Serial Storage Architecture

by I. D. Judd
P. J. Murfet
M. J. Palmer

This paper describes Serial Storage Architecture (SSA), a definition and general specification of a high-performance serial link for the attachment of input/output devices. An overview is given of the architecture itself, followed by a general description of the hardware implementation of a dual-port SSA node on a single chip.

Introduction

Serial Storage Architecture (SSA) defines a high-performance serial link for the attachment of input/output devices. It has been optimized for storage applications such as hard disk drives, host adapter cards, and array controllers. SSA has many advantages over existing parallel interfaces such as the Small Computer Systