Host Interface 18

18.1 OVERVIEW

This chapter describes host interfacing techniques in general using the example of the Motorola 680x0 family of processors. Today's computer CPUs are very powerful. They have large, versatile instruction sets and addressing modes. They can handle complicated sequencing and stacking, address and manage large data spaces, even while keeping track of virtual memory, user and supervisor modes, etc. However, these CPUs do not perform fast numerical operations. The ADSP-2100 can act as a fast numerical coprocessor to a host CPU. The host CPU takes care of administrative duties while the ADSP-2100 processes numerical data concurrently. This coprocessor configuration is useful in computationally intensive applications, such as graphics, spectral analysis, data compression, linear algebra, vector estimation, encryption, error coding, image processing, and speech recognition.

18.2 INTERFACE CONFIGURATIONS

One of the most common host/coprocessor architectures is the master-slave arrangement. The host CPU acts as the master in a system, passing data and/or instructions to the slave coprocessor. Communication from the slave coprocessor back to the host is usually very minimal, consisting of interrupt signals sent to the host or flags set for the host to poll.

Host interfacing is also an issue for interprocessor communications in distributed processing architectures. Hardware and software interface issues are similar to those of the master-slave configuration. Although only the master-slave(s) architecture is discussed in this chapter, distributed processing intercommunications can be developed from the principles outlined here.

There are four general methods of communication between processors:

- Memory bus sharing/arbitration
- Hardware communication ports separate from the memory interface
- Concurrent memory sharing using dual-ported memory
- Memory swapping

18.2.1 Bus Sharing

The ADSP-2100 has bus request (BR) and bus grant (BG) pins for memory bus sharing and arbitration protocol (see Figure 18.1). A host CPU asserts the BR input of the ADSP-2100 to request access to the ADSP-2100's local program or data memory. The ADSP-2100 responds by asserting its BG output and releasing control of its memory interface. The host CPU can then drive the ADSP-2100's memory interface with its own signals. Data stored in data memory or program memory can be read, written or modified, and program instructions in program memory can be initialized or inspected. When the host CPU is done driving the ADSP-2100's memory interface, it deasserts the BR input. The ADSP-2100 responds by re-establishing its memory interface, deasserting BG and continuing program execution.

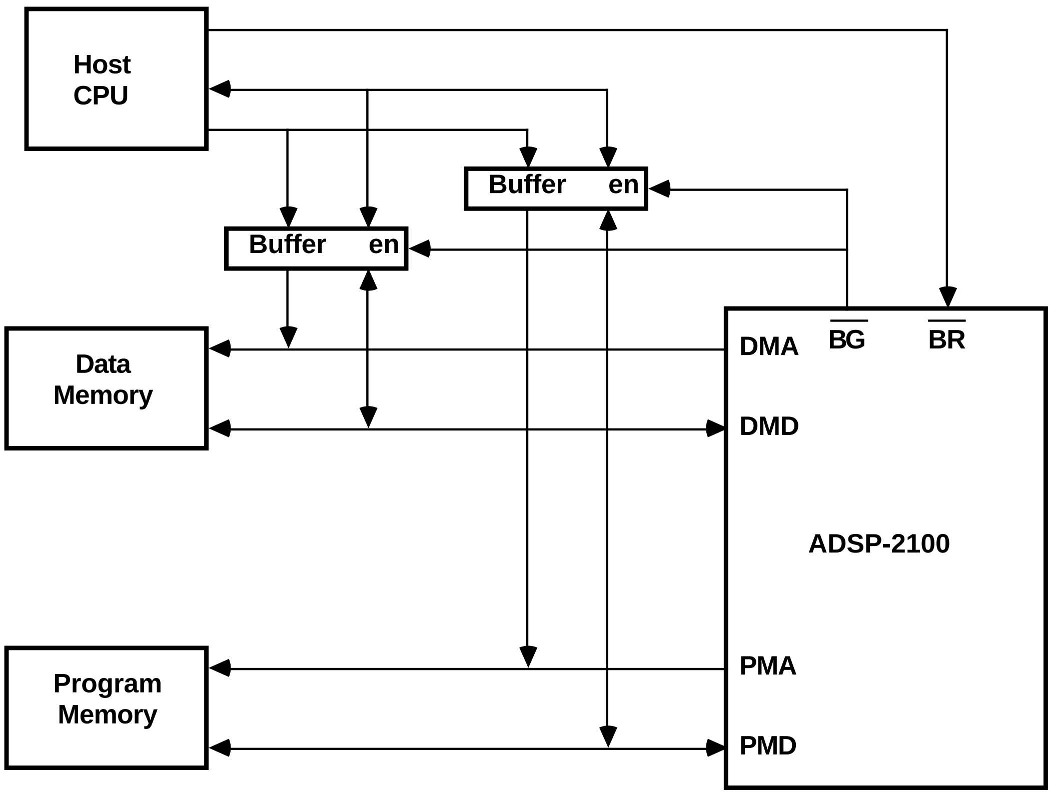


Figure 18.1 Bus Sharing Interface

18.2.2 Communication Ports

Hardware communication ports on chip provide a communication channel separate from the memory interface. Whether serial or parallel, these ports usually operate at a rate considerably slower than direct memory access through bus sharing. The ADSP-2100 does not provide any hardware communication ports. However, the ADSP-2101/2 microcomputer has two serial communication ports (see Figure 18.2). Use of the serial ports is explained in the *ADSP-2101/2 User's Manual* and the application note "Loading an ADSP-2101 Program via the Serial Port."

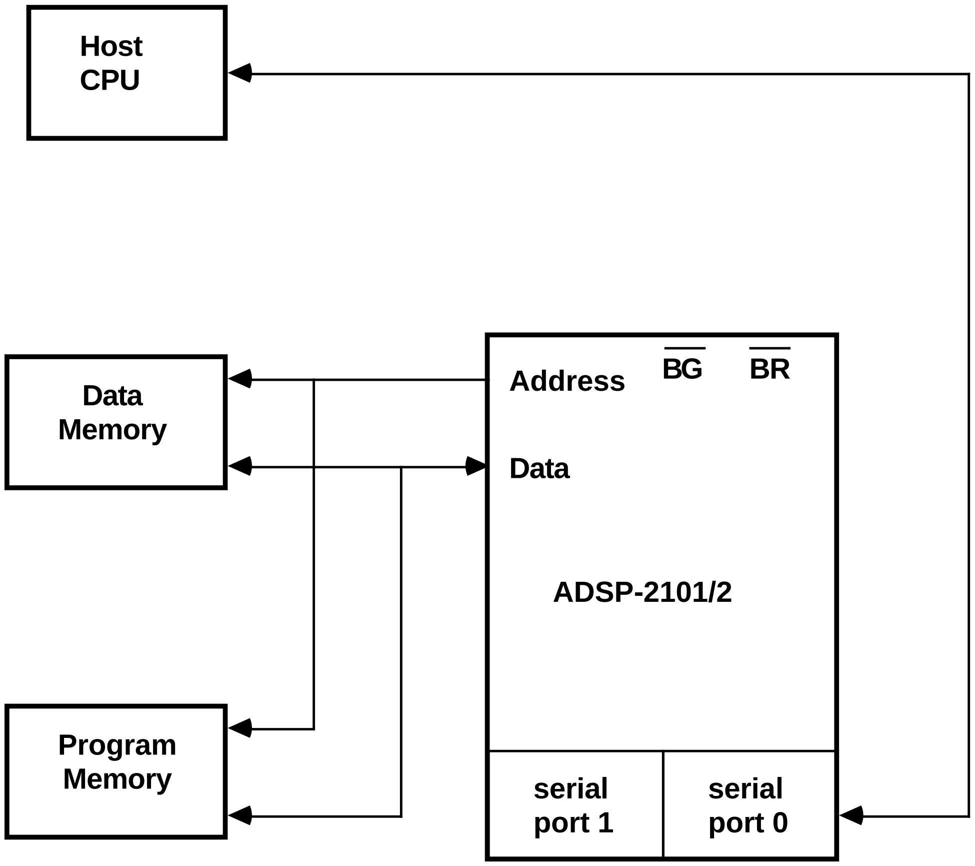


Figure 18.2 Communication Port Interface

18.2.3 Dual-Port Memory

Dual-port memories are high-speed SRAM memories which have two sets of address, data and read/write control signals (see Figure 18.3). Each set of memory controls can independently and simultaneously access any word in the memory; both sides of the dual-port RAM can access the same memory location at the same time. For host interfacing, you connect one side of the dual-port memory to the main CPU and the other side to the ADSP-2100. Because the two processors can write to the same location in memory at the same time, or one can read a location while the other is changing the same location, there must be some arbitration to decide which device has precedence, and there must be a way to hold off the processor of lower priority so it can extend its bus cycle until the memory is accessible.

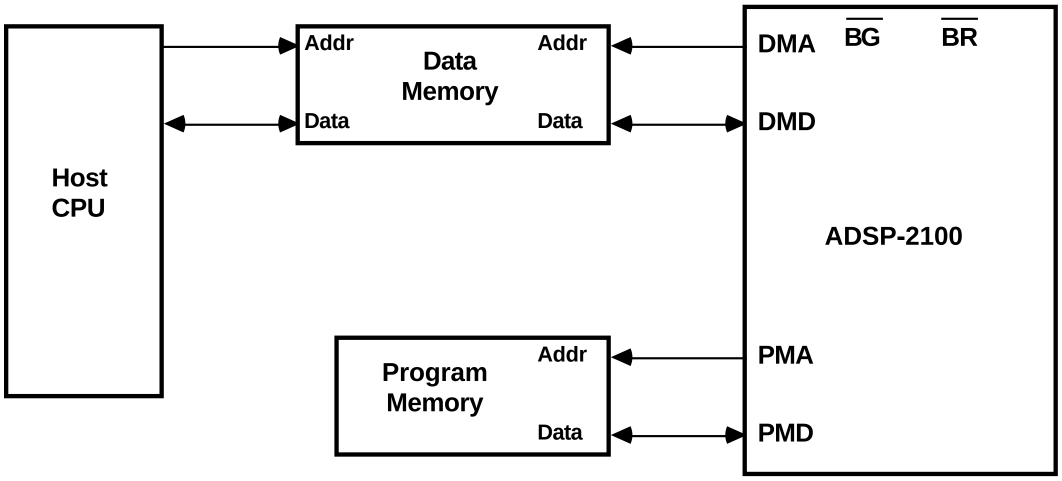


Figure 18.3 Dual-Port Memory Interface

Dual-port memories offer four types of arbitration:

- *Hardware busy logic* generates a busy signal to one port when the other port is writing the same location. The busy signal holds off Port B access to a location to which Port A is writing, and vice versa.

- *Semaphore logic* has on-chip semaphore latches that can be set and polled; each port passes a flag, or token, to the other to indicate that a shared resource is in use. Semaphore logic allows one processor to lock out the other when accessing a particular block of data.

- *Interrupt logic* generates an interrupt at Port B when Port A writes to a special location. Port B clears the interrupt by reading that location. When Port B writes to another special location, a similar interrupt is generated at Port A.

- *No arbitration logic* at all is used on some dual-port RAMs. If these devices are used, the system must have been designed so that contention never occurs.

See Chapter 17, *Multiprocessing*, for an example of the use of dual-port memory.

18.2.4 Memory Swapping

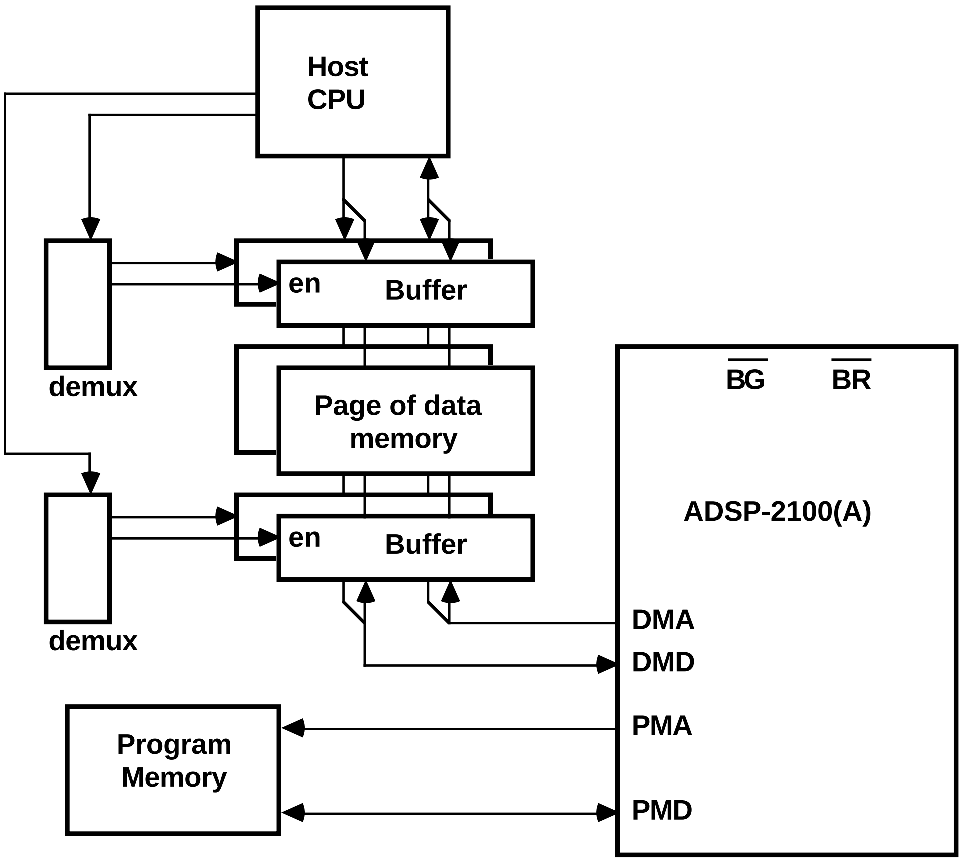
Memory swapping uses several hardware copies of the same memory space; only one copy (called a frame) is enabled at a time per processor (see Figure 18.4, on the following page). A different frame is enabled for each processor. The host processor can load one frame with data while the slave processor is processing data in another frame. When both processors are done, the enable signals to the memories are switched, and it appears to each processor as if all the data in its frame has changed instantaneously. This scheme is often used in systems in which multiple processors compute assigned sections of a larger algorithm. When two frames are used, or when two buffers within a frame are exchanged in software, this is called ping-ponging.

