control
68000 in control
$\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$
$\overline{BR}$ (to 68000)
$\overline{BG}$ (from 68000)
$\overline{BGACK}$ (to 68000)
68000
68000
TBM
TBM
TBM
TBM requests buses
68000 completes cycle
68000 issues bus
TBM takes control of buses
TBM releases buses

Figure 3.29. The 68000 bus request/bus grant operation

The device requesting the buses responds to the bus grant by asserting $\overline{BGACK}$ (bus grant acknowledge) and negating $\overline{BR}$. The temporary bus master holds $\overline{BGACK}$ asserted for as long as it needs the bus and then negates it. The 68000 then resumes operation. This is in contrast to most other microprocessors, which have no bus grant acknowledge. In such systems, the temporary bus master simply keeps bus request asserted until it is done using the buses.

The reason for $\overline{BGACK}$ is to allow bus requests from multiple devices to be ORed into the $\overline{BR}$ input. Thus, when one or more of the devices are requesting the bus, $\overline{BR}$ is asserted. $\overline{BGACK}$ is needed so that the temporary bus master can indicate when it is done. If it simply held $\overline{BR}$ asserted while it was using the bus, and if another device was also requesting the bus, then the 68000 would see no change in $\overline{BR}$ when the first device was done, and the second device would not know that it could take over the bus.

When the temporary bus master is ready to release the bus, it negates $\overline{BGACK}$. If $\overline{BR}$ is still asserted, indicating that another device is also requesting control of the buses, the 68000 asserts $\overline{BG}$ again.

External priority logic is required if multiple bus requests are ORed into $\overline{BR}$. When the bus grant is issued, it must be accepted only by the highest priority device currently requesting the bus.

Due to pin limitations, the 48-pin DIP version of the 68008 does not include the $\overline{BGACK}$ signal. Instead, more conventional bus request and bus grant signals are used.

3.7.4 Bus Arbitration with Multiple Bus Requesters

The bus arbitration schemes discussed above are those provided directly by the microprocessors. If there is more than one potential bus requester, however, additional logic is required. If multiple bus requests are simply ORed together to provide the bus request to the processor, a problem can occur if two devices request the bus at the same time: both requesters will see the acknowledge, and both will attempt to take control of the bus. The result is a bus conflict, and neither requester will be able to effectively control the bus. The end result is generally a system "crash."

It is thus necessary that a priority be assigned to each potential bus master, so that when two or more requests are received simultaneously the higher-priority device is granted the bus before the lower-priority device(s). Two common approaches are *serial* and *parallel* arbitration.

Serial arbitration, as shown in Fig. 3.30, uses a daisy chain similar to the Z80's interrupt priority control discussed in Sec. 3.6.4. Each possible bus master has a bus priority in and a bus priority out pin. The highest-priority device is at the start of the chain and can always request the bus. Normally, each device copies the level on its bus priority input to its bus priority output. When a device wants to access the bus, it first checks the state of BUS ACK. If BUS ACK is asserted, indicating that the main processor has already granted the bus to a temporary bus master, then the bus requester must wait for this device to complete its bus activity and release the bus. The requester then negates its bus priority output and asserts BUS REQ. If more than one requester does this at the same time, the lower-priority device will know that it has been overridden because its bus priority in will be negated. The lower-priority device therefore ignores the BUS ACK, and the higher-priority device takes control of the buses.

There are several disadvantages of serial arbitration. It is slow if there are many potential bus masters, since the bus priority signal must pass through each of them

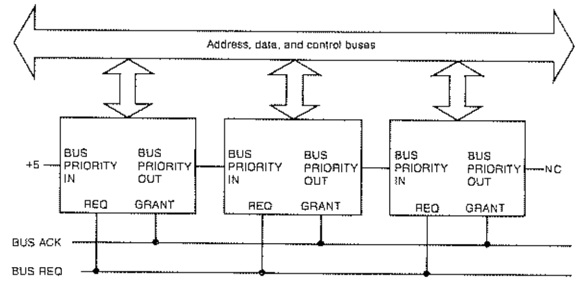

Figure 3.30. Serial bus arbitration using priority daisy chain

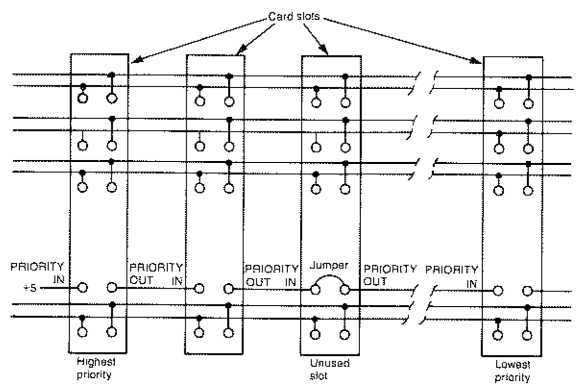

Figure 3.31. Daisy-chained priority signals on backplane.

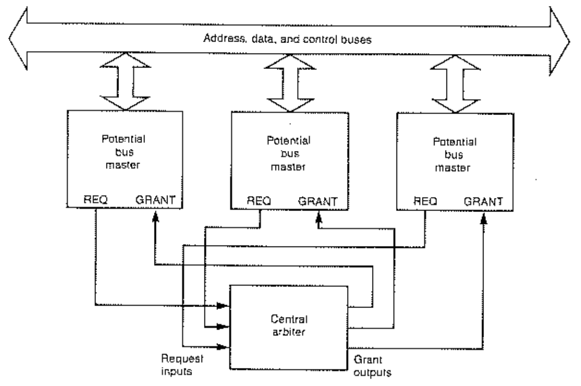

Figure 3.32. Parallel bus arbitration using central arbiter.

before it is valid at all points. The priority chain signal cannot be wired to all connectors on the backplane in parallel but must be connected as shown in Fig. 3.31. The priority of each board is thus determined by its physical position in the backplane and cannot be dynamically changed. Any open slots between occupied slots must have a jumper to provide continuity for the priority daisy chain signal. The primary advantage of the serial technique is that it is simple and inexpensive to implement since no central arbiter is needed.

Parallel arbitration, as shown in Fig. 3.32, uses a separate bus request and grant signal for each potential bus master. A central arbiter, which is typically part of the backplane, receives all bus requests, decides which is the highest priority, and grants the bus to that device. This technique is faster, since additional levels of logic are not needed for each possible bus master.

The most common central arbiter implementation assigns a fixed priority to each bus request input (in effect, each backplane connector). More sophisticated arbitration algorithms are also possible with a central arbiter. One example is *round-robin* priority, in which the priority of each device changes each time the bus is granted. Round-robin priority is less commonly used than fixed priority due to the additional logic required.

5.7 DIRECT MEMORY ACCESS CONTROLLERS

Direct memory access *(DMA)* controllers allow data to be transferred faster than is possible under program control. A common example of a DMA application is reading from or writing to a disk drive that must transfer data at a rate determined by the disk's speed of rotation and other characteristics. Another example is an arbitrary waveform generator that synthesizes analog waveforms by repetitively writing a series of data bytes to a D/A converter. This section describes the types of DMA transfers and several example DMA controller chips.

5.7.1 DMA Controller Operation

Consider the problem of writing a block of memory to an output port, one byte at a time. The following tasks must be performed:

1. Initialize memory and output port addresses.
2. Repeat until all bytes transferred:
   a. Read byte from memory.
   b. Write byte to output port.
   c. Increment memory address.
   d. Check to see if all bytes transferred.
   e. Wait until output port ready for next byte.

To implement this transfer with software, the microprocessor must fetch one or more instructions to perform each of these steps. Since several instructions must be fetched and executed to transfer a single byte of memory, only a fraction of the memory cycles are used for the actual data transfer. Thus, the speed of the data transfer is much less than the maximum rate at which data can be read from the memory.

A *DMA controller (DMAC)* can be thought of as a very specialized microprocessor, with the program to transfer data implemented in hardware. The DMAC includes counters to provide the memory and port addresses and to count the number of words transferred. Before a transfer can occur, the microprocessor must initialize the DMAC to specify the direction and type of transfer, the source and destination addresses, and the number of bytes or words to be transferred. Once this initialization is completed, the DMAC takes control of the system buses and performs the entire transfer.

Unlike a data transfer performed by the microprocessor, no instructions need be fetched during the transfer to tell the DMAC how to perform the transfer. Thus, all memory cycles are available for transferring data, and the transfer can occur at the maximum speed possible. The peripheral device generally operates at a slower data rate than this maximum, so the DMAC can allow the microprocessor to run for a few cycles in between transfers while the DMAC waits for the peripheral to be ready to transfer the next byte.

DMA controllers add hardware to the system and generally provide only one benefit: faster data transfers. They are therefore used primarily when the data rate possible under program control is insufficient. Even if the data rate is low enough so that the transfer can be performed under program control, the use of a DMA controller increases the amount of time available for other tasks by minimizing the time spent transferring data. This performance improvement can justify the use of a DMAC in some applications. The increase in performance must be weighed against the additional hardware cost of the DMAC and associated circuits.

After the DMA controller has been initialized by the microprocessor, the peripheral (such as a disk controller) can initiate the transfer at any time by asserting the DMA REQUEST input to the DMAC. The DMAC then asserts BUS REQUEST to the microprocessor (this signal is called HOLD on some microprocessors). The microprocessor completes the instruction it is currently executing, disables its address, data, and control bus outputs, and asserts the BUS ACKNOWLEDGE signal. The DMAC then takes control of the buses to perform the transfer. The DMAC controls the buses in the same manner as the microprocessor.

One benefit of DMA transfers is that the transfer can begin very quickly after the peripheral asserts the DMA REQUEST signal. The microprocessor needs only to complete the bus cycle then executing, and the transfer can begin. Since the microprocessor's operation is suspended during the transfer, there is no need for the registers to be saved, as in the case of an interrupt. The time from the request to the beginning of the transfer is called the *latency time*. The worst-case latency for a DMA transfer is the length of the longest bus cycle plus the relatively short time required for the microprocessor to disable its bus outputs and assert BUS ACKNOWLEDGE. One important exception to this is a system with multiple DMA channels, in which case the latency can be as long as the longest DMA transfer occurring on a higher-priority channel.

5.7.2 DMA Transfer Types

The simplest mode of DMA transfer is *burst* mode, in which the entire transfer is completed without interruption. If the peripheral cannot accept data at the maximum transfer rate, then the DMAC keeps the bus idle between transfers. In *cycle-stealing* mode, a single transfer is initiated for each DMA request from the peripheral. This is

useful if the data rate from the peripheral is slower than the maximum DMA transfer rate; it allows the microprocessor to execute instructions while waiting for the next byte from the peripheral.

Transparent DMA uses a dual-port RAM to allow both the DMAC and the microprocessor to access the memory concurrently. The microprocessor's bus request input is not used, since it never has to release its buses. This provides the highest performance but its additional cost is generally not justified.

Two-Cycle DMA Transfers

There are two basic types of DMA transfers: one-cycle and two-cycle. Figure 5.17 illustrates a memory-to-output-port transfer using the two-cycle approach. The DMA controller begins the transfer by reading the first memory location to be transferred and storing the data byte in a temporary register in the DMAC. It performs the read in the same manner as a microprocessor; it places the memory address on the address bus, asserts the MEMRD (memory read) control signal, and reads the data from the data bus. When the DMAC has completed the read cycle, it drives the data back onto the data bus, addresses the output port, and asserts the IOWR (I/O write) control signal. Thus, one word of data is read from the memory and written to the output port. The memory address is then incremented, and the process is repeated to transfer the next

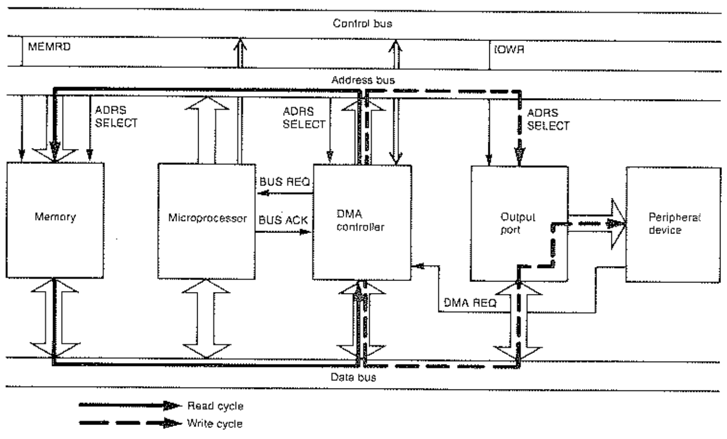

Figure 5.17. Two-cycle DMA transfer from memory to output port. The solid paths indicate the read cycle, the dashed paths indicate the write cycle.

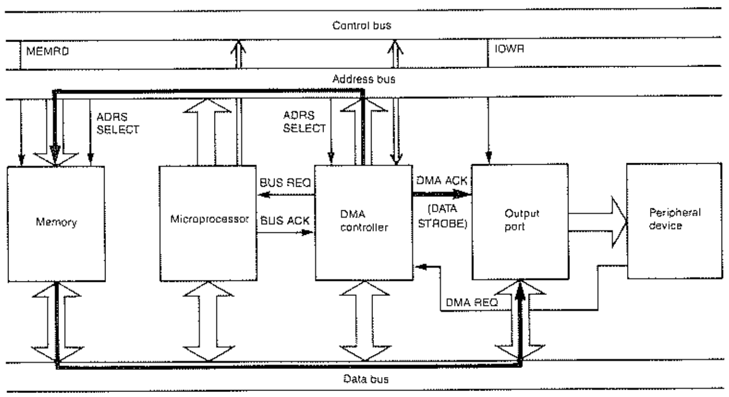

Figure 5.18. One-cycle DMA transfer from memory to output port.

word. When the specified number of words has been transferred (or after each word in cycle-stealing mode), the DMAC negates the BUS REQUEST signal to the microprocessor and the microprocessor continues operation from the point at which it was halted.

