BE CAREFUL WHEN WRITING SCSI-2 WIDE BUS SOFTWARE

WHILE THE PROPOSED SCSI-2 WIDE BUS HAS ADVANTAGES, IT ALSO CHALLENGES SOFTWARE DESIGNERS.

GEORGE HAHN
Adaptec Inc., 691 S. Milpitas Blvd., Milpitas, CA 95035; (408) 945-8600.
MARK S. GORDON
Digital Finesse Inc., 355 W. Olive Ave., Ste. 105, Sunnyvale, CA 94086; (408) 737-9156.

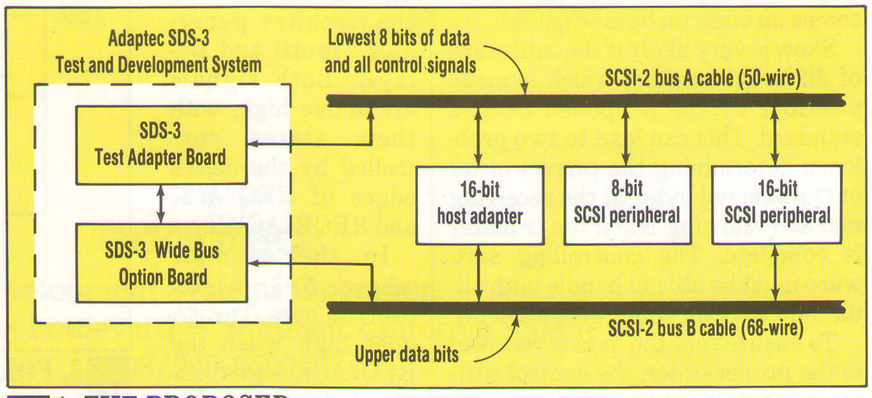
The proposed SCSI-2 ANSI standard adds numerous features to the original SCSI specification. Among the more useful is a wider data bus called Wide Bus, which transfers 8-, 16-, or 32-bit data to and from SCSI peripherals. The advantages of the Wide Bus come with some problems, though, including accounting for the skew between the two cables used for data byte transfers, proper ordering of bytes in the buffer containing sector data, proper negotiation for bus width, determining when a transfer is complete, and handling parity. Designers must solve these problems in the algorithms that manage Wide Bus data transfers at the controller level.

The proposed SCSI-2 Wide Bus consists of two cables *(Fig. 1)*. Many of the software requirements of SCSI derive from the fact that data transfers on the cables need not be synchronized, creating a possible skew between them. The A cable is identical to SCSI-1's single cable, so SCSI-1 and -2 equipment can coexist in a system. The A cable's 50 lines carry a 9-bit data bus (8 data bits and 1 parity bit), nine control signals, a resistor-terminator power line, and ground lines.

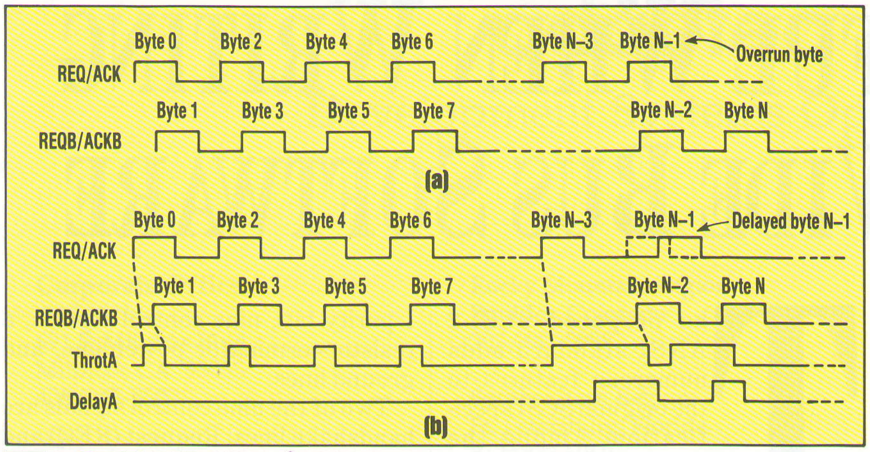
Two of the nine control signals are handshake lines: Request (REQ) and Acknowledge (ACK). A target asserts REQ when it's ready to receive or transmit one data byte; an initiator asserts ACK to indicate that a 1-byte data transfer is complete. Note that the target always controls the data transfer process. It starts a transfer by asserting REQ and awaits an ACK from the initiator prior to commencing another single-byte transaction.