18.3 ADSP-2100 INTERFACE CONSIDERATIONS

This section describes in detail the considerations for implementing a bus sharing interface to the ADSP-2100.

18.3.1 Bus Request

The normal synchronous mode of granting a bus request proceeds as follows. The external device (host CPU) requests the buses by asserting the BR input of the ADSP-2100. This input is recognized by the ADSP-2100 at the end of the next internal clock state three, and the ADSP-2100 halts in state eight of the same instruction cycle. The ADSP-2100 handshakes with the host CPU by asserting BG at the end of state three of what would have been the next instruction cycle (four CLKIN cycles after the bus request is recognized). Typically, the BG output of the ADSP-2100 is used to enable bus transceiver chips (74F244 or 74F245), establishing a connection between the host CPU's bus signals and the ADSP-2100's memory.



Interface to only data memory shown

The host must ensure that it and the ADSP-2100 use different memory pages

Figure 18.4 Memory Swapping Interface

The ADSP-2100 then tristates all outputs, including PMA, PMD, PMDA, PMWR, PMRD, and PMS for program memory, and DMA, DMD, DMWR, DMRD, DMS for data memory. The PMD, PMS, PMWR, PMRD, DMD, DMS, DMWR and DMRD signals are internally pulled up by 50kΩ while BG is active. The PMA, PMDA, and DMA signals are not internally pulled up.

When the host CPU is finished accessing the ADSP-2100's memory, it deasserts BR. The ADSP-2100 deasserts BG four CLKIN cycles (internal clock states) later and re-establishes control over its program and data memory signals in state one of the next instruction cycle, resuming operation as it left off.

A bus grant can occur even in the double instruction cycle execution of a program data memory fetch when the cache is not valid. It is possible to interrupt that instruction's execution with BR in between the two cycles.

The ADSP-2100's internal state is not affected by the bus granting operation. The only restriction is that RESET should not be changed during a bus grant. Activity on the RESET input while the ADSP-2100 is asserting BG causes indeterminate operation. When downloading a program to program memory from an external device (on power-up for example), you should proceed in the following order. This order ensures that program execution starts at PC=h#0004 with the ADSP-2100 in a known state.

1. Assert the RESET signal for the standard full instruction cycle
2. Assert the BR signal
3. Download the information
4. Deassert the BR signal
5. Deassert the RESET signal

During RESET, the timing of BR and BG is different from their timing during normal operation; the operation is asynchronous during RESET. See the *ADSP-2100/2100A Data Sheet* or the *ADSP-2100 User's Manual* for details.

18.3.2 Software Handshake

Handshaking information is needed to tell the slave coprocessor when new data is available and the host when the slave coprocessor is done processing its data. A simple method uses two locations in ADSP-2100 memory reserved as flags: one, which the ADSP-2100 polls, to indicate the presence of new data, and the other, which the host polls, to indicate the completion of the data processing task. The ADSP-2100 can poll the new-data flag until data is available, process the data, set the data-done flag, and go back to polling for new data. An example of the ADSP-2100 code to execute this loop is shown below:

```
.VAR/ABS=h#3FFE   new_data;
.VAR/ABS=h#3FFD   data_done;
.INIT             new_data:   0;          {0=FALSE, else =TRUE}
.INIT             data_done:  0;          {0=FALSE, else =TRUE}

idlestart:        DO idleloop UNTIL NE;   {wait for new data}
                     AR=DM(new_data);
idleloop:            AR=PASS AR;
                  AR=PASS 0;
                  DM(new_data)=AR;        {clear the flag}
                  CALL process_it;        {process the data}
                  AX0=h#FFFF;
                  DM(data_done)=AX0;   {indicate processing done}
                  JUMP idlestart;
```

18.3.3 Hardware Handshake Using Interrupts

The ADSP-2100 and host CPU can also handshake by sending hardware interrupts to indicate, for example, that new data is available or that data processing is complete. The simplest configuration connects the ADSP-2100's TRAP output directly to the host CPU's interrupt input. The routine that the ADSP-2100 executes is terminated by a TRAP instruction. This halts the ADSP-2100 and asserts the TRAP output high. The host's interrupt service routine can use $\overline{\text{BR}}$ and $\overline{\text{BG}}$ to download new data to the ADSP-2100's memories and then assert and release the ADSP-2100's $\overline{\text{HALT}}$ line, causing the ADSP-2100 to continue processing on the next instruction. The next instruction can jump back to the beginning of the same routine.

Another way to handshake with interrupts is to have address decoding logic generate an interrupt to one processor when the other processor accesses a particular memory address. The interrupt signal can be tailored for an active low, active high, or edge-triggered interrupt, depending on the input of the particular processor.

18.3.4 Software and Hardware Handshake Comparison

The advantage of a hardware handshake over a software flag handshake is speed. The receiving processor responds to the interrupt as soon as it becomes active. Code is reduced as well because there is no need to perform the polling functions using $\overline{BR}$ and $\overline{BG}$.

The advantage of software flag polling over the hardware interrupts is that the whole system remains in a static state on a cycle-by-cycle basis. The handshaking status can be inspected using the CPU's software development tools, making the system easier to debug. In addition, there is no need to redesign interrupt logic if the interrupts are used for other purposes. The example described in this chapter performs software flag handshaking.

18.4 68000 INTERFACE CONSIDERATIONS

This section describes the characteristics of the 68000 microprocessor that are relevant to the ADSP-2100 interface.