With the two-cycle approach just described, memory-to-memory transfers can also be performed. In this case, the microprocessor initiates the transfer via a software command to the DMAC, allowing the DMAC to be used to speed up programs that require moving blocks of data. Some DMA controllers also have the capability to search for a particular data pattern while transferring or to only search and not transfer at all. This feature can be used to improve the performance of programs that search large blocks of data.

One-Cycle DMA Transfers

While the two-cycle transfer provides maximum flexibility, the transfer rate is only one-half the highest possible rate, since two bus cycles are required for each transfer. To implement one-cycle transfers, a DMA ACKNOWLEDGE signal is added, as shown in Fig. 5.18. This signal replaces the address select signal for the I/O port and allows the DMAC to select an I/O port while simultaneously addressing memory.

To transfer from memory to an output port, the DMAC places the memory address on the address bus and asserts the DMA ACKNOWLEDGE signal to select the port. The DMAC then asserts both the MEMRD and IOWR control signals. The memory

provides the data on the data bus, which is read directly by the output port. The data does not pass through the DMA controller.

The main benefit of this technique is the high transfer speed. However, it is much more limited than the two-cycle approach. Since only one device can be addressed at a time via the address bus, a DMA ACKNOWLEDGE signal must be connected to each I/O port that can be used in a DMA transfer. The port addresses cannot be changed by the system software, as with the two-cycle approach. In addition, memory-to-memory transfers cannot be performed.

5.7.3 Am9517A DMA Controller

AMD's Am9517A is an example of a DMA controller that can perform either one- or two-cycle transfers. (This device is also made by Intel as the 8237A.) It was originally designed for use with the 8080 and 8085, although it can also be used with other microprocessors. It includes four independent DMA channels. Figure 5.19 shows the block diagram of the 9517.

The 9517 is a complex device, with numerous registers and operating modes. This section provides an overview of the device's operation. For detailed information, refer to the manufacturer's data sheets.

Figure 5.20 shows the 9517 interfaced to a microprocessor system. The four bus control lines are inputs to the 9517 when the microprocessor is initializing it and outputs from the 9517 when a DMA transfer is taking place. Similarly, address pins A_0 to A_3 are inputs to the 9517 for selecting registers during initialization and are outputs from the 9517 for addressing memory during DMA transfers. Address lines A_0 to A_7

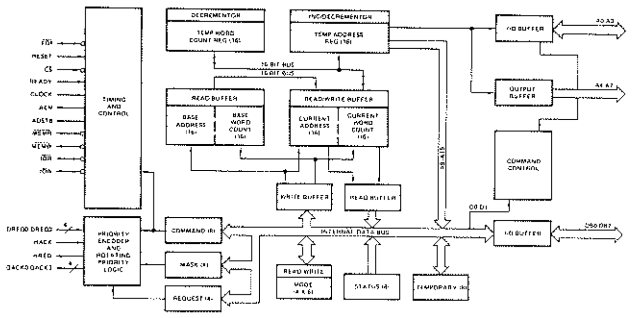

Figure 5.19. Am9517A DMA controller block diagram. *(Courtesy of Advanced Micro Devices, Inc.)*

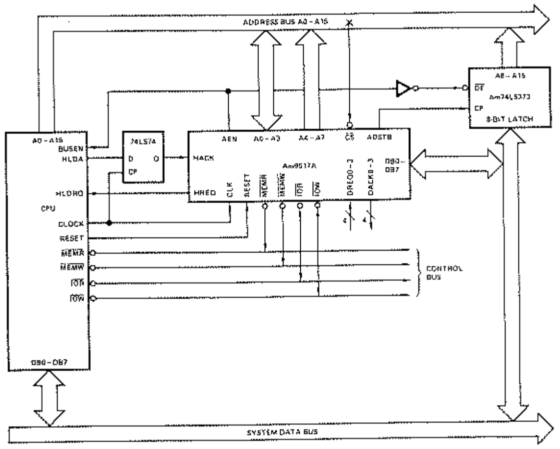

Figure 5.20.
Am9517A DMA controller interface (*Courtesy of Advanced Micro Devices, Inc.*)

are driven directly by the 9517 during DMA transfers. Because of pin limitations, the upper 8 bits of the address bus are not driven directly by the 9517. These address bits are provided by an external 8-bit latch, which is kept updated by the 9517. Whenever the upper 8 bits of address change during a DMA transfer, the 9517 outputs the new address on the data bus (which is used as a multiplexed bus during DMA transfers) and pulses the address strobe (ADSTB) signal to clock the address into the external latch. Note that this is similar to the manner in which some microprocessors multiplex address and data information, except that the 9517 multiplexes the most-significant address bits, rather than the least-significant bits, with the data. This is done because DMA transfers generally consist of many accesses to successive locations. Thus, the high-order half of the address does not change very often: on the average, once every 256 transfers. Unlike multiplexed-bus microprocessors that perform an address cycle at the beginning of every bus cycle, the 9517 updates the address latch only when necessary. This increases the transfer rate by eliminating the address latch update on most transfer cycles.

Each of the four channels in the 9517 has its own DMA request and acknowledge signals and its own set of control registers. Each channel can perform an independent memory-to-I/O or I/O-to-memory transfer.

Each channel is assigned a priority. If multiple DMA requests occur at the same time, or if one channel's DMA request is asserted while another channel is performing a transfer, the highest-priority channel is serviced first. Thus, the highest-priority

channel has a very short worst-case latency time, while the others can have much longer latency if a higher-priority channel is performing a lengthy transfer.

Each channel has a 6-bit Mode register, and four 16-bit registers:

Base Address register
Base Word Count register
Current Address register
Current Word Count register

Because the address and word count registers are 16 bits wide but the data bus is only 8 bits wide, each register must be written in two parts. The first time a byte is written to a particular register address, it is written to the low-order half of the register, and an internal flip-flop (indicating that the low half has been written) is set. The next byte written to that register address is written to the high half, and the internal flip-flop is cleared.

The Mode register allows various types of transfers to be selected, such as read or write and incrementing address or decrementing address. The Base Address register stores the address of the first memory location to be read or written, and the Base Word Count register indicates the number of bytes to be transferred. When a transfer begins, the Current Address and Current Word Count registers are loaded from the base registers. As the transfer proceeds, the Current Word Count register is decremented, and the Current Address register is incremented or decremented, depending on the mode selected. The transfer is terminated when the current word count reaches zero or when the external end of process ($\overline{\text{EOP}}$) signal is asserted. The base registers are not modified during the transfer. If "autoinitialize" mode is selected (by writing the appropriate value to the Mode register), then the Current Address and Current Word Count registers are automatically loaded from the base registers when the transfer is terminated. This allows another transfer to occur without the microprocessor reinitializing the 9517.

When performing single-cycle transfers, the Current Address register addresses the memory location to be read or written, and the I/O port to be read or written is selected by the DMA acknowledge (DACK) signal for that channel. Thus, while the memory address can be programmed when the microprocessor initializes the 9517, the I/O port to be used in the transfer is determined by the hardware connections.

The 9517 also allows two channels to be used together to perform two-cycle transfers. A read cycle is performed using channel 0, and the data read is stored in a temporary register in the 9517. A write cycle is then performed using channel 1, and the data from the temporary register is driven on the data bus. This allows memory-to-memory transfers to be performed, and also allows DMA transfers to or from memory-mapped I/O ports.

The 9517 does not decode the WAIT or READY control signal, so it will not work properly with very slow memory. The memory must be fast enough to meet the 9517's timing, since the 9517 will not insert wait states as the microprocessor would. For fast memory, there is a "compressed" transfer mode that increases the transfer rate.

Although it was designed for 8-bit microprocessors with 16-bit address buses, the 9517 can be adapted for microprocessors with a wider address bus, such as the 8088. (In fact, it is used in the IBM PC, XT, and AT.) Since the 9517 produces only 16-bit addresses, a "DMA page" register must be added to provide the most-significant address bits during DMA transfers. This register must be set by the microprocessor before the transfer is initiated. Thus, transfers are limited to locations within the 64-Kbyte page selected by this register.

5.7.4 Other DMA Controllers

Most microprocessor families include at least one DMA controller. Most use the two-cycle transfer method. The Z80 is supported by the Z80-DMA (also known as the MK3883 or Z8410). Motorola's 6844 DMAC is designed for use with the 6800- and 6809-family microprocessors and can also be used in a limited manner with the 68000. In addition to the 8237A, which is the same as the Am9517A, Intel manufactures a simpler DMAC, the 8257. This device operates only in the one-cycle transfer mode.

Most DMA controllers for 16- and 32-bit microprocessors are considerably more complex than those for 8-bit microprocessors. In addition to wider address registers (typically 24 bits) and data paths (16 or 32 bits), they include a variety of additional features. Typical features include the ability to read a 16-bit word from a memory location and then perform two byte writes to a peripheral (or vice versa), and automatic "chaining" of a series of data transfers that can be performed without processor intervention. AMD's Am9516 is a "universal" DMAC designed for use with the 8086- or 68000-family microprocessors. Intel's 82258 is a high-performance DMAC that is designed for use with the 80286 but can also be used with the 8086/88 or 80186/188. Three DMA controllers are available for use with the 68000: the 68430 single-channel DMA controller, the 68440 two-channel controller, and the 68450 four-channel controller.

Intel's 8089 I/O processor combines the functions of a DMA controller and a specialized microprocessor and is described in Chap. 10.

5.8 DESIGN EXAMPLE: 68901 MFP

In this section, we add the 68901 multifunction peripheral (MFP) to the example system. This chip provides four counter/timers, an interrupt controller, a serial communications interface, and an 8-bit parallel I/O port. The I/O capabilities are covered in later chapters; this section covers the bus interface and the counter/timers.

Figure 5.21 shows the bus interface to the 68901 MFP. The five register select lines enable direct selection of any of the 32 internal registers, making programming simpler than with peripherals that require multibyte sequences to access numerous internal registers with only one or two address lines. The CLK input is used to synchronize the bus interface signals and must be between 1 and 4 MHz. The 10-MHz system clock is divided by 4 to produce a 2.5-MHz clock. The MFP provides an open-drain $\overline{\text{DTACK}}$ output, so it is easily interfaced to a 68000-family system operating at any clock rate. The assertion of the $\overline{\text{CS}}$ signal is delayed by the flip-flop circuit until the first clock edge after $\overline{\text{AS}}$ is asserted, for reasons described in the following subsection.

The MFP includes an interrupt vector register, which is initialized by the system software with the desired base interrupt vector type number. The least-significant four

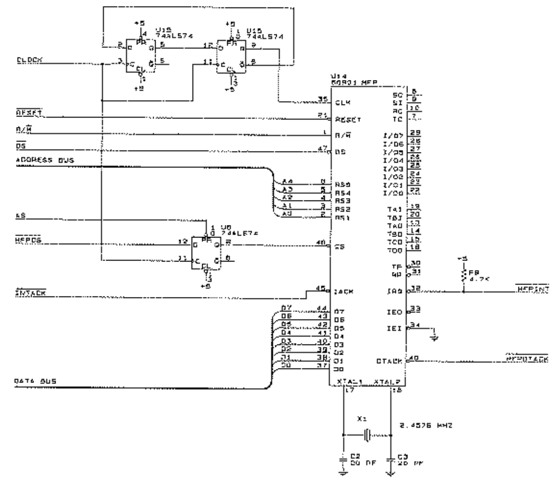

Figure 5.21. Design example 68901 interface.

bits of this vector are modified by the MFP according to the source of the interrupt. Possible interrupt sources include any of the eight parallel I/O lines, any of the four timers, and the serial communications interface. Each of these sources can be individually enabled or disabled, and the parallel I/O lines can be programmed to produce an interrupt in response to either a rising or a falling edge.

The MFP includes an $\overline{\text{IEI}}$ input and an $\overline{\text{IEO}}$ output that operate similarly to the Z80-family peripherals' pins of the same name, as described in Chap. 3. These signals can be connected in a daisy-chained manner when more that one MFP is used in a system, allowing the $\overline{\text{IRQ}}$ output of all the MFPs to connect to the same interrupt level. A pull-up resistor is required, since the $\overline{\text{IRQ}}$ output is open-drain. The interrupt enable daisy chain establishes priority among the MFPs.

When an enabled interrupt condition occurs and the MFP's $\overline{\text{IEI}}$ input is asserted, the $\overline{\text{IEO}}$ output is negated and the $\overline{\text{IRQ}}$ output is asserted. The microprocessor responds with an interrupt acknowledge cycle, which is decoded by the logic described in Chap. 3 to produce the $\overline{\text{IACK}}$ signal. This causes the MFP to place the interrupt vector on the data bus, and the microprocessor then executes the selected interrupt service routine.

The clock from the timers is derived from the signal on XTAL1 rather than from the CLK input. This allows the timer clock to be independent of the system clock. For the simplest configuration, XTAL1 can be connected to the CLK pin. Alternatively, a separate crystal oscillator can be connected to XTAL1 and XTAL2, as shown in Fig. 5.21.

The seemingly odd clock frequency of 2.4576 MHz is 300 × 2^{n} Hz, which divides down exactly to produce standard serial data rate signals as described in Chap. 8.

Each of the four timers is based on an 8-bit down counter. In addition, there is a separate prescaler for each timer. Timers A and B are multimode counter/timers and can operate in an event-count mode, in which transitions of the signal at the TAI or TBI input are counted, or in a pulse-width measurement mode. In addition, all four timers can operate in timer mode. In this mode, the counter is clocked by the prescaled XTAL1 signal. The prescaler can be programmed to divide by 4, 10, 16, 50, 64, 100, or 200. Each timer channel has a time constant register, which must be loaded to start the timer. This value is loaded into the down counter, which is decremented until it reaches 1. An interrupt is then generated (if enabled), the timer output is toggled, and the down counter is reloaded from the time constant register. Thus, this mode can be used to generate periodic interrupts or to produce a square-wave output of a programmable frequency. One of the timers is typically used to provide the data rate clock for the serial communications interface.

Figures 5.22 and 5.23 show the read and write timing for the MFP, and Table 5.1 lists the timing parameter values. Because the MFP does not assert its $\overline{\text{DTACK}}$ output until data is available for a read cycle, or until the data setup time has been met for a write cycle, the required access time and read and write pulse widths are guaranteed to be met, regardless of the speed of the microprocessor. $\overline{\text{DTACK}}$ is asserted time 5 after the second falling CLK edge following the assertion of $\overline{\text{CS}}$.

While the MFP is in most respects a well-behaved 68000-family peripheral, there is one potential timing problem. The MFP specifies a minimum register select (address) setup time of 30 ns (parameter 2). Since the $\overline{\text{CS}}$ output from the 74ALS138 is asserted as soon as the decoder's propagation delay time elapses, this setup time will not be met

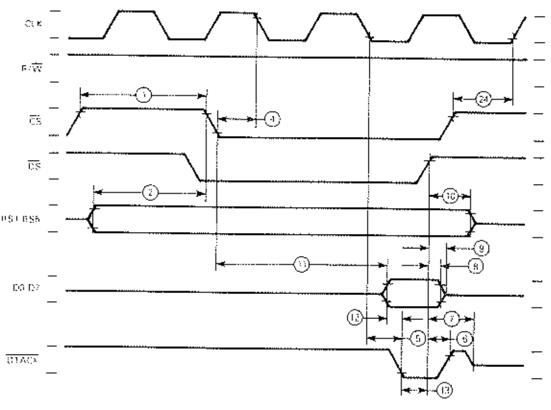

Figure 5.22. The 68901 read cycle timing. (Courtesy of Motorola, Inc.)

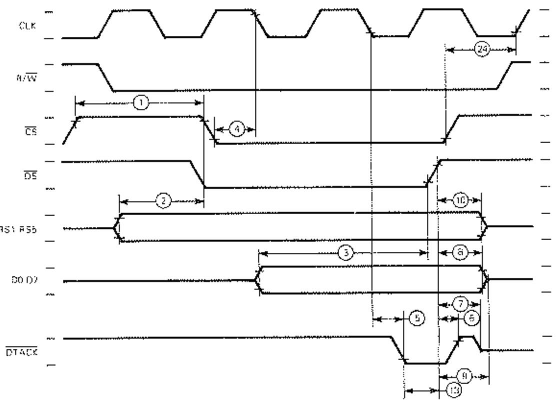

Figure 5.23. The 68901 write cycle timing. (Courtesy of Motorola, Inc.)

Table 5.1 The 68901 Read and Write Cycle Timing Parameters

No.	Characteristic	Time, ns Min	Time, ns Max
1	$\overline{\text{CS}}$, $\overline{\text{IACK}}$, $\overline{\text{DS}}$ width high	50	
2	Address valid to falling $\overline{\text{CS}}$ setup time	30	
3	Data valid prior to rising $\overline{\text{DS}}$ setup time	280	
4*	$\overline{\text{CS}}$, $\overline{\text{IACK}}$ valid to falling clock setup time	50	
5	Clock low to $\overline{\text{DTACK}}$ low		220
6	$\overline{\text{CS}}$ or $\overline{\text{DS}}$ high to $\overline{\text{DTACK}}$ high		60
7	$\overline{\text{CS}}$ or $\overline{\text{DS}}$ high to $\overline{\text{DTACK}}$ high impedance		100
8	$\overline{\text{CS}}$ or $\overline{\text{DS}}$ high to data invalid hold time	0	
9	$\overline{\text{CS}}$ or $\overline{\text{DS}}$ high to data high impedance		50
10	$\overline{\text{CS}}$ or $\overline{\text{DS}}$ high to address invalid hold time	0	
11	Data valid from $\overline{\text{CS}}$ low		250
12	Read data valid to $\overline{\text{DTACK}}$ low setup time	50	
13	$\overline{\text{DTACK}}$ low to $\overline{\text{DS}}$ or $\overline{\text{CS}}$ high hold time	0	

* If the setup time is not met, $\overline{\text{CS}}$ will not be recognized until the next falling clock.
Source: Motorola.

if the address decoder output drives $\overline{CS}$ directly. Gating the address decoder output with $\overline{AS}$ helps, but not enough, since the 68008 guarantees only 20 ns of address setup before $\overline{AS}$ is asserted. (This solution would likely work but does not work worst-case. The setup time provided to the MFP would be the setup time provided by the 68008, plus the delay of the gate. With a minimum gate delay of a few nanoseconds, if the 68008 provided 5 to 8 ns more setup time than is guaranteed, the resulting setup time would be adequate.)

