DUAL-PORT SRAM SIMPLIFIES PC-TO-TMS320 INTERFACE

APPLICATION NOTE AN-68

by Jim Handy, Barry Seidner and Jon Bradley

This application note describes a "no hassles" interface between the IBM PC-style backplane and a TMS320C30 DSP chip via an IDT dual-port static RAM. The interface provides an extremely simple means of downloading cross-compiled DSP code as well as sample data sets for debugging a high speed TMS320 based system in real time.

This example also shows how easily interprocessor communications hardware can be implemented via the simple insertion of a dual-port SRAM between a DSP chip and a general purpose processor in a standard DSP system. A system like this one would typically use a standard CPU for data input/output and ordering, and would pass complete data sets to the DSP chip for intense calculation. Similar architectures are often used in graphics and image processing, where an entire image is manipulated as a single data set, in transform calculations (i.e. FFTs) for sonar and radar processing. Certain systems even use this scheme several times with numerous DSP chips in order to get processing speeds proportional to the number of DSP chips in the system.

System Objective

The design presented here is the TMS320C30 Software Development Board. This board is one portion of a system which helps the TMS320C30 programmer to download and debug code from an IBM PC or similar computer. In order to support the special hardware needs of the TMS320C30 programmer, an expansion connector allows memory to be added to the DSP chip's primary bus, while a target connector provides a fully buffered version of the chip's expansion bus to allow its connection to special purpose hardware. Most of the TMS320C30's status signals are also routed to the expansion bus to make them available to the hardware being debugged.

The majority of the control software is PC-resident, and is provided on magnetic media. This includes such tools as the assembler, compilers, and download and debug routines. A 2K x 32 EPROM array on the primary bus of the TMS320 provides the host processor with a set of commands to allow it to load the software development board's RAMs, to set and clear breakpoints, to examine and preset internal status, and to load or store values in individual memory locations. All of these are controlled by the host's sending a command to the TMS320, which interprets that command and takes appropriate action.

A high speed 16K x 32 static RAM is attached to each of the DSP chip's two buses: the expansion bus, and the primary bus. The expansion bus' SRAM would typically be used to store a data set to be operated upon, and the primary bus' RAM would be used to store code which would be debugged using this board. Both of these SRAMs are zero wait-state (25ns access times at 33MHz) to allow real-time debugging and benchmarks to be performed. Since the TMS320's expansion bus only supports addressing of up to 8K locations, a bank select signal is used to switch between the upper and lower halves of this port's 16K x 32 memory. This signal is software-controlled from the processor's expansion bus.

One design goal for this system was to move data into and out of the DSP's dedicated memory without taking an inordinate amount of time or hardware. If standard memory were to be shared between the host and the DSP chip, multiplexing logic would need to be inserted between each processor and the RAM's address, data, and control lines. This logic would find itself right in the critical timing path of the memories on the primary and expansion buses, and would make zero wait-state operation nearly unachievable. An additional headache would have been finding room on the board for the large amount of multiplexing logic required. Should the design have used a simpler method of passing data back and forth between processors either via a UART or a single byte-wide I/O buffer, the developer would have had to endured long delays during download and other communication functions as the software on either side of the port performed massive amounts of handshaking to pass even the smallest of data sets.

It became obvious early in the design cycle that the simplest method of performing fast host to DSP communication would be to use a large high-speed true dual-port static RAM to perform interprocessor communications. A dual-port RAM would allow both the host and the DSP chip to transfer data in packets, rather than as individual bits or bytes, thus accelerating downloading. The selected dual-port device would have to be one which provided some means of signalling that data packets were ready to be handed back and forth between processors.

An IDT71342 was chosen because of its speed, its depth (4K bytes), the simplicity of its interface, and its ability to perform interprocessor communications through its eight internal semaphore flags (see Appendix: "Dual-Port Semaphores"). By using an IDT71342, the designers could use a single chip to implement 4K byte high speed block transfers between the host and the TMS320, and to signal the completion of a transfer without additional hardware. Although the 45ns access time dual-port used in this system does not support zero-wait data transfers at maximum CPU speeds, data transfers are not in the critical path of the sort of software this system is used to debug. A true zero wait-state system could have been