18.4.1 68000 Addressing

The 68000 processor has a 16-bit data bus and a 24-bit address bus. Its 16 general purpose internal registers are each 32 bits wide. The 68000 recognizes five data types:

- Bit
- BCD (4-bit)
- Byte
- Word
- Long word (32-bit)

The 24-bit address bus allows the 68000 to access 16 megabytes of external memory; however, only 23 address bits actually are externally available as address pins. The least significant address bit (A_0) is not externally available. Therefore, the 16 megabytes of memory are actually organized as 8 megawords of 16-bit words located at even-numbered addresses (LSB = 0).

Byte addressing allows the 68000 to access 8-bit byte data on both even and odd address boundaries in memory. The $\overline{UDS}$ (upper data strobe) and $\overline{LDS}$ (lower data strobe) outputs of the 68000 are used for byte-oriented addressing.

18.4.2 68000 Bus Signals

The block diagram of the 68000 in Figure 18.5 shows the control signals to the 68000. The interface with the ADSP-2100 described in this chapter involves only the address bus, data bus, and asynchronous bus control signals. All other signals are left to the 68000 system designer's implementation because they do not affect the ADSP-2100 interface directly.

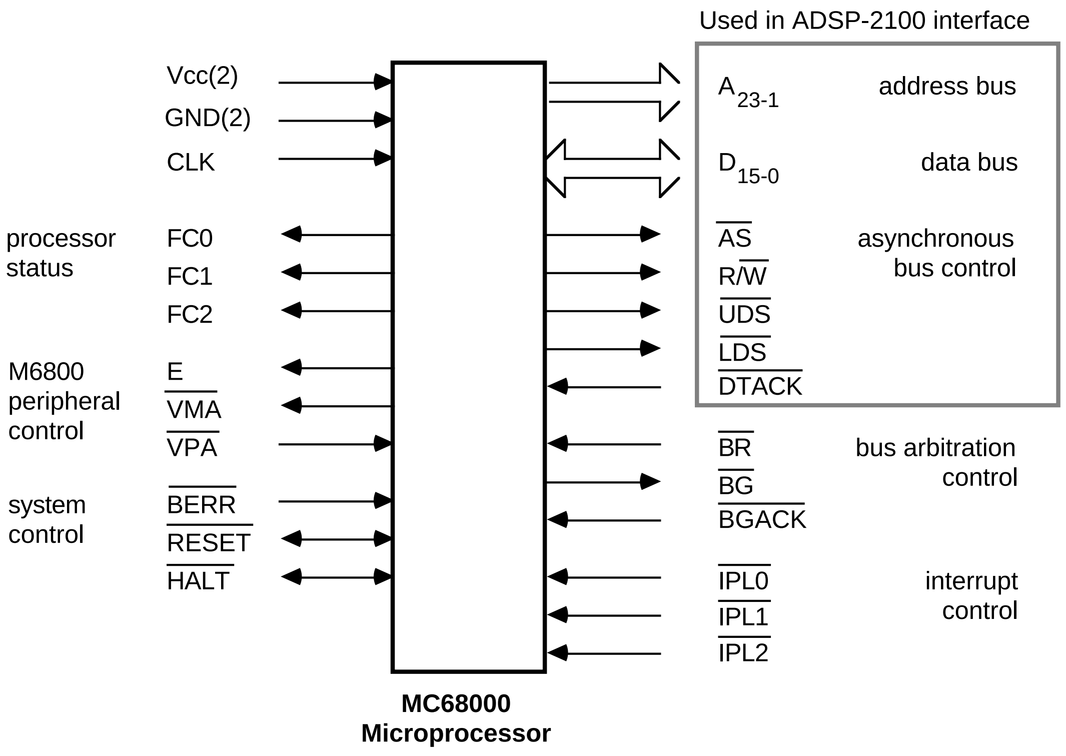


Figure 18.5 68000 Block Diagram

The control of the 68000's bus is asynchronous. Once a bus cycle is initiated, it is not completed until a handshaking signal (DTACK) is asserted by external circuitry. The signals that control address and data transfers are:

- Address strobe (AS)
- Read/write (R/W)
- Upper data strobe (UDS)
- Lower data strobe (LDS)
- Data transfer acknowledge (DTACK)

The 68000 asserts AS when an address is available and asserts R/W high or low to indicate whether a read or a write is to take place over the bus. Because the bus cycle is asynchronous, external circuitry must signal the 68000 when the bus cycle can be completed by asserting the DTACK input to the 68000. During a read cycle, DTACK low tells the 68000 that valid data exists on the data bus. In response, the 68000 latches in the data into the chip and terminates the bus cycle. Similarly, on a write cycle, DTACK low informs the 68000 that the data has been successfully written to memory or a peripheral device, and the 68000 responds by ending the bus cycle.

On the ADSP-2100, DMACK can be tied high if all memory accesses can complete in a single instruction cycle; in the absence of a DMACK low, the ADSP-2100 data memory read or write cycle ends in one cycle. Similarly, DTACK on the 68000 can be hardwired low if the design of the system is such that the 68000 bus cycle can always be completed without any wait states.

The UDS and LDS signals act as an extension of the address bus, replacing the address LSB A_0. In the case of a byte transfer, they indicate whether the data is on the upper data lines ($D_{15\text{-}8}$) or the lower data lines ($D_{7\text{-}0}$) as shown below:

R/W	*UDS*	*LDS*	*Operation*
0	0	0	word —> memory or peripheral
0	0	1	high byte —> memory or peripheral
0	1	0	low byte —> memory or peripheral
0	1	1	(invalid data)
1	0	0	word —> 68000
1	0	1	high byte —> 68000
1	1	0	low byte —> 68000
1	1	1	(invalid data)

18.5 68000-TO-ADSP-2100 BUS SHARING INTERFACE

The example in this chapter shows how a 68000 CPU can control DMA to and from an ADSP-2100 processor using bus sharing and arbitration. Other CPUs in the 680x0 family have similar interfaces. A schematic for the interface is shown in Figure 18.11 later in this chapter.

18.5.1 Memory Mapping

Data transfers between the 68000 and the ADSP-2100 occur in a 16-bit word format because that is the data type expected by the ADSP-2100.

Note that 16-bit words are stored in ADSP-2100 memory on every address boundary, whereas the 68000 expects 16-bit words only on even address boundaries. In this example, the ADSP-2100 data memory space overlays the 68000 memory in the address range $02xxxx. The interface circuit maps the ADSP-2100 data memory addresses h#0000 to h#3FFF into 68000 memory addresses $020000 to $027FFE, as shown in Figure 18.6.