Containing 68 pins, the B cable carries 24 data and 3 parity lines, REQB and ACKB control signals, its own resistor-terminator power signals, and ground lines. The data lines are organized as three sets of 8, each with its own parity bit. Separate REQB and ACKB handshake lines are defined because the B cable's length may be different than the A cable. With a set of handshake lines on each cable, the skew between handshake signals and data lines on any one cable is minimized. This is necessary if SCSI-1 and -2 equipment are going to coexist.

In a SCSI-2 system, each target/


Adaptec SDS-3 Test and Development System
SDS-3 Test Adapter Board
SDS-3 Wide Bus Option Board
Lowest 8 bits of data and all control signals
SCSI-2 bus A cable (50-wire)
16-bit host adapter
8-bit SCSI peripheral
16-bit SCSI peripheral
SCSI-2 bus B cable (68-wire)
Upper data bits

1. THE PROPOSED SCSI-2 wide bus standard uses two cables: an A cable that carries the first 8 bits of data, and a B cable that can carry 24 bits for a total bus width of 32 bits.



2. BECAUSE SCSI-2's CABLES need not be synchronized, data on one cable may lead or lag data transmitted on the other cable, a condition known as skew (a). Using a throttling technique ensures that the data bytes will be received in the proper order (b).

initiator pair must negotiate for the width of the data bus. The negotiation process that chooses a particular data bus width is a factor software designers must consider.

Only the A cable is used for an 8-bit bus width. For a 16-bit bus width, even bytes (0, 2, 4, etc.) move on the A cable as odd bytes (1, 3, 5, etc.) pass along the B cable. For 32-bit data, bytes 0, 4, and so on are transferred on the A cable, while bytes 1, 2, 3, 5, 6, 7, and so on are sent on the B cable. Because the B cable contains 24 data and 3 parity signal lines, 2- or 3-byte transfers are made on this cable in one transaction (one REQB and one ACKB).

Because the two cables need not be synchronized, data bytes transferred on one cable may lead or lag those moving on the other cable—a condition known as skew. For instance, if byte N–1 begins transferring before byte N–2, byte N–1 becomes an overrun byte *(Fig. 2a)*.

Skew is very likely if the cables are of different lengths, which is made possible by the proposed SCSI-2 standard. This can lead to two problems: determining the correct order of transferred bytes at the receiving end and knowing when the transfer is complete. The controlling software must be able to handle both situations, making debugging difficult.

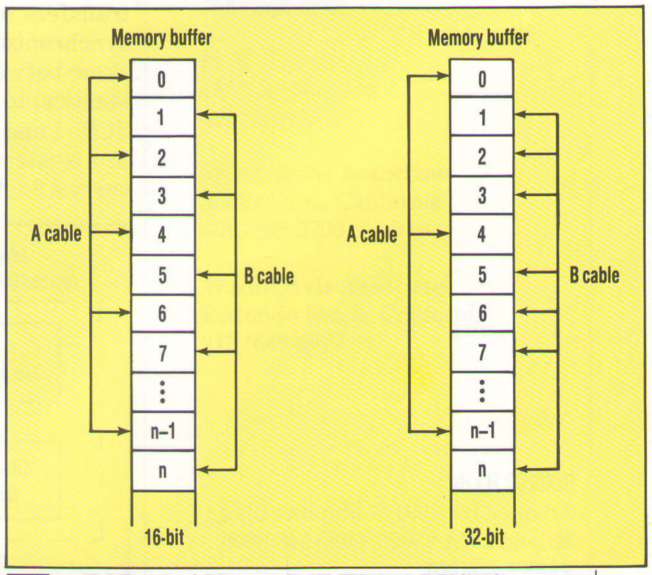
To ensure that the data is received in the proper order, the control software can prevent skew by appropriately controlling the REQ/REQB (target) or ACK/ACKB (initiator) handshake signals. This process is called throttling. Alternatively, the software can manage the problem by skew counting, which is to maintain a running tally of the skew. Throttling, which facilitates debugging, is an important feature of test equipment. The technique can also be designed into the hardware and firmware for a controller or other SCSI device, but the reduced data throughput caused by throttling is generally unacceptable.

Throttling requires specialized hardware that keeps track of which cable transferred the last data byte (ELECTRONIC DESIGN, *Sept. 14, p. 73)*. The hardware generates ThrotA and DelayA signals if cable A leads cable B by more that one data byte. If cable B leads cable A, the circuitry generates ThrotB and DelayB. Both signals are active high, with their states controlled by the active edges of REQ/ACK and REQB/ACKB.

In the example where N–1 is the overrun byte, ThrotA goes high when the REQ/ACK signal associated with byte N–3 goes high *(Fig. 2b)*. DelayA goes high when REQ/ACK goes high again. DelayA gates the REQ/ACK signal onto the SCSI bus, preventing REQ or ACK from going active until the next active edge of REQB/ACKB. This next active edge of REQB/ACKB causes ThrotA to go low, forcing DelayA low.

By preventing the occurrence of either REQ or ACK, the throttling circuitry prevents the occurrence of the other. Consequently, the handshaking used in both the target and initiator roles enables designers to throttle a data transfer by suppressing one of the two handshaking signals—the other end of the link suppresses the other signal. Thus, throttling synchronizes the two cables.

If the need for maximum performance eliminates throttling as an alternative, the control software must track the number of bytes transferred on each cable, as well as each byte's arrival time. The system can then derive the skew count from this information, which is the difference between the number of transfers made on the A cable (one REQ/ACK sequence per transfer) and the number made on the B cable (one REQB/ACKB sequence per transfer). As a measure of the amount by which one cable leads or lags the other, skew count can help in the proper ordering of transferred bytes and in determin-



3. FOR 16- AND 32-BIT WIDE BUSES, the location in which data bytes are stored in the buffer depends on which cable carried the data and how wide the bus is.

ing when a transfer is complete.

The skew count, however, isn't necessarily the same as the number of bytes sent on the A cable, less the number sent on the B cable. For a 32-bit bus, each REQ/ACK sequence accompanies a 1-byte transfer, while every REQB/ACKB sequence is associated with a 3-byte transfer. This discussion considers the case of a 16-bit Wide Bus, in which the number of bytes sent on the A cable less the number sent on the B cable is identical to the skew count.

To improve performance, SCSI systems often buffer data bytes at the receiving end of the SCSI cable. For reads, buffering typically takes place at the host end of the cable; for writes, it's generally at the drive end. The algorithm that controls the byte order must also account for buffering.

For an 8-bit bus, all of the bytes come from the A cable and are stored in the order received. For 16- and 32-bit busses, the offset at which incoming bytes are stored from the beginning of the buffer depends on the cable from which a byte came and the width of the SCSI data bus *(Fig. 3)*. Thus, the algorithm must consider each byte's source as well as the relative skew of the cables.

A Warnier-Orr diagram highlights the prominent features of an example algorithm *(Fig. 4)*. This is for an asynchronous 16-bit bus, whereby even bytes are transferred on the A cable and odd bytes are transferred on the B cable.

In the diagram's upper left corner is the index, or the offset within the buffer at which the next byte from each cable will be stored. The A index is initialized to zero, and the B index to 1. At the center of the leftmost brace is a loop that constantly checks for Command Complete (a second algorithm determines command completion). The loop executes as many times as necessary, therefore the C in the parentheses is a variable.

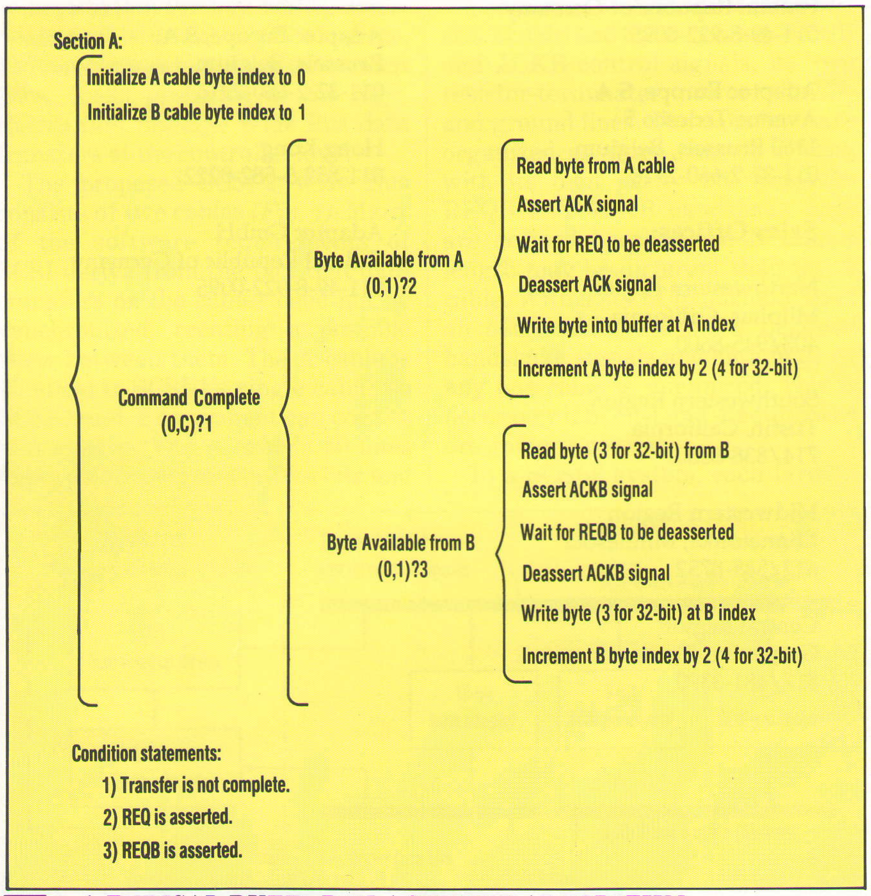
Within the command completion loop are two "if" statements: Byte Available from A and Byte Available from B. Based on conditions specified at the bottom of the diagram, the "if" statements look for REQ asserted and REQB asserted, respectively. The statements are loops that execute 0 or 1 time, depending on the condition specified.

When the condition in one of the "if" statements is satisfied, the algorithm executes a series of tasks, which are in the rightmost brace of the Warnier-Orr diagram. These tasks involve the following: reading the incoming byte, performing the REQ/ACK handshake protocol, writing the byte to the buffer at the appropriate index, and incrementing the index value.

The algorithm loops continuously through the tests for Command Complete, Byte Available from A, and Byte Available from B, in that order, until the condition for Command Complete is met. Thus, if Byte Available from A is true, the algorithm executes the statements associated with this "if" statement, then goes to Byte Available from B. A true result there causes the execution of the corresponding statements and then an evaluation of Command Complete.

This sample algorithm is for an initiator. For instance, the algorithm could be used when a host is placing bytes received from a disk controller or other SCSI target device into a buffer. The algorithm for a target is similar: REQ/REQB and ACK/ACKB are transposed and the Command Complete loop is replaced by one that determines if all of the bytes were transferred.

The next problem is how to determine when a data transfer is complete. One obvious value to check is skew count. Because it's the difference between the number of transfers made on the A cable and those on the B cable, the skew count will be zero when a transfer is complete. It may also be zero, however, when a transfer is partially complete, as when an equal number of transfers have occurred on both cables. This


Section A:
Initialize A cable byte index to 0
Initialize B cable byte index to 1
Command Complete (0,C)?1
Byte Available from A (0,1)?2
Read byte from A cable
Assert ACK signal
Wait for REQ to be deasserted
Deassert ACK signal
Write byte into buffer at A index
Increment A byte index by 2 (4 for 32-bit)
Byte Available from B (0,1)?3
Read byte (3 for 32-bit) from B
Assert ACKB signal
Wait for REQB to be deasserted
Deassert ACKB signal
Write byte (3 for 32-bit) at B index
Increment B byte index by 2 (4 for 32-bit)
Condition statements:
1) Transfer is not complete.
2) REQ is asserted.
3) REQB is asserted.

4. A TYPICAL BUFFER PLACEMENT ALGORITHM includes condition statements to determine when REQ or REQB is asserted.

may happen repeatedly if a transfer proceeds in lock step. Thus, a skew count of zero is a necessary but insufficient condition for checking command completion.

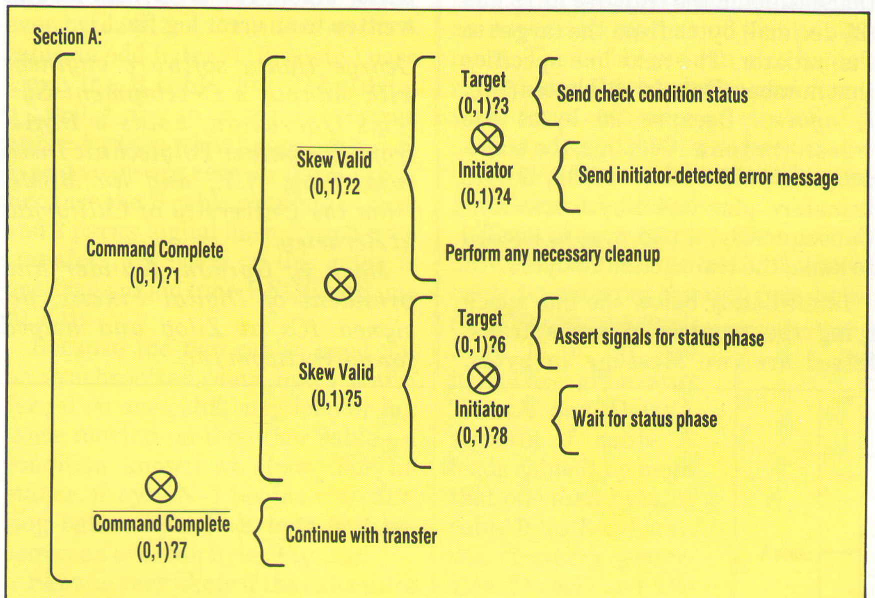
Another measure of a completed SCSI transfer is the Command Complete message that's sent from the target to the initiator. Consider that the least significant byte of all messages is always transferred on the A cable. In the case of a Command Complete message, which has a value of zero, there are no further message bytes. In messages that do have additional bytes, the first byte must be nonzero.

The SCSI specification prohibits the sending of a Command Complete message until all bytes are sent on both cables. This restriction prevents the Command Complete message on the A cable from leading the last data bytes moving across the B cable. If the two cables are of vastly different lengths, however, the circuitry at the receiving end might accept the Command Complete message before processing the last data bytes transferred on the B cable.

Thus, the algorithm that determines when a data transfer is done must contain two conditions:

1. A Command Complete message must be received on the A cable, indicating that all of the bytes were transferred on this cable.
2. The skew count must be zero, indicating that all of the bytes were transferred on the B cable.

In the Warnier-Orr diagram of a general algorithm for assessing whether command execution is finished, the leftmost brace contains the Command Complete message *(Fig. 5)*. For an initiator, the message is received; for a target, the message is sent. After the Command Complete message is received, the algorithm checks for a zero skew count.



Condition statements:

1) Hardware detects completion (implementation-dependent).
2) Skew count is not equal to 0 (REQs ≠ REQBs).
3) Device detecting bad skew is a target.
4) Device detecting bad skew is initiator.
5) Skew count equals 0 (REQs = REQBs).
6) Device checking skew count is target.
7) Hardware does not detect completion (implementation-dependent).
8) Device checking skew count is initiator.

5. THE ALGORITHM DETERMINING when a data transfer is complete must ensure that a Command Complete message was received on the A cable and that the skew count is zero.

If both conditions are met, a test for status phase is made.

If the skew count isn't zero after a Command Complete message is received or sent, an error condition must be flagged. In the case of a target, this takes the form of a Check Condition Status statement; for an initiator, it emerges as an Initiator-Detected Error Message.

Another piece of software that's peculiar to the SCSI-2 Wide Bus is the negotiation sequence. Each target/initiator pair must negotiate the width of the data bus they will use for their transactions prior to their first data transfer. In the absence of this negotiation, an 8-bit width is used. This default feature allows the simultaneous use of devices with different bus widths.

The negotiation is simple: The initiator requests a width using a Message Out sequence, and the target responds with a Message In sequence specifying that width or a smaller one. There isn't any further negotiation.