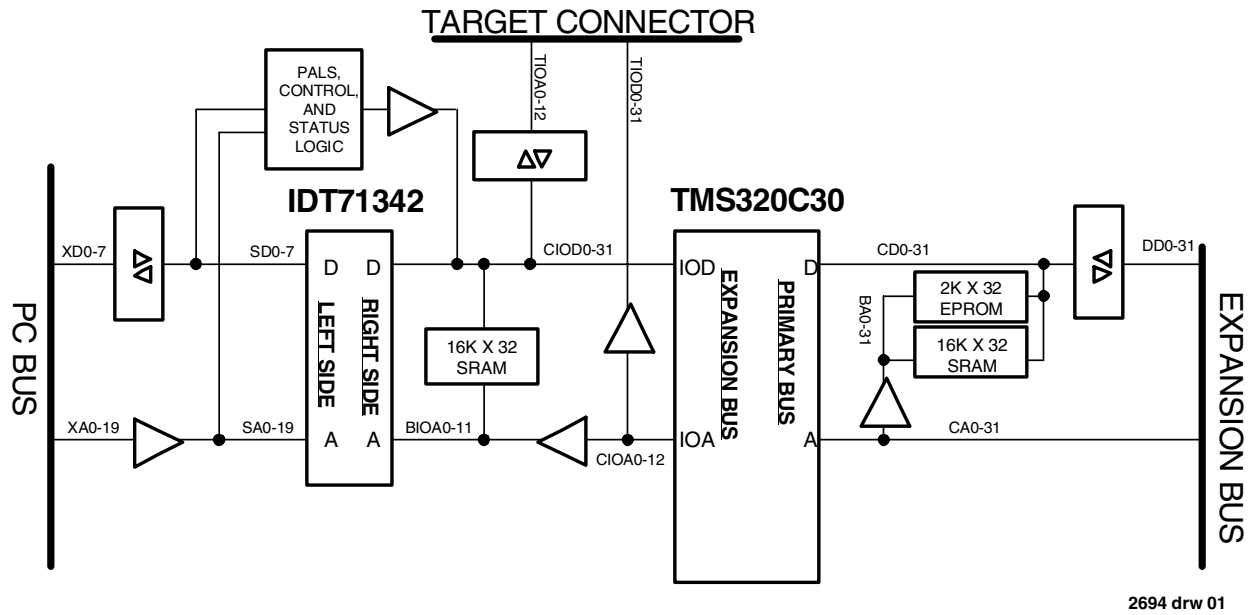

Figure 1. TMS320C30 Software Development Board Block Diagram

realized had the designers used a 25ns dual-port.

Figure 1 shows a block diagram of the complete system. The full schematic of the system is shown in Figure 6.

Interfacing to the Dual-Port SRAM

The IDT71342 dual-port RAM uses an interface which is similar to any standard single-port byte wide static RAM. Each of the two ports (left and right) uses a separate set of control, address, and I/O pins. Address inputs are not multiplexed with data I/O. The control interface consists of three pins on either side: read/write (R/$\overline{W}$), output enable ($\overline{OE}$), and chip enable ($\overline{CE}$). The R/$\overline{W}$ and $\overline{OE}$ pins also operate in conjunction with the semaphore select pin ($\overline{SEM}$), which imitates the functionality of the chip enable pin, but rather than allowing reads and writes of the memory array, this pin routes the read and write control to the eight on-chip semaphore flags.

Write cycles are controlled by the simultaneous application of a logic LOW on both the $\overline{CE}$ and R/$\overline{W}$ inputs for one side of the SRAM, and either signal can be used to control the timing of a write cycle. If the $\overline{CE}$ signal is held low and the timing is set by a LOW pulse on the R/$\overline{W}$ pin, it is called a "R/$\overline{W}$ controlled write cycle" (figure 2). Write cycles where R/$\overline{W}$ stays low while $\overline{CE}$ is pulsed low are called "$\overline{CE}$ controlled write cycles" (figure 3). By offering both methods of communication, IDT's dual-port SRAMs can be easily connected between systems with greatly differing bus interface specifications. An interesting point about this design is that while the PC or host side of the dual-port uses a R/$\overline{W}$ controlled write cycle, the DSP writes to its side of the dual-port by using a $\overline{CE}$ controlled write cycle.

The PC Bus Interface

In this design, the PC bus control signals are routed nearly directly from the backplane to the IDT71342's R/W and $\overline{OE}$ pins. The signal functions and timing of the backplane are an ideal match with those of the dual-port RAM. However, a decision was made to map the memory array into a 4K space in the PC's memory space, while the semaphores were to be mapped into the PC's I/O space, which forced the $\overline{IO}_W$ and $\overline{MEM}_W$ signals to be ORed before driving them into the IDT71342's R/$\overline{W}$ input. Likewise $\overline{IO}_R$ and $\overline{MEM}_R$ signals are ORed before driving them into the IDT71342's $\overline{OE}$ input.

The dual-port's chip enable ($\overline{CE}$) pin is driven indirectly by an address decoder consisting of an eight bit comparator 74ALS521 which compares the output of a 74LS377 register with addresses A_{12}-A_{19}. The 74LS377 is an I/O mapped register that allows the dual-port SRAM to be mapped into any 4K-byte region in the PC's main memory space. A PAL resident control register bit on the board allows the dual-port memory to be disabled, which is its state at power-up or reset.

The semaphore enable pin ($\overline{SEM}$) is driven by a 20L8 PAL which decodes addresses from the PC Bus. This decoder determines whether the host is accessing memory or I/O space via the $\overline{MEM}_R$, $\overline{MEM}_W$, $\overline{IO}_R$, and $\overline{IO}_W$ signals, and enables the semaphores during an I/O access if the proper address (A_0-A_9) is applied to the inputs of the PAL. The PAL also uses the $\overline{IO}_W$ and $\overline{MEM}_W$ signals to generate a R/$\overline{W}$ controlled write cycle, while using decoded addresses to drive the $\overline{CE}$ and $\overline{SEM}$ inputs.

Symbol	Parameter
WRITE CYCLE	
t_{WC}	Write Cycle Time
t_{EW}	Chip Enable to End-of-Write
t_{AW}	Address Valid to End-of-Write
t_{AS}	Address Set-up Time
t_{WP}	Write Pulse Width
t_{WR}	Write Recovery Time
t_{DW}	Data Valid to End-of-Write
t_{HZ}	Output High-Z Time
t_{DH}	Data Hold Time
t_{WZ}	Write Enable to Output in High-Z
t_{OW}	Output Active from End-of-Write

2694 tbl 01

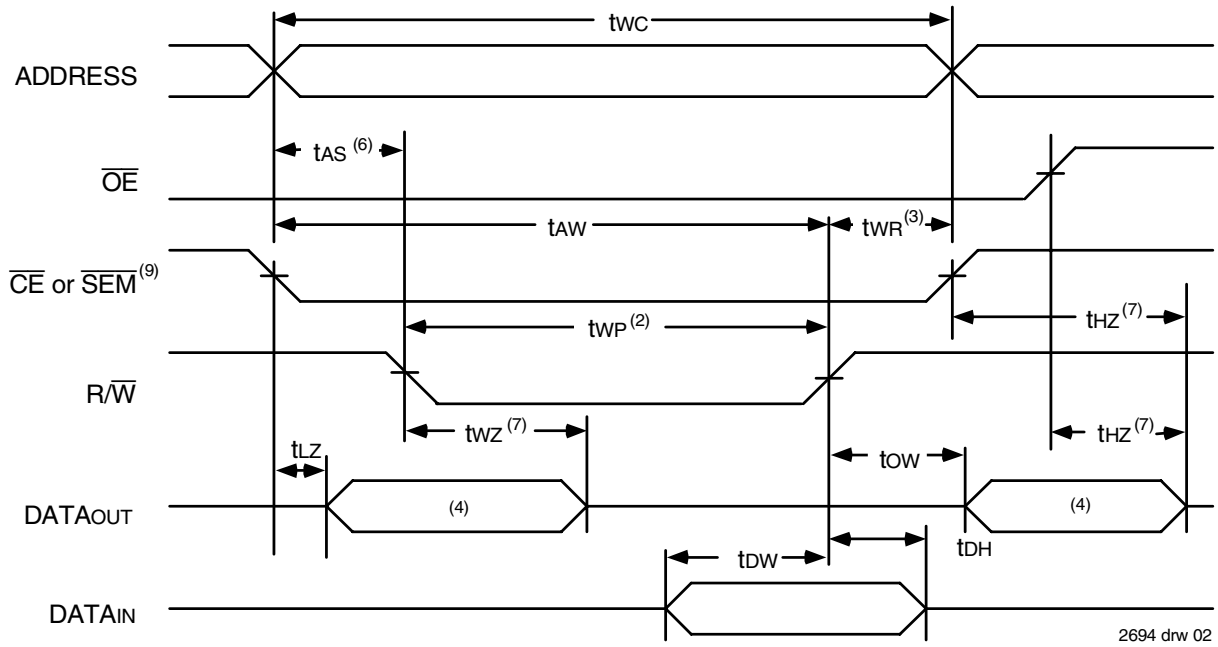

Figure 2. Timing Waveform of Write Cycle No. 1, R/$\overline{W}$ Controlled Timing(1,5,8).

NOTES:
1. R/$\overline{W}$ or $\overline{CE}$ must be HIGH during all address transitions.
2. A write occurs during the overlap (t_{EW} or t_{WP}) of either $\overline{CE}$ or $\overline{SEM}$ = V_{IL} and R/$\overline{W}$ = V_{IL}.
3. t_{WR} is measured from the earlier of $\overline{CE}$ or R/$\overline{W}$ going HIGH to the end-of-write cycle.
4. During this period, the I/O pins are in the output state, and input signals must not be applied.
5. If the $\overline{CE}$ LOW transition occurs simultaneously with or after the R/$\overline{W}$ LOW transition, the outputs remain in the High-impedance state.
6. Timing depends on which enable signal ($\overline{CE}$ or R/$\overline{W}$) is asserted last.
7. This parameter is guaranteed by device characterization, but is not production tested. Transition is measured ±500mV from steady state with the Output Test Load (Figure 2).
8. If $\overline{OE}$ is LOW during a R/$\overline{W}$ controlled write cycle, the write pulse width must be the larger of t_{WP} or (t_{WZ} + t_{DW}) to allow the I/O drivers to turn off data to be placed on the bus for the required t_{DW}. If $\overline{OE}$ is HIGH during an R/$\overline{W}$ controlled write cycle, this requirement does not apply and the write pulse can be as short as the specified t_{WP}.
9. To access SRAM, $\overline{CE}$ =V_{IL} and $\overline{SEM}$ = V_{IH}. To access semaphore, $\overline{CE}$ = V_{IH} and $\overline{SEM}$ = V_{IL}. Either condition must be valid for the entire t_{EW} time.

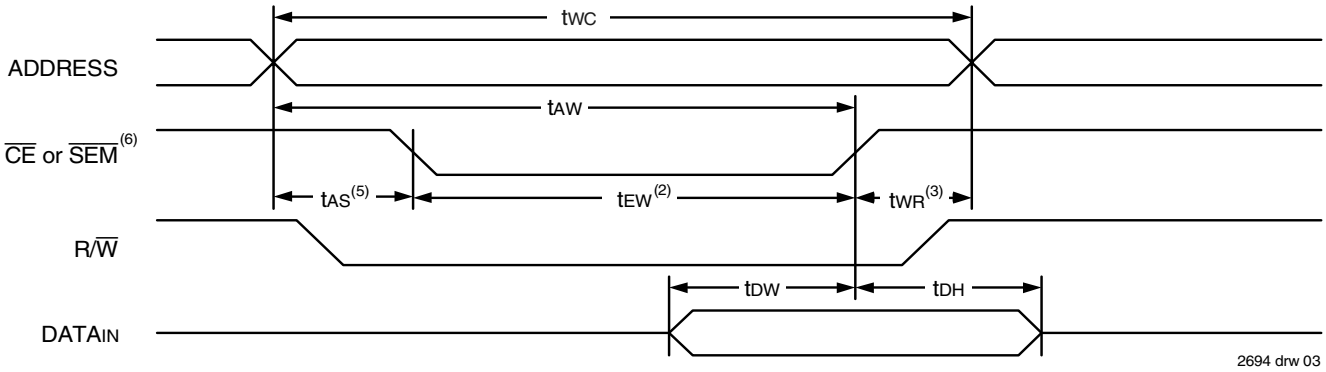

Figure 3. Timing Waveform of Write Cycle No. 2, $\overline{CE}$ Controlled timing(1,4).

NOTES:
1. R/$\overline{W}$ or $\overline{CE}$ must be HIGH during all address transitions.
2. A write occurs during the overlap (t_{EW} or t_{WP}) of either $\overline{CE}$ or $\overline{SEM}$ = V_{IL} and R/$\overline{W}$ = V_{IL}.
3. t_{WR} is measured from the earlier of $\overline{CE}$ or R/$\overline{W}$ going HIGH to the end-of-write cycle.
4. If the $\overline{CE}$ LOW transition occurs simultaneously with or after the R/$\overline{W}$ LOW transition, the outputs remain in the High-impedance state.
5. Timing depends on which enable signal ($\overline{CE}$ or R/$\overline{W}$) is asserted last.
6. To access SRAM, $\overline{CE}$ =V_{IL} and $\overline{SEM}$ = V_{IH}. To access semaphore, $\overline{CE}$ = V_{IH} and $\overline{SEM}$ = V_{IL}. Either condition must be valid for the entire t_{EW} time.

All data and address pins of the IDT71342 are isolated from the backplane with TTL buffers. A detail of the PC to dual-port interface is shown in Figure 4.

The reader should note that several considerations increased the complexity of this interface. If this design had involved a dedicated host processor rather than a general purpose PC, the need for buffering would probably have been drastically reduced. Had both the 4K byte SRAM and the semaphores been mapped into the memory space of the host, no ORing would have been required on the $\overline{MEM}_W$, $\overline{MEM}_R$, $\overline{IO}_W$, and $\overline{IO}_R$ signals. Finally, a very complex address decoder was imple-mented in this system to allow the IDT71342's RAM to be mapped anywhere within the PC's memory space. By using a more straightforward fixed-address scheme, logic complexity could be significantly reduced. It is conceivable that the entire interface including address decoding could have been handled with a single IC.