One solution to the setup-time problem is to delay the $\overline{CS}$ signal with two inverters in series. However, this solution depends on minimum gate delays. A better solution that provides proper timing even in a worst-case situation is shown in the schematic. The flip-flop is held set when $\overline{AS}$ is negated, and it is cleared (thus asserting the $\overline{CS}$ signal) at the first rising clock edge after $\overline{AS}$ is asserted if $\overline{MFPCS}$ is also asserted. This delays $\overline{CS}$ sufficiently to ensure that the register select setup time is met. The calculation of the actual setup time provided is left as an exercise for the reader.

5.9 SUMMARY

Peripheral chips supplement the microprocessor's capabilities. In addition to providing basic I/O capability, they are useful for timekeeping functions. Programmable timers can serve as frequency generators, frequency counters, or time-base interrupt generators. For applications requiring calendar time, clock/calendar chips keep time even when the system power is off and eliminate the need for the software to deal with the peculiarities of incrementing seconds, hours, minutes, days, months, and years. When long-term unattended operation is important, watchdog timers put the system back on track in case a noise spike or other uncommon event causes the system to malfunction.

DMA controllers increase the rate at which the microprocessor system can move data and reduce the load on the microprocessor. They are most useful when used with high-speed peripherals that transfer blocks of data. Since they must perform all the bus read and write operations that the microprocessor itself performs, compatibility with the microprocessor used is essential.

Many microprocessor families include a variety of peripheral chips that are designed for easy interfacing to that microprocessor. However, some peripheral chips are designed to be general-purpose, and it is sometimes desirable to use a chip from another microprocessor family. In this case, additional logic is required to translate the control signals, and a careful timing analysis must be performed to ensure compatibility.

5.10 EXERCISES

5.1. A peripheral chip has the following registers:

Data register	(read/write)
Command register	(write only)
Status register	(read only)
Interrupt Vector register	(write only)

a. List the control and address signals needed for the interface.
b. Assuming that there is one bit available in the command register, design an interface scheme that does not require any address lines. Assume an 8-bit data bus.

5.2. A microprocessor-based instrument must write to its display every 25 ms. Assuming a 4-MHz system clock, what values must be written to the registers of a Z80-CTC for it to provide an appropriate interrupt signal?

5.3. In the instrument described in Exercise 5.2, it is also necessary to read an input port exactly once per minute. How can this time interval be generated with the Z80-CTC?

5.4. A security system must operate reliably for long periods of time, so a watchdog timer circuit is included. The clock for the timer is 2 MHz, and the software can reset the timer no more frequently than once every 100 ms. The output required in case of a failure is a 10-ms minimum reset pulse. Design the circuit for the watchdog timer. How many counter stages are required?

5.5. Draw a flowchart for a program to implement a clock/calendar function, using an interrupt from a programmable counter/timer as the time base. Select either 12- or 24-hour time format. The program must account for the varying length of the months, but leap years may be ignored.

5.6. A personal computer includes a floppy disk interface that provides bursts of data at a rate of 62,500 bytes/s. A program to perform the transfer requires 45 clock cycles to read each byte and prepare for the next. What is the minimum clock rate required for the system to operate without a DMA controller?

5.7. A microprocessor-based laboratory instrument collects data from a data acquisition system (DAS). The DAS buffers data from a variety of sources, and every 100 ms a block of 1024 bytes is available to be read. The data can be read at any rate, since it is buffered by the DAS, but all 1024 bytes must be read within the 100-ms period. A program to read and store one byte of data takes 10 μs to execute.

a. Is a DMA controller required?
b. The instrument can use all the available processing time to perform calculations based on the data. A DMA controller is used that reads the data at a 1-Mbyte/s rate. What is the percent increase in available processing time as compared to a non-DMA configuration (if it is possible without DMA)?
c. In different applications, the DAS provides 8192 or 16,384 bytes of data every 100 ms. Repeat parts a and b above for each of these data block sizes.

5.8. Draw a complete timing diagram for the 68901-to-68008 interface as used in the design example. How many wait states are produced during 68901 accesses? What is the register-select-to-chip-select setup time?