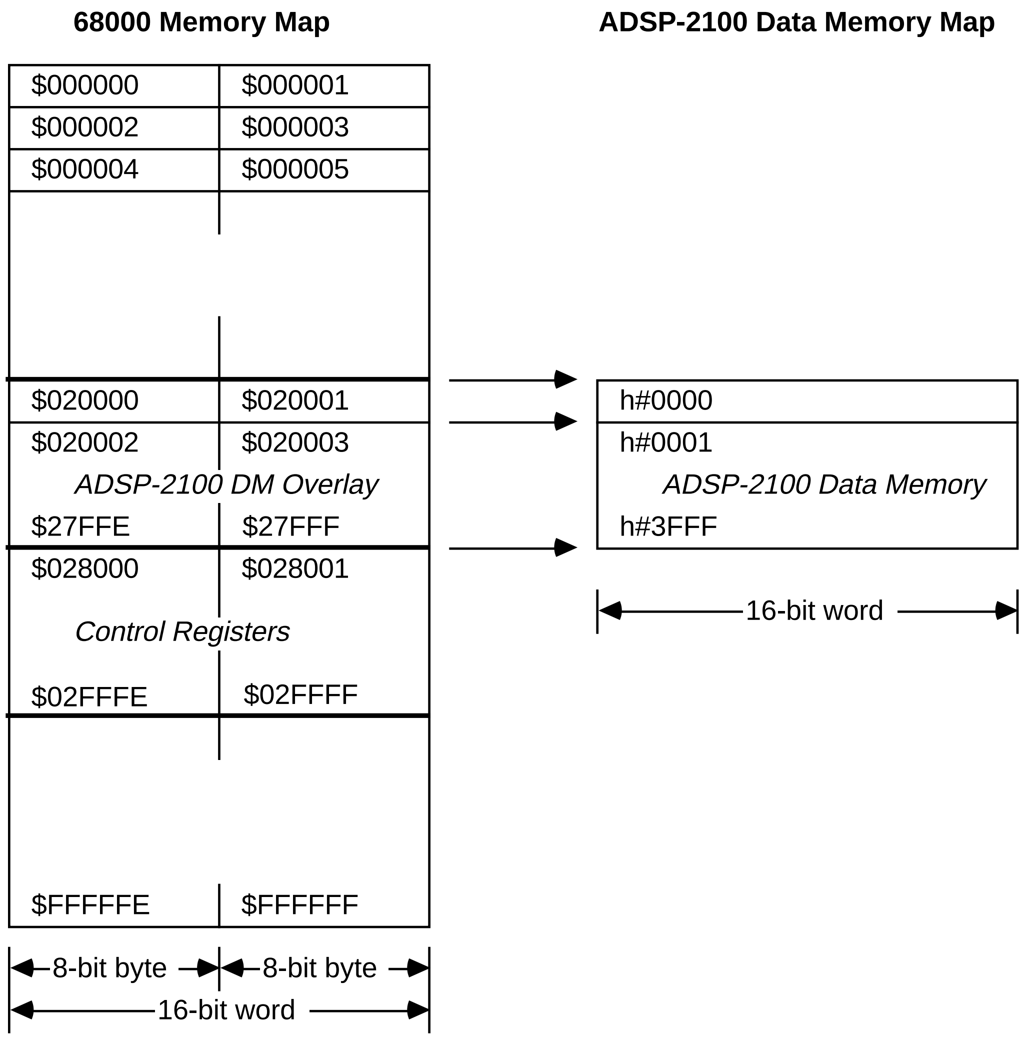
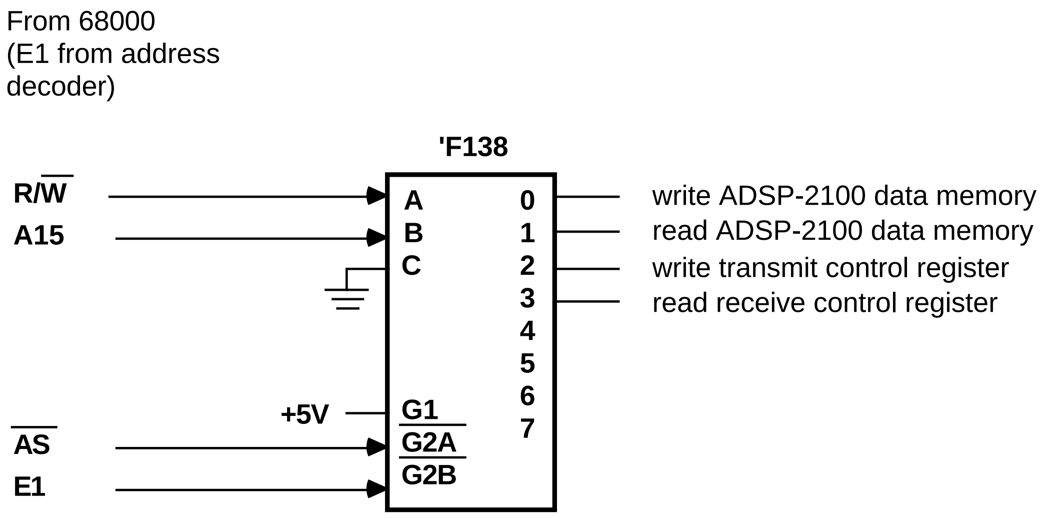


Figure 18.6 Memory Map Overlap

Software handshaking is implemented by memory-mapping two hardware control registers in the 68000's address space. Figure 18.7 shows the 68000 decoding logic to select between ADSP-2100 memory and the transmit and receive control registers. The control registers are used to pass hardware signals between the 68000 and the ADSP-2100, under software control. These registers are described in the next section. The 68000 address bit A_{15} differentiates between accesses to the "68000 memory" that is actually the ADSP-2100 data memory space and accesses to the control registers. The control registers are memory-mapped in the 68000 memory space at address \$028000. However, because only \$02xxxx and A_{15} are decoded, the registers can be accessed at any address from \$028000 to \$02FFFE. The 68000 R/$\underline{W}$ output differentiates between the transmit and receive control registers.



A15	$R/\overline{W}$	$\overline{AS}$	E1	
0	0	0	0	write ADSP-2100 data memory
0	1	0	0	read ADSP-2100 data memory
1	0	0	0	write transmit control register
1	1	0	0	read receive control register
x	x	x	1	none
x	x	1	x	none

Figure 18.7 Decoder for Data Memory and Control Registers

18.5.2 Control Registers

The 68000 requests access to the ADSP-2100 memory by writing to the transmit control register. The 68000 reads back the status of BG from the ADSP-2100 by reading the receive control register. An example circuit for the transmit control register is shown in Figure 18.8. The register in this example consists of 74F74 D flip-flops. One of the 74F74 Q output pins is tied directly to the BR input of the ADSP-2100. To assert or deassert BR, software writes a logic 0 or 1 to the BR bit position.