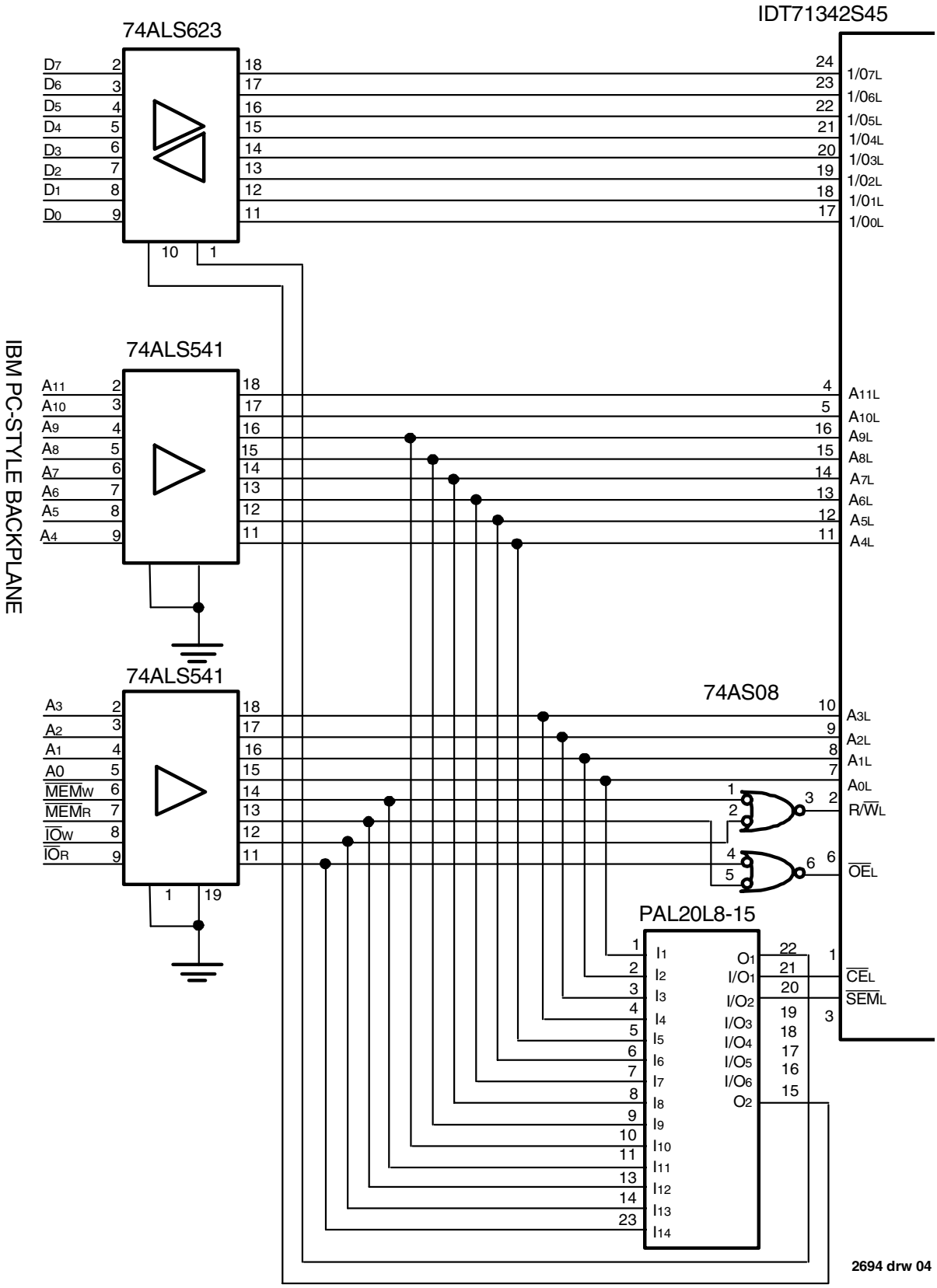

Figure 4. PC Bus to IDT71342 Dual-Port Interface (Left-Hand Side of Dual-Port).

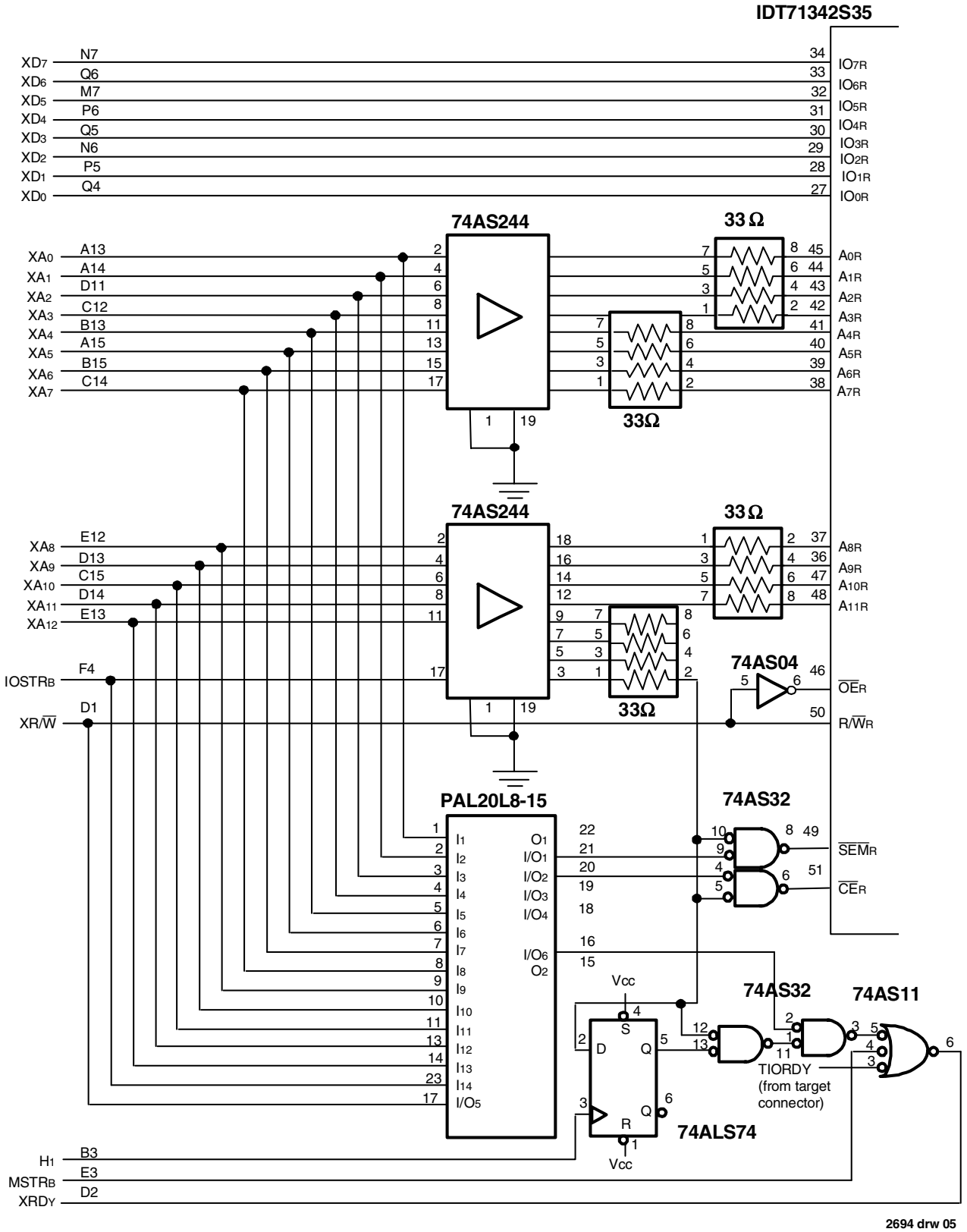

Figure 5. TMS320C30 to IDT71342 Dual-Port Interface (Right-Hand Side of Dual-Port).

