Accredited Standards Committee
X3, Information Processing Systems

Doc:	X3T10.1/96a127r2
Date:	June 19, 1996
Project:	X3T10.1 / 1147
Ref Doc.:	SSA-TL2
Reply to:	Ian Judd

To: X3T10.1 Membership
From: Ian Judd

Subject: Device Services Interface

Preface

This document defines the interface between the SSA disk drives and the controller in an SES-compliant enclosure.

This is a working document for discussion. It represents current thinking within IBM but it should not be regarded as a final definition.

1.0 Introduction

DSI is a proposed low-speed serial interface for internal use within a storage device enclosure. It links each storage device to a central controller which provides access to VPD, power/cooling status and slot identification. The host can then retrieve this information via the external device interface, i.e. using the SSA loop and the SCSI-3 Enclosure Services (SES) command set.

SES is a development of the former SAF-TE proposal from Intel and Conner. It uses the SCSI Send Diagnostic and Receive Diagnostic Results commands to access the data. Since SSA-S2P is essentially a mapping of SCSI-2 the SES command set can also be supported on SSA. For the definition of the SES command set see the SCSI-3 Enclosure Services specification in document X3T10/95-324.

The storage devices merely pass through the SES commands from the SSA loop to the DSI port for interpretation by the central controller. The operation of the enclosure does not depend on the controller, i.e. the controller is not a single point of failure.

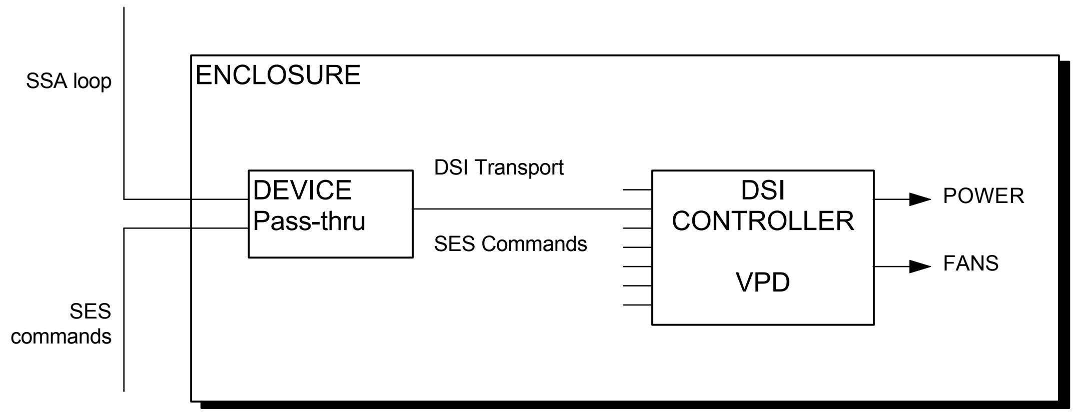

Figure 1. DSI Concept

DSI requires minimal hardware in the storage device - just 2 bi-directional I/O pins. The remaining device function is virtually time-independent and it can be implemented entirely in firmware.

To ensure inter-operability with a wide range of SSA devices, DSI should be proposed as an ANSI standard, i.e. part of SSA version 2.

1.1 Electrical interface

Each device is radially connected to the controller via a dedicated line, DSI_A_n. It is also connected to the controller and all other devices via a common line, DSI_B. These I/O pins are already available in the current SSA-PH1 specification as 'Programmable 1' and 'Programmable 2' respectively. (Pins 9 and 10 in the options bay of the unitised device connector.)

The radial DSI_A_n lines support arbitration between multiple devices and they also allow the controller to assign a unique slot number to each device. The common DSI_B line supports a full hand-shake on every bit which avoids the need to recover a clock from the data.

The controller provides a 1K resistive pull-up to +3.3V for each line. CMOS 3.3 volt levels are used for signalling.

Both lines are asserted at different times by the controller and the device. A line is asserted by actively pulling it down; a line is negated by merely releasing it. This avoids driver contention, even during error conditions.