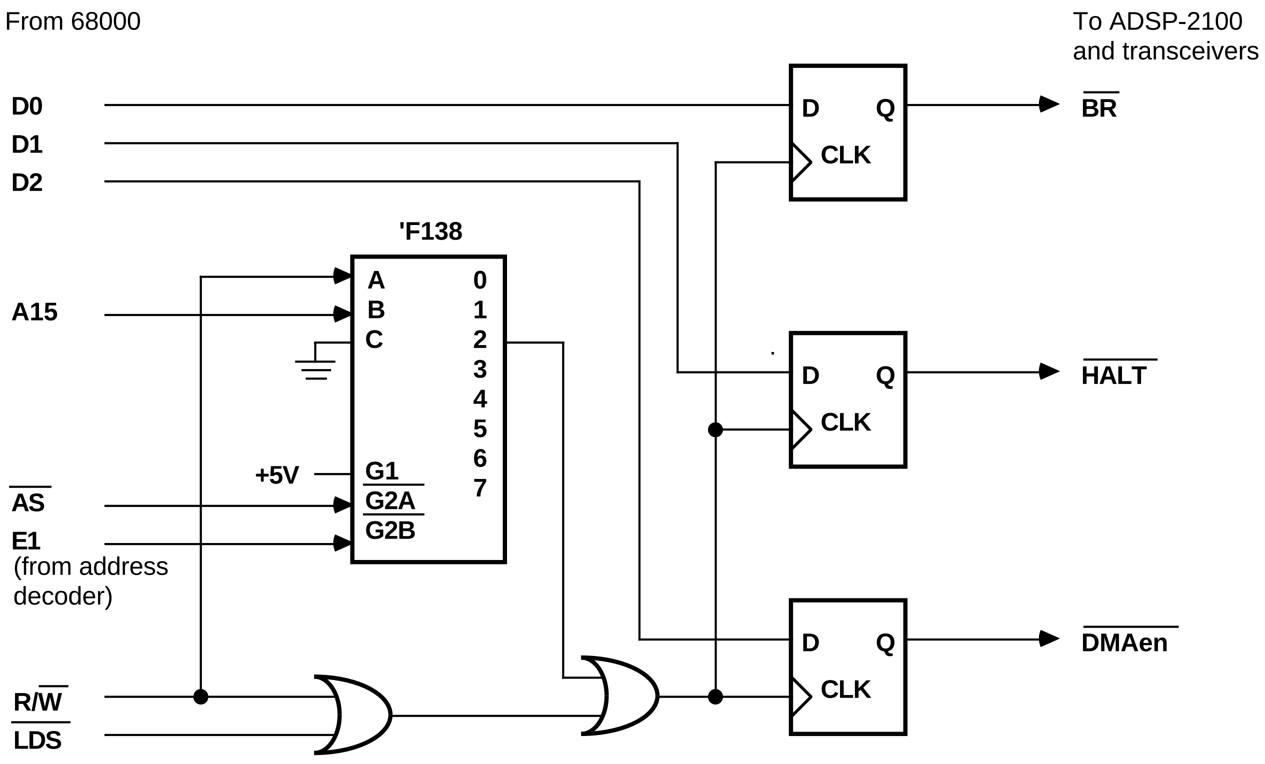


Figure 18.8 Transmit Control Register

An example receive control register circuit is shown in Figure 18.9. The receive control register is implemented by simply connecting the BG output of the ADSP-2100 to an input of a 74F244 bus transceiver and connecting the output of the transceiver to the 68000's data bus. The 74F244 bus transceiver is normally deselected, but when the 68000 outputs the address for the control register, the transceiver is enabled, and BG is then electrically connected to the 68000's data bus. The same transceiver can be used to inspect seven other hardware logic levels or to provide a read-back function to the transmit control register or can even be expanded in parallel to monitor more signals.

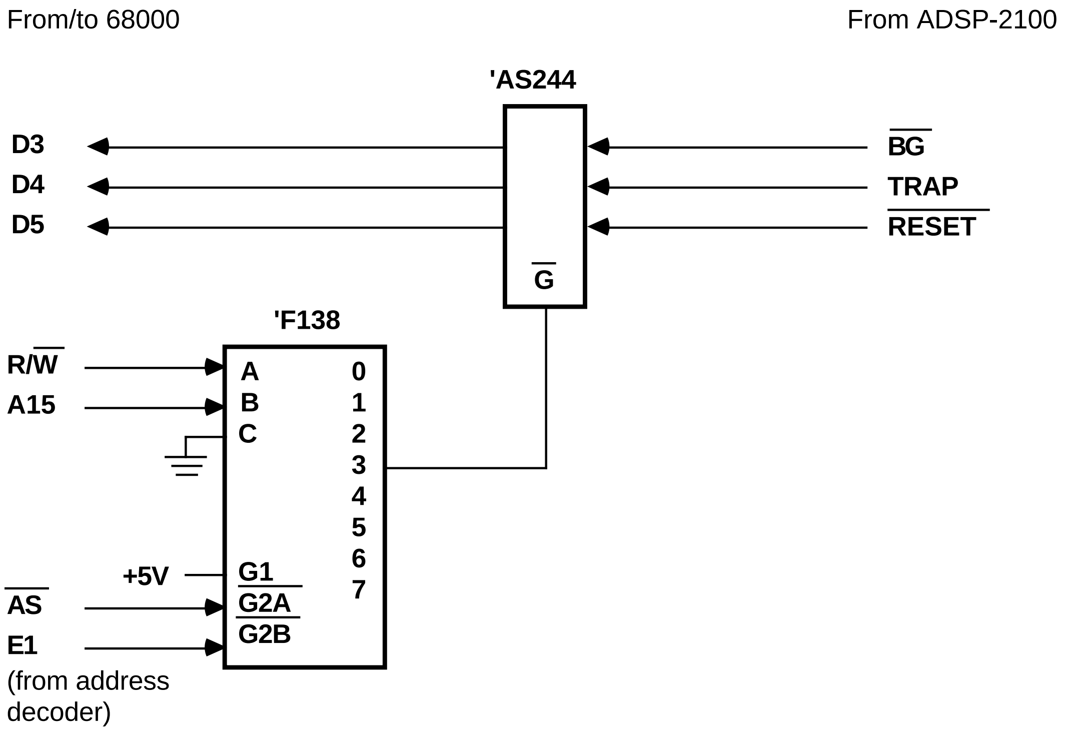


Figure 18.9 Receive Control Register

Examples of the 68000 address decoding logic used to select the hardware control registers are shown in Figure 18.10. Each register's address is decoded with a 74F138 and/or 74F521 and is qualified by AS. The output of the decoder clocks the data from the 68000 data bus into the 74F74 flip-flops of the transmit control register or enables the transceiver of the receive control register.