An example of such a negotiation is a state log display generated by the Adaptec SDS-3 SCSI test and development system *(Fig. 6)*. The first Message Out byte (C0) is an Identify message sent from the initiator to the target. This is signalled by the highest order bit, bit 7, which is set to a one. Also, bit 6 of this message byte specifies whether the initiator has granted the target the right to disconnect. In this example, the bit is a 1, indicating that the target may disconnect. This byte also contains bits that make it possible to select either a specific logical unit number or a target routine number. The proposed SCSI-2 standard gives further details.

The next Message Out byte (01) indicates the beginning of a message containing more than one byte. Afterwards comes a 02 Message Out byte, which specifies that two more message bytes will follow. The first of these, a 03, tells the target that this is a Wide Bus negotiation sequence. The second, a 02, requests a bus width of 32 bits.

The next four Message In bytes in the state log are the target's response. The first byte, a 01, specifies that more Message In bytes will follow. The second, a 02, reports the number of Message In bytes that will be sent. Next comes a 03, which tells the initiator that this is a Wide Bus negotiation sequence. The last Message In byte, a 01, is the target's reply. The message specifies a width of 16 bits, which is less than the requested width of 32 bits. Because the smaller of the two widths is used, this negotiation sequence will result in a bus width of 16 bits.

A further consideration is the data padding that may be needed by a SCSI-2 Wide Bus. The SCSI standard states that the number of bytes in every data transfer must be an integer multiple of the bus width in bytes. As a result, a 16-bit bus must transmit an even number of bytes, and the bytes transferred by a 32-bit bus must be a multiple of four. If the data doesn't meet this restriction, the transmitter must pad the data.

To ensure data integrity when pad bytes are needed, the transmitter must also send an Ignore Wide Residue message. This message specifies the number of undefined pad bytes included in the final bus transaction (the last set of REQ/ACK and REQB/ACKB handshakes).

In the example code, the Command Out message with an op code of 03 is a Request Sense command that results in the transfer of 19 hex (25 decimal) bytes from the target to the initiator. The next line specifies that number: Data In 19H bytes *(Fig. 6, again)*. Because 25 bytes are transferred on a 16-bit bus, the transaction would consist of 12, 2-byte transfers plus one 1-byte transfer. Consequently, a pad byte is needed to make the transaction 26 bytes.

Immediately below the line specifying the number of bytes transferred are two Message In bytes. The first byte, a 24, tells the initiator that it must ignore a wide residue. The second, a 01, indicates the number of bytes to be ignored—one. This corresponds to bits 8 through 15, the second byte of the last transfer.

The proposed SCSI-2 standard also defines a parity bit for each data byte. As a result, the A cable has one parity bit, and the B cable has three. Even undefined bytes have valid parity. Therefore, in the above example of a wide residue, the pad byte sent from the target to the initiator has valid parity to ensure that all bytes transferred have valid parity.

Handling of parity errors is implementation-dependent. Retries up to a specified number is one type of handling method. If this doesn't produce a successful transfer, an error message is supplied from one level of software to another. The result may be a display message flagging the parity error. The error may also be written to an error log file.□

STATE LOG BUFFER DISPLAY

```
00020.30760 Bus Free Detected                                      046
00045.88097   Arbitration as 07                         45.88108   045
00045.88122     Selection ids = (1001 0000b)            45.88143   044
00045.88243       Message out C0                                   043
00045.88254       Message out 01                                   042
00045.88265       Message out 02                                   041
00045.88271       Message out 03                                   040
00045.88276       Message out 02                                   03F
00045.88332       Message out 01                                   03E
00045.88366       Message out 02                                   03D
00045.88380       Message out 03                                   03C
00045.88400       Message out 01                                   03B
00045.88501          Command out 03 00 00 00 19 00      45.88534   03A
00045.89881             Data in     19H byte (s)                   039
00045.89914       Message in 24                                    038
00045.89950       Message in 01                                    037
```

6. THIS STATE LOG DISPLAY shows a typical bus-width negotiation sequence for a SCSI-2 Wide Bus.

George Hahn, software engineer with Adaptec's Development Systems Operation, holds a BSAE from Rensselaer Polytechnic Institute, Troy, N.Y., and an MSME from the University of California at Berkeley.

Mark S. Gordon, founder and president of Digital Finesse, designed ICs at Zilog and helped found Verticom Inc.

adaptec
Development Systems Operation
691 s. milpitas boulevard • milpitas, california 95035
408/945-8600