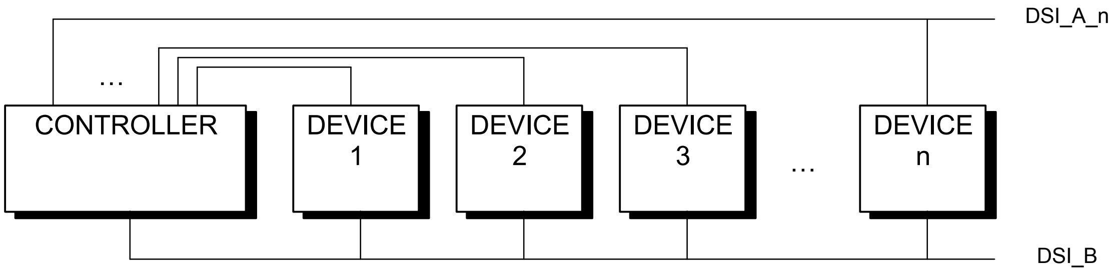

Figure 2. Electrical Interface

A 4 mA 3-state push-pull driver (as currently defined in SSA-PH1) can be used by connecting the data input to logic zero and activating the enable input to assert the signal. An open-drain driver could also be used.

1.2 Transport layer

1.2.1 Transaction phases

Each device has a master-slave relationship with the controller. DSI Transactions are only initiated by a device. Each Transaction is divided into 3 Phases as follows:

1. Arbitration. The device repeatedly sends a Request to the controller until it is acknowledged.
2. Command. The device transmits a single Command packet to the controller which may also contain write data.
3. Response. The controller returns a corresponding Response packet to the device which may also contain read data.

Successful arbitration establishes a connection between the device and the controller. The connection normally lasts until the end of the Response phase, unless it is terminated earlier by an error. If the device has another Transaction to perform then it must establish a new connection by arbitrating again.

1.2.2 Idle state

In the Idle state the DSI signals are driven as follows:

DSI_A_n	Released by the device and the controller.
DSI_B	Asserted by the controller and released by the device.

When the controller is idle it periodically monitors the DSI_A_n signals for an Arbitration Request from any device.

When a device is idle it periodically monitors the DSI signals for an Alert from the controller.

1.2.3 Arbitration phase

When a device wishes to perform a DSI transaction it must first arbitrate with the controller. A device may begin arbitration at any time, except when it already has a connection with the controller.

The arbitration protocol is as follows:

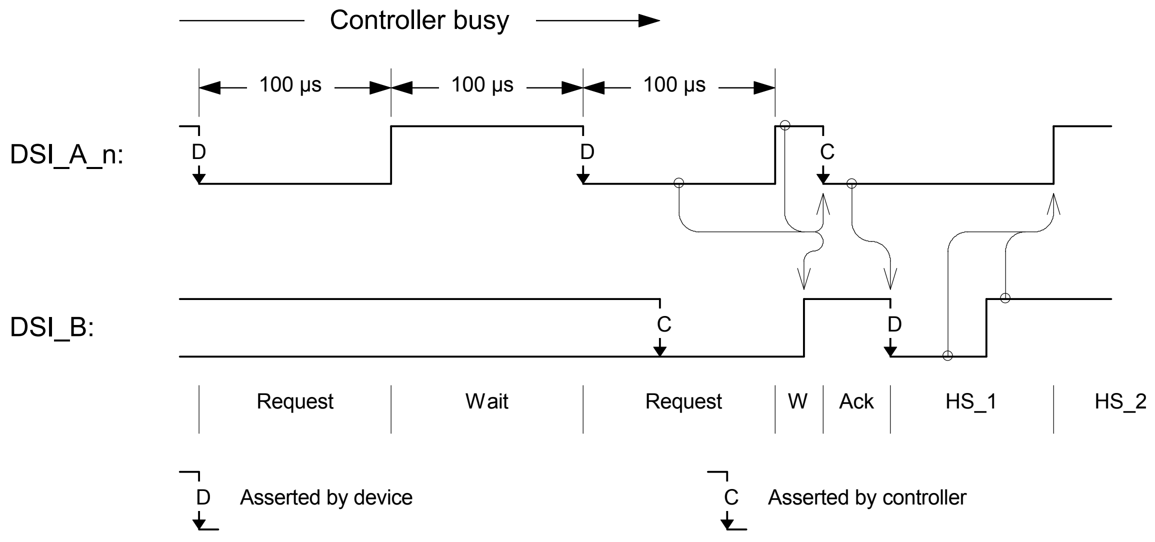

Figure 3. Arbitration

1. Request. The device asserts its DSI_A_n line for a fixed period of approximately 100 µs and then releases it.
2. Wait. The device starts a 100 µs time-out and waits until the controller asserts DSI_A_n. If the timer expires the device returns to the Request phase above.
3. Acknowledge. When the controller is ready to service the device it waits until the device releases DSI_A_n. Then the controller releases DSI_B and asserts DSI_A_n.
4. Handshake_1. When the device detects that the controller has asserted DSI_A_n with DSI_B released it asserts DSI_B for approximately 100 µs. Then the device releases DSI_B and waits for the controller to release DSI_A_n.
5. Handshake_2. When the controller detects that the device has asserted and released DSI_B it releases DSI_A_n.

When the device detects that the controller has released DSI_A_n it has successfully completed arbitration. The device must now transmit a Command packet to the controller using the data transfer protocol.

Note that the device is not required to respond to an asynchronous state change within any particular period of time. However the controller must be able to detect the 100 µs Request and Handshake_1 pulses when it is idle. If the controller is busy with another device then it can simply ignore a new Request pulse.

1.2.4 Data transfer

Data transfer does not depend on the speed of either node; there is a full hand shake on every bit. It may be implemented entirely in firmware by polling the lines periodically. No interrupts, hardware shift registers or clock-recovery circuits are required.

The sequence to transmit a single bit (in this case 1b) is:

1. Wait. The transmitter waits until DSI_A_n and DSI_B are released. The receiver waits for the transmitter to assert DSI_A_n or DSI_B.
2. Transmit_1. The transmitter asserts DSI_B and waits for the receiver to assert DSI_A_n.
3. Handshake_3. The receiver samples DSI_B asserted and records 1b; it then asserts DSI_A_n and waits for the transmitter to release DSI_B.
4. Handshake_4. The transmitter releases DSI_B and waits for the receiver to release DSI_A_n.
5. Wait. The receiver releases DSI_A_n.

To transmit 0b, the roles of DSI_A_n and DSI_B are simply reversed.

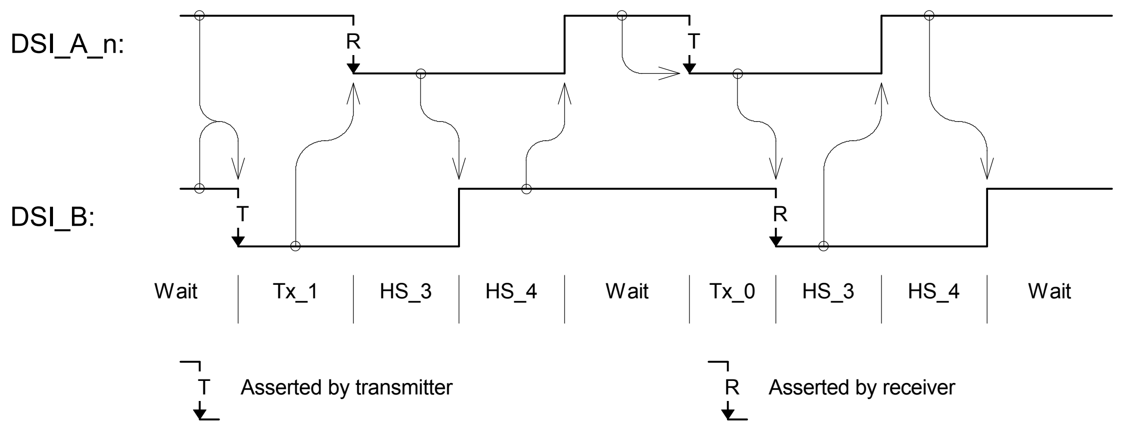

Figure 4. Data Transfer

A byte is transmitted as 8 bits. The most-significant bit (i.e. bit 7) is transmitted first. There is no need for any framing bits or character encoding.

1.2.5 Alerts

The controller may wish to alert a device of a change in the enclosure status, e.g. the failure of a redundant power supply. This might cause the device to send an Async_alert SMS to the Master node on the SSA Web.

The Controller signals an Alert to a particular device by asserting the corresponding DSI_A_n signal for approximately 1 ms when the interface is idle. The Controller must only signal an Alert to a device which has previously identified itself as supporting DSI.

A device detects an Alert if DSI_A_n is asserted by the controller while DSI_B is also asserted. When a device detects an Alert it should wait until the controller has released DSI_A_n, and then respond by issuing a Transaction containing a Read Status command. (This Transaction includes the Arbitration, Command and Response Phases, as described in Transaction phases, on page 2.)

If a device does not respond to an Alert within approximately 1 ms the controller reissues the Alert signal periodically until the device responds. The controller only resets its internal alert condition when the device has successfully completed a Read Status command.

1.3 Packets

Each Transaction consists of a Command packet and a Response packet. Command packets are always sent from a device to the controller. Response packets are always sent from the controller to a device. Byte 0 of each packet is transmitted first.

1.3.1 SCSI command

This command is used to forward a Send Diagnostics or Read Diagnostic Results command to the controller:

Table 1. SCSI Command packet

Byte	Name	Contents
0:1	Length	The total number of bytes in all of the following fields, but excluding the Length field itself. (Most-significant byte first.)
2	Command	This byte should be set to 00h for a SCSI command packet.
3:8	CDB	This field contains the 6-byte SCSI CDB for a Send Diagnostics or Read Diagnostic Results command.
9:m	Data	This variable-length field contains the write data for the Send Diagnostics command.
m+1	LRC	This byte is used to check the transmission. It should contain the XOR of each byte in all of the preceding fields.

The controller returns the following Response packet:

Table 2. SCSI Response packet

Byte	Name	Contents
0:1	Length	The total number of bytes in all of the following fields, but excluding the Length field itself. (Most-significant byte first.)
2	Status	This byte contains SCSI Status, as generated by the controller.
3:5	Sense	This 3-byte field contains sense data generated by the controller. It contains the SCSI Sense Key, followed by the Additional Sense Code and Additional Sense Code Qualifier. These bytes are only valid if the Status byte indicates Check Condition.
6:m	Data	This variable-length field contains the read data for the Receive Diagnostic Results command, as generated by the controller.
m+1	LRC	This byte is used to check the transmission. It should contain the XOR of each byte in all of the preceding fields.

1.3.2 Read Status command

This command is generated internally by the device in response to an Alert signalled by the controller. It is used to synchronise the device status and the enclosure status:

Table 3. Read Status command packet

Byte	Name	Contents
0:1	Length = 3	The total number of bytes in all of the following fields, but excluding the Length field itself. (Most-significant byte first.)
2	Command	This byte should be set to 01h for the Read Status command.
3	DevStatus	This byte contains device status as indicated by the following flags: Bit 7 PFA. This bit is set to 1b if the device is reporting a Predictive Failure Analysis error. Bit 6 Fault. This bit is set to 1b if the device is reporting any of the following fault conditions: • One or more links not operational • Motor won't start
4	LRC	This byte is used to check the transmission. It should contain the XOR of each byte in all of the preceding fields.

The controller returns the following Response packet:

Table 4. Read Status response packet

Byte	Name	Contents
0:1	Length = 4	The total number of bytes in all of the following fields, but excluding the Length field itself. (Most-significant byte first.)
2	Slot	This byte identifies the enclosure slot that is occupied by the device.

Byte	Name	Contents
3	Control	This byte controls the device state as follows: Bit 7 Identify. When this bit is set to 1b the device should set its indicators to the Identify state. Bit 6 Remove. When this bit is set to 1b the device should set its indicators to the Remove state. Bit 5 In use. When this bit is set to 1b the device should set its indicators to the Do Not Remove state. Bit 4 Device Fault. When this bit is set to 1b the device should set its indicators to the Device Fault state. Bit 3 PFA error. When this bit is set to 1b the device should enter the PFA Error state. Only one of the Indicator bits should be set at a time.
4	EncStatus	This byte reports enclosure status as follows: Bit 7 Enclosure fault. When this bit is set to 1b the enclosure is reporting a redundant power or fan fault. If this is the first such report the device should send an Async_alert SMS to the Master node on the SSA Web.
5	LRC	This byte is used to check the transmission. It should contain the XOR of each byte in all of the preceding fields.

1.4 Error recovery

This is intended to be as simple as possible. The concepts are as follows:

- The receiver ignores invalid packets, e.g. in case of an LRC error.
- Each hand-shake has a time-out, T_1 » 1 ms.
- If either party detects an invalid packet or a time-out then it waits for a much longer period, T_2 » 10 ms. This ensures that the other party also recognises an error.
- When either party recognises an error it aborts the transaction and returns the interface to the Idle state. This ensures that the controller and the device are resynchronized.
- If a Transaction fails the device arbitrates again and reissues it.

1.5 Performance

Some of the SES diagnostic pages contain several hundred bytes of data. An SES command should not take significantly longer than a disk access, say 10 - 20 ms. This implies a data rate of about 25 Kbytes/s, or 5 µs per bit. Assuming each processor is capable of 5 MIPs this is within reach of the data transfer protocol previously defined.

Higher speeds would require special hardware or faster processors. One of the benefits of the inter-locked data transfer protocol is that the speed automatically scales up as the processors get faster.

1.6 Appendix

1.6.1 Controller implementation

Typically the controller will be based on a single-chip micro, e.g. an 8751 or something even cheaper. An external multiplexer for the DSI_A_n lines may be required if there are many device slots. In some enclosures the micro may also have other duties outside the DSI specification, e.g. an operator panel. A small non-volatile memory would be useful, e.g. to contain a user-assigned location code for the enclosure.

1.6.2 Inter-IC Bus

This is a 2-wire serial interface invented by Philips, originally to provide a digital control interface for analogue chips in TV sets. I^2C is now widely used for other purposes, e.g. as the basis for the Access bus.

I^2C has been suggested as an alternative transport mechanism for DSI. However it would require significant modifications for this application:

1. A geographic addressing scheme is needed to identify the device slots, e.g. radial wiring or X-Y matrix addressing. A simple solution would be to dedicate an I^2C bus to each slot but this is unnecessarily expensive.
2. I^2C is not fully inter-locked, e.g. during Start and Stop conditions. This would require the device to sample the bus at a guaranteed minimum rate which is difficult to ensure when the I^2C protocol is implemented purely by firmware in a disk drive.
3. A licence may be required if I^2C is implemented purely in firmware.

1.6.3 X-Y matrix addressing

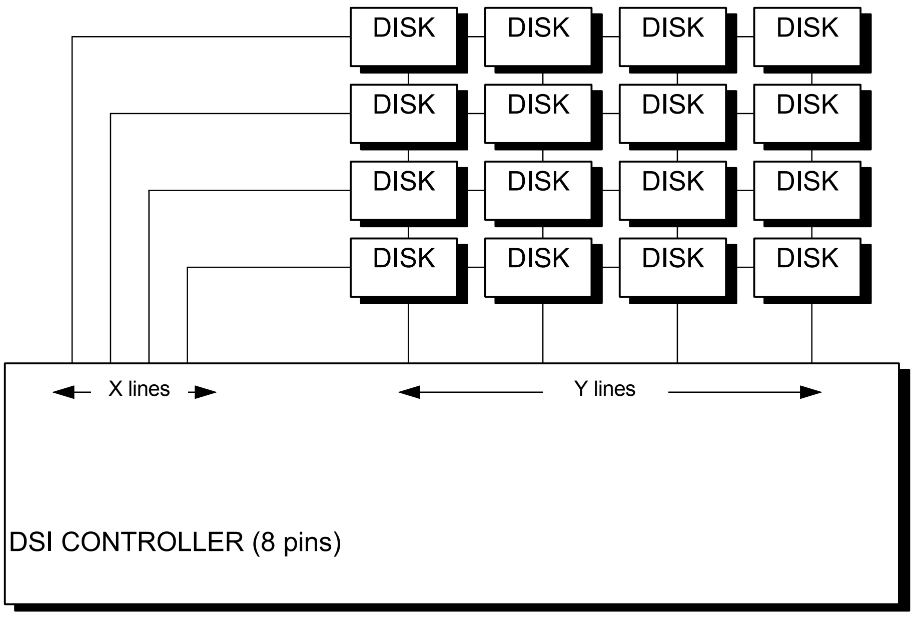

Figure 5. Alternative X-Y addressing scheme

X-Y matrix addressing has been proposed as an alternative to radial wiring. This would reduce the number of pins required on the controller if there are many devices.

The data transfer protocol would remain unchanged. However the controller must now poll each device in turn for requests. A device detects a poll when its X and Y pins are both asserted simultaneously; then it waits until X and Y are negated and starts to transmit a Command packet within, say, 50 µs.

This idea adds some polling delay (about 100 µs per device) and it introduces a minor timing dependency at the device. The saving in controller pins is not very large for a moderate number of devices, e.g. 8 X-Y signals versus 17 radial signals with 16 devices.

1.6.4 IBM 7131 and 7133 compatibility

The existing IBM 7131 and 7133 enclosures already use the Programmable pins to signal an ‘external fault’. (Loss of redundant power and cooling.) Future SSA device with SES support must implement an SES mode bit to work correctly in these enclosures:

- After power-on reset the device defaults to non-SES mode, i.e. the SES bit is reset to 0b in the Inquiry data. (Byte 6, bit 6.)
- At any time after power-on if the device observes that Programmable 1 is released and Programmable 2 is asserted (the DSI Idle state) then it enters SES mode, i.e. the SES bit is set to 1b in the Inquiry data and the device responds to the Send Diagnostics and Receive Diagnostic Results commands.

In non-SES mode the device treats both Programmable pins as inputs:

- The device monitors the Programmable 1 pin.
- Normally Programmable 1 is released.
- If the device observes that Programmable 1 is asserted then it recognises an external fault.

In SES mode the device treats both Programmable pins as a DSI port.

Question: What about a non-SES device in a DSI enclosure?