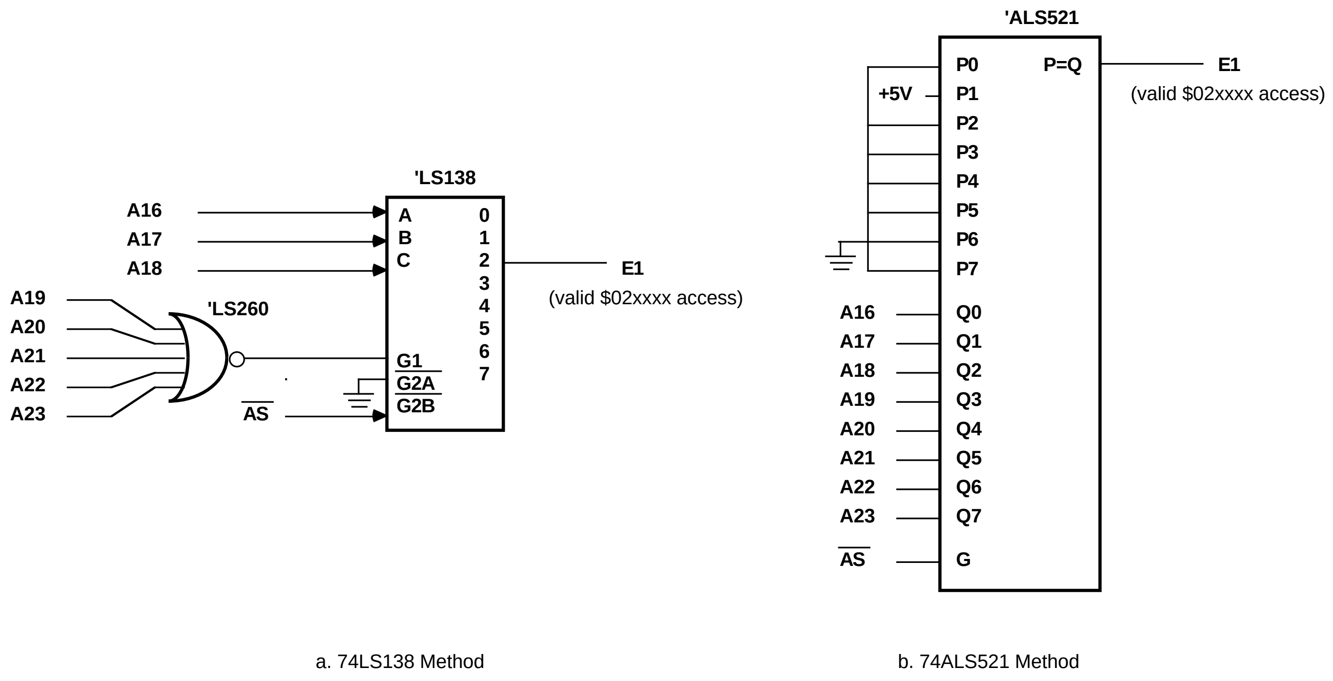


Figure 18.10 Two Methods for Decoding $02xxxx Accesses

When the 68000 writes to the transmit control register, DTACK is returned to the 68000 by a 74F244 buffer that is enabled whenever a valid access to address $02xxxx memory space (the interface circuitry) is detected. Because the 68000 in this example runs so much slower than the ADSP-2100, there is no need to extend the 68000 bus cycle. Therefore, the DTACK input to the 74F244 buffer is simply tied to ground. In a different application requiring wait states in the 68000 bus cycle, the grounded input of the 74F244 would be replaced by the output of a wait state generator.

18.5.3 Interprocessor Data Transfers

Once control of the ADSP-2100's buses has been granted to the 68000, the 68000 must be able to send and receive data and memory control signals. Another bit in the transmit control register enables a pair of 74F244 bus buffers which provide a path for 68000-generated addresses to the ADSP-2100's data memory address bus. Because the two 74F244 buffers can pass 16 signals, but the ADSP-2100 requires only 14 address signals, the two extra lines can be used for memory control signals. They can be used to send DMRD and DMWR, while the existing ADSP-2100 address decoding logic generates DMS. In this example, the ADSP-2100 data memory has all DMRD inputs grounded, and DMS is used as an enable to the decoding logic (74F138), so the two lines are used to send DMS and DMWR. A 4.7kΩ resistor pulls up the DMWR signal so that when the 74F244 buffers are deselected (outputs are tristated, but BG is still asserted), the DMWR signal is forced high.

Two 74F245 bus transceivers connect the 68000's data bus to the ADSP-2100's data bus. In this case, all 16 bits are used for data. The transceivers' enables (G) are connected to address decoding logic. The direction controls (DIR) are tied directly to the 68000's R/W output.

The direct memory access operation between the 68000 and the ADSP-2100 proceeds as follows:

1. The 68000 and ADSP-2100 are processing separate tasks
2. The 68000 writes a 1 to the BR bit in the transmit control register
3. The 68000 reads the receive control register, and continues to read it until the BG bit is low
4. The 68000 writes a 1 to the bit in the transmit control register which enables the address buffers to the ADSP-2100 data memory (make sure to keep the BR bit set)
5. The 68000 reads or writes ADSP-2100 memory by accessing addresses $020000 to $027FFE in its own memory space
6. The 68000 writes a 0 to the bit in the transmit control register to deselect the address buffers
7. The 68000 writes a 0 to the BR bit in the transmit control register to deassert BR
8. The 68000 reads the receive control register, and continues to read it until the BG bit is high
9. The 68000 and ADSP-2100 continue processing separate tasks

Important: If the 68000 is running much slower than the ADSP-2100, it is possible to simplify the 68000 software by not checking for the deassertion of BG in step 8, and steps 6 and 7 can be combined into one data write. Step 3 should not be skipped, however, because the ADSP-2100 might be waiting for DMACK to be asserted, in which case the ADSP-2100 would not immediately assert BG. When the ADSP-2100 is halted or in a TRAP state, the bus request is latched by the ADSP-2100 and serviced after the normal synchronization delay; in this case, the ADSP-2100 remains halted, and tristates its buses. See the *ADSP-2100 User's Manual* for details.

18.6 USING PALS

The ADSP-2100 interface to a host processor using bus arbitration involves some logic functions which can be performed by a PAL (programmable array logic; a registered trademark of Monolithic Memories, Inc.) device rather than discrete logic devices. The savings in chip count is not significant because most of the chips needed for the interface are bus buffers and transceivers. It may be desirable, however, to integrate discrete gate functions into a PAL to allow more flexibility; the PAL can be reprogrammed, if necessary, to debug the circuit. A GAL (generic array logic; a registered trademark of Lattice Semiconductors, Inc.) device can emulate several types of PALs, providing even more flexibility.

To program a PAL or a GAL, you determine Boolean expressions for all the output signals as functions of the input signals. PAL programming software generates a file from these expressions which is used to burn (program) the PAL.