The TMS320C30 Interface

The TMS320 interfaces to the dual-port SRAM through the I/O strobe on the expansion bus. The same bus is used to interface to a 16K x 32 static RAM via its memory strobe signal. These two strobes signify two different ranges on the DSP chip's internal address map. A detailed diagram of the TMS320 to IDT71342 interface is shown in Figure 5.

As in the PC bus interface, the address lines are buffered between the processor and the dual-port SRAM, however the light loading on the data bus removes the need for data buffering on this side. The only devices connected to the data pins are: the dual-port SRAM, the DSP chip, a static RAM, a status latch, and a transceiver. The address bus needed buffering since all eight 16K x 4 RAM chips, as well as the dual-port, a PAL, and an address buffer are attached to these pins.

The TMS320's expansion bus uses a strobe to activate an I/O cycle, and a level to distinguish read cycles from write cycles. In this design, the expansion read/write (XR/$\overline{W}$) output of the TMS320 is connected directly to the IDT71342 dual-port to drive the read/write (R/$\overline{W}$) input, and is simply inverted to drive the output enable ($\overline{OE}$) input. This inverter is not truly necessary, since the dual-port places its data outputs into a high-impedence state automatically upon the application of a write (LOW) level on the R/$\overline{W}$ input. The $\overline{OE}$ pin on this side could have been permanently tied active (grounded).

A 20L8 PAL is used to control the chip enable ($\overline{CE}$) input for this side of the dual-port SRAM. This signal is a decoding of the DSP's expansion bus address bits XA_0-XA_{12}. The PAL used in this interface had too few product terms to allow the combination of the I/O strobe with the decoded address, so the buffered I/O strobe (BIOSTRB) has been externally ANDed with the decoded address output from the PAL before being fed into the dual-port. The semaphore select is handled the same way, but a different address decoding is used from the same PAL, and the I/O strobe is ANDed through a different gate into the semaphore ($\overline{SEM}$) input of the dual-port. Both of these signals can be disabled by writing to the control register.

The TMS320C30 writes to the dual-port SRAM by implementing a $\overline{CE}$ controlled write cycle. The $\overline{CE}$ and $\overline{SEM}$ inputs are driven by two-input OR gates. One of the inputs of each of these gates represents a decoded address output from a 20L8 PAL, while the second input is driven by a buffered version of the I/O strobe. The only other qualifying input is the read/write (R/$\overline{W}$) input, which is directly driven by the expansion read/write (XR/$\overline{W}$) signal on the TMS320. When the DSP chip writes to the dual-port, the address and read/write signals are output first, followed by the I/O strobe. Since IOSTRB is used to gate the $\overline{CE}$ or $\overline{SEM}$ signal, the timing meets the criteria for a $\overline{CE}$ controlled write cycle.

The expansion ready (XRDY) input to the TMS320, which tells it that the expansion bus cycle is complete, is a combination of the decoded address range from the PAL and a clock delay from the TMS320's H1 (clock/2) output. This signal is required for systems using slower dual-port RAMs, but is not necessary in systems where faster dual-ports are used. If the system designer choses a 25ns or faster part for use in a 33MHz TMS320C30 system, the XRDY input can be generated immediately upon accessing the dual-port RAM.

The gating used here generates a single wait-state on any I/O strobe within the address range of the IDT71342. This logic could be removed if a faster IDT71342 were used. On an IOSTRB output from the TMS320, if the PAL decodes a dual-ported address, the strobe and decoded address are combined in the second of the two AND gates in Fig. 5. This AND gate's output is fed into the XRDY OR gate to extend the expansion bus cycle. On the next rising edge of H1, the IOSTRB is clocked into the flip flop. This flip flop's output is connected to the first AND gate and disables the IOSTRB from reaching the second AND gate. This in turn allows the XRDY input to the TMS320 to go active, and allows the cycle to end. A single wait-state more than compensates for the 45ns address access time of the dual-port used in this application. Other signals called target I/O ready (TIORDY) from the target connector, and the MSTRB signal from the DSP chip itself can also signal an expansion bus ready state. Since the MSTRB signal is used only to control accesses to the expansion bus' 16K x 32 zero wait-state RAM, it is ORed directly back to the XRDY input through the 74AS11 gate as shown.

Conclusion

The TMS320C30 Software Development Board shows the simplicity of designing an interface between a TMS320 DSP chip and the IBM PC bus using an IDT71342 dual-port RAM. The dual-port RAM serves to reduce component count, increase interprocessor communications throughput, and simplify design. Designers should be able to follow the example given here to profitably use dual-port SRAMs to handle communications in any similar dual processor system.

Appendix

Dual-Port Semaphores

Eight extra address locations in the IDT71342 4K x 8 dual-port RAM are dedicated to binary semaphore flags. These flags allow either the TMS320 or the host processor to claim a privilege over the other processor for functions defined by the programmer's software. As an example, the semaphore can be used by the PC to inhibit the TMS320C30 from accessing a portion of the dual-port SRAM, or some other shared resource.

The dual-port SRAM features a fast access time, and both ports are completely independent of each other. This means that the activity on the left port in no way slows the access time of the right port. Both ports are identical in function to standard static RAMs and can be read from, or written to, at the same time with the only possible conflict arising from the simultaneous writing of, or a simultaneous READ/WRITE of, a non-semaphore location. Semaphores are protected against such ambiguous situations and may be used by the system program to avoid any conflicts in the non-semaphore portion of the dual-port SRAM.

Multiple processor sytems like the TMS320C30 Software Development Board can benefit from a performance increase by using these semaphores, which provide a lockout mechanism without requiring complex programming.

Software handshaking between processors offers the maximum in system flexibility by permitting shared resources to be allocated in varying configurations. The IDT71342 does not use its semaphore flags to control any resources through hardware, thus allowing the programmer to determine each flag's meaning.

How the Semaphore Flags Work

The semaphore logic is a set of eight latches which are independent of the dual-port SRAM. These latches can be used to pass a flag, or token, from one processor to the other to indicate that a shared resource is in use. The semaphores provide a hardware assist for a use assignment called "Token Passing Allocation." In this method, the state of a semaphore latch is used as a token indicating that a shared resource is in use. If the TMS320 wants to use this resource, it requests the token by writing a zero into the latch. The TMS320 then verifies its success in writing the latch by reading it. If it was successful, it proceeds to assume control over the shared resource. If it was not successful in writing a zero into the latch, it determines that the PC had set the latch first, is in posession of the token, and is using the shared resource. The TMS320 can then either repeatedly inquire the status of the semaphore it requested, or it can remove its request for that semaphore by writing a one into its location. The TMS320 can then perform another task and occasionally attempt to gain control of the token via the set and test sequence. Once the PC has relinquished the token, the TMS320 can succeed in gaining control of the shared resource.

The semaphore flags are active low. A token is requested by writing a zero into a semaphore location, and is released when the same processor writes a one into that location.

The eight semaphore flags reside within the IDT71342 in a seperate memory space from the dual-port RAM. This address space is accessed by placing a low input on the $\overline{SEM}$ pin (which is used as a chip select for the semaphore flags), and using the other control pins (Address, $\overline{OE}$, and R/$\overline{W}$) as they would be used in accessing a standard static SRAM. Each of the flags has a unique address which can be accessed by either side through address pins A_0 - A_2. When accessing the semaphores, none of the other address pins has any effect.

When writing to a semaphore, only data pin D0 is used. If a low level is written into an unused semaphore location, that flag will be set to a zero on that side and a one on the other (see Table I). That location can now only be modified by the side showing the zero. When a one is written into the same location from the same side, the flag will be set to a one for both sides (unless a semaphore request from the other side is pending) and then can be written to by both sides. The fact that the side which is able to write a zero into a semaphore subsequently locks out writes from the other side is what makes semaphore flags useful in interprocessor communications. A zero written into the same location from the other side will be stored in the semaphore request latch for that side until the semaphore is freed by the first side.

When a semaphore flag is read, its value is spread into all data bits, so that a "set" flag reads as a one in all data bits and a flag containing a zero reads as all zeros. The read value is latched into one side's output register when that side's semaphore select ($\overline{SEM}$) and output enable ($\overline{OE}$) signals go active. This serves to disallow the semaphore from changing state in the middle of a read cycle due to a write cycle from the other side. Because of this latch, a repeated read of a semaphore in a test loop must cause either signal ($\overline{SEM}$ or $\overline{OE}$) to go inactive, or the output will never change. This is not a concern in the TMS320C30 Software Development Board, since either bus' accesses to other memory locations between semaphore accesses inactivate both of these signals for a relatively long period no matter how tight of a loop is used to interrogate the device.

FUNCTION	PC BUS D0-D7 LEFT	TM320 D0-D7 RIGHT	STATUS
No action	1	1	Semaphore free
PC writes “0” to semaphore	0	1	PC has semaphore token
TMS320 writes “0” to semaphore	0	1	No change. TMS320 has no write access to semaphore
PC writes “1” to semaphore	1	0	TMS320 obtains semaphore token
PC writes “0” to semaphore	1	0	No change. PC has no write access to semaphore.
TMS320 writes “1” to semaphore	0	1	PC obtains semaphore token
PC writes “1” to semaphore	1	1	Semaphore free
TMS320 writes “0” to semaphore	1	0	TMS320 has semaphore token
TMS320 writes “1” to semaphore	1	1	Semaphore free
PC writes “0” to semaphore	0	1	PC has semaphore token
PC writes “1” to semaphore	1	1	Semaphore free

2694 tbl 02

Table 1. Example Semaphore Procurement Sequence

A sequence of WRITE/READ must be used to acquire a semaphore in order to guarantee that no system level contention will occur. A processor requests access to shared resources by attempting to write a zero into a semaphore location. If the semaphore is already in use, the semaphore request latch will contain a zero, yet the semaphore flag will appear as a one, a fact which the processor will verify by the subsequent read (see Table I). As an example, assume the PC writes a zero to the left port at a free semaphore location. On a subsequent read, the PC will verify that it has written successfully to that location and will assume control over the resource in question. Meanwhile, if the TMS320 attempts to write a zero to the same semaphore flag, it will fail, as will be verified by the fact that it will read a one from that semaphore during a subsequent read cycle. Had a sequence of READ/WRITE been used instead, contention problems could have occurred during the gap between the read and write cycles.

It is important to note that a failed semaphore request must be followed either by repeated reads, or by writing a one into the same location to remove the semaphore request. The reason for this is easily understood by looking at the simple logic diagram of a semaphore flag shown in Figure 6. Two semaphore request latches feed into a semaphore flag. Whichever latch is the first to present a zero to the semaphore flag will force its side of the semaphore flag low, and the other side high. This condition will continue until a one is written into the same semaphore request latch. Should the other side’s semaphore request latch have been written to a zero in the meantime, the semaphore flag will flip over to the other side as soon as a one is written into the first side’s request latch. The second side’s flag will now stay low until its semaphore request latch is written with a one. From this it is easy to understand that, if a semaphore is requested and the processor which requested it no longer needs the resource, the entire system could hang up until a one is written into that semaphore request latch.

The critical case of semaphore timing is when both sides request a single token by attempting to write a zero into it at the same time. The semaphore logic is specially designed to resolve this problem. If simultaneous requests are made, the logic guarantees that only one side receives the token. If one side is earlier than the other in making the request, the first side to make the request will receive the token. If both requests happen at the same time, the assignment will be arbitrarily made to one side or the other.

One caution that should be noted when using semaphores is that semaphores alone do not guarantee that access to a resource is secure. As with any powerful programming technique, if semaphores are misused or misinterpreted a software error can easily happen. Code integrity is of the utmost importance when semaphores are used instead of hardware handshaking.

Initialization of the semaphores is not automatic and must be handled via the initialization program at power-up. Since any semaphore which is written to a zero must be reset to a one, both the TMS320 and the PC must write a one into all semaphore locations at initialization to assure that the semaphores will be free when needed.

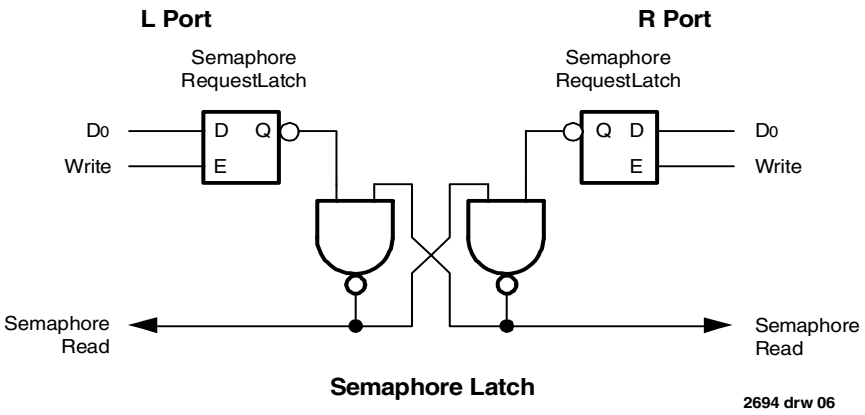

Figure 6. IDT71342 Semaphore Logic

Using Semaphores - Some Examples

Perhaps the simplest application of semaphores is their application as resource markers for the IDT71342's dual-port SRAM. Say the 4K x 8 SRAM was to be divided into two 2K x 8 blocks, which were to be dedicated at any one time to servicing either the PC or the TMS320. Semaphore 0 could be used to indicate the side which would control the lower section of memory, and Semaphore 1 could be defined as the indicator for the upper section of memory.

To take a resource, in this example the lower 2K of dual-port RAM, the PC could write then read a zero into Semaphore 0. If this task was successfully completed (a zero was read back, rather than a one), the PC would assume control of the lower 2K. Meanwhile, the TMS320 might attempt to perform the same function. Since the TMS320 was attempting to gain control of the resource after the PC, it would read back a one in response to the zero it had attempted to write into Semaphore 0. At this point, the TMS320's software could choose to try and gain control of the second 2K section by writing, then reading a zero into Semaphore 1. If it succeeded in gaining control, it would lock out the PC.

Once the PC was finished with its task, it would write a one to Semaphore 0, then may try to gain access to Semaphore 1. If Semaphore 1 was still occupied by the TMS320, the PC could remove its semaphore request and perform other tasks until it was able to write, then read a zero into Semaphore 1. If the TMS320 performs a similar task with Semaphore 0, this protocol would allow the two processors to swap 2K blocks of dual-port RAM with each other.

The blocks do not have to be any particular size and could even be of variable length, depending upon the complexity of the software using the semaphore flags. All eight semaphores could be used to divide the dual-port RAM or other shared resources into eight parts.

Semaphores are a useful form of arbitration in real-time DSP applications, when the PC must be locked out of a section of memory during a transfer, and the TMS320 cannot tolerate any wait states. With the use of semaphores, once the two processors had determined which memory area was "off limits" to the PC, both the PC and the TMS320 could access their assigned portions of memory continuously without any wait states. Both processors can access their assigned RAM segments at full speed.

Another application of semaphores is in the area of complex data structures. In this case, block arbitration is very important to the maintenence of data integrity. For this application one processor may be responsible for building and updating a data structure, which the other processor then reads and interprets. If the interpreting processor reads an incomplete data structure, a major error condition may exist. Therefore, some sort of arbitration must be used between the TMS320 and the PC. Software semaphores are a perfect fit. The building processor uses the semaphore to arbitrate for the block and to lock it once that processor is able to acquire the semaphore flag. This processor then is able to go in and update the data structure. When the update is completed, the semaphore and the corresponding data structure block are released. The interpreting processor then acquires the semaphore which allows it to come back and read the complete data structure, thereby guaranteeing consistency.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.