The following example of generating the BR signal shows how to derive the Boolean equations for PAL programming. BR is one bit of the transmit control register and is shown in the schematic of Figure 18.11 as the output of a D-latch or flip-flop. A registered PAL has internal D-latches (registers); one of the D-latches on a registered PAL can be used instead of discrete logic to generate BR. The three signals needed for the D-latch are D (input), Q (output), and CLK. The input is the LSB (D_0) of the 68000 data bus, and the output is BR. CLK must be generated internally by the logical combination of other input signals. The transmit control register is clocked by a combination of R/W low, LDS low, A_{15} high, AS low, and E1 ($02xxxx access) low (see the schematic in Figure 18.11). These signals are inputs to the PAL and are logically combined to yield an output signal which is tied directly to the PAL's clock input (pin 1 on a 16R4A).

PLEASE PASTE IN
FIGURE 18.11 HERE

Figure 18.11 68000 Interface Schematic

18.7 680X0 FAMILY OF MICROPROCESSORS

The example in this chapter shows the ADSP-2100 interface with the Motorola 68000 processor, a typical host CPU. Any other CPU requires a similar interface, including the other processors in Motorola's 68000 family. Some of the differences are described in this section. See Johnson in *References* for more information about the differences between the 680x0 processors.

The other processors in the 68000 family are:

68008	8-bit byte version of the 68000
68010	same as 68000, but supports virtual memory
68012	extended physical address space as well as virtual memory support
68020	complete 32-bit virtual memory processor, coprocessor interface, instruction cache, dynamic bus sizing, more instructions, more addressing modes
68030	similar to the 68020, but more of everything, including data cache

Because all processors in this family are object-code upward compatible, the interface software is easily adapted from one family member to another. And because the 68000's asynchronous bus signals are practically the same throughout the 68000 family, the hardware is likewise easily adapted from one family member to another.

The 68010 hardware interface is exactly the same as the 68000 interface because the two processors are pin compatible. The 68010 supports virtual memory; if the ADSP-2100 has a full complement of memory physically present, there is no need to be concerned with page faults (access to virtual memory outside the currently active physical memory).

The 68012 is similar to the 68010 in its virtual memory support, but it has 30 external address bits (A_{29-1}, A_{31}) instead of 23 (A_{23-1}). Its asynchronous bus signals are the same as those of the 68010 and 68000, so the 68012 interface is the same as the 68000 interface except that it must decode more address bits. The 68012 has an additional asynchronous bus signal, $\underline{\text{RMC}}$ (Read-Modify Cycle), which indicates that an indivisible read-modify-write cycle is being executed. Because read-modify-write operations are not necessary for the ADSP-2100 interface, this signal can be ignored.

The 68008 is similar to the 68000, but it has only an 8-bit wide external data bus. Its asynchronous bus signals are the same as the 68000's, except that UDS and LDS are consolidated into DS. If ADSP-2100 uses only 8 bits of its data bus, the 68008 interface is the same as the 68000 interface; however, if 16-bit data is needed, you must multiplex the 68008 data bus into high-byte and low-byte operations or, better yet, use a 68000.

The 68020 is a complete 32-bit processor with virtual memory support. All 32 address bits (A_{31-0}) and all 32 data bits (D_{31-0}) are available externally. The 68020 supports more instructions, more addressing modes, and more supervisor functionality than the 68000, plus a 256-byte instruction cache. The 68020 also features a coprocessor interface, intended for instruction-mapped coprocessors, such as the MC68882 floating-point coprocessor, not a memory-mapped coprocessor, such as the ADSP-2100 in this example.

An important additional feature of the 68020 is dynamic bus and data sizing. Because the 68020 can operate on many data types (bit, byte, word, long-word), the UDS and LDS signals of the 68000 are replaced by a data strobe (DS) used in conjunction with two data-size output encoding signals (SIZ1, SIZ0). The SIZ1 and SIZ0 outputs indicate the number of bytes of an operand remaining to be transferred during a given bus cycle. Because all transfers between the 68020 and the ADSP-2100 are assumed to be 16-bit words, the SIZ1 and SIZ0 signals can be ignored.

DTACK is replaced by two signals, DSACK1 and DSACK0, on the 68020. The DSACKx pins perform the same asynchronous bus transfer acknowledge function, but with greater functionality. The DSACKx pins are defined as follows:

DSACK1	*DSACK0*	*Result*
H	H	insert wait states in current bus cycle
H	L	cycle complete - data bus port size is 8 bits wide
L	H	cycle complete - data bus port size is 16 bits wide
L	L	cycle complete - data bus port size is 32 bits wide

For example, if the processor is executing an instruction that requires a read of a long-word operand, it attempts to read 32 bits in the first bus cycle. If the port is 32 bits wide, the 68020 latches all 32 bits of data and continues with the next operation. If the port is 16 bits wide, the 68020 latches the 16 bits of data and runs another cycle to obtain the remaining 16 bits. An 8-bit port is handled similarly, requiring four read cycles.

In the ADSP-2100 interface, the 68020 operates on 16-bit word data (specified in 68020 assembly language by the .W qualifier). Therefore, the

ADSP-2100 interface keeps DSACK0 always high, and returns DSACK1 high to insert wait states and low to complete the cycle. Notice that the 16 bits of the ADSP-2100 data must be tied to the 16 MSBs (D_{31-16}) of the 32-bit 68020 data bus. Additional asynchronous bus signals (ECS, OCS, RMC and DBEN) can be ignored for the ADSP-2100 interface.

The 68030 has all the features of the 68020, plus a data cache with several external cache control pins. Its asynchronous bus signals are identical to those of the 68020, and thus its ADSP-2100 interface is the same as that of the 68020.

18.8 SUMMARY

There are four general methods of interfacing a host processor to the ADSP-2100. The same methods can be used to interface multiple ADSP-2100s with a host processor or with each other. The method of bus arbitration using the BR and BG pins of the ADSP-2100 to provide an interface with a Motorola 68000 host CPU has been presented in this chapter. The same interface can be easily adapted to other members of the 68000 family.

18.9 REFERENCES

Johnson, Thomas L. 1986. “A Comparison of MC68000 Family Processors.” *BYTE Magazine.* Volume 11, Number 9. New York: McGraw-Hill, Inc.

Motorola, Inc. 1986. *M68000 8-/16-/32-Bit Microprocessors Programmer’s Reference Manual, fifth edition*. New Jersey: Prentice-Hall.

Motorola, Inc. 1985. *MC68020 32-Bit Microprocessor User’s Manual, second edition*. New Jersey: Prentice-Hall.

Motorola, Inc. 1982. *MC68000 Educational Computer Board User’s Manual, second edition*. Motorola, Inc. MEX68KECB/D2.

Motorola, Inc. *32-Bit Solution Information.* Motorola, Inc. Literature Package M32BITPAK.

Motorola, Inc. *M68000 Family Information.* Motorola, Inc. Literature Package M68KPAK/D.

Triebel, Walter A. and Avtar Singh. 1986. *The 68000 Microprocessor.* New Jersey: Prentice-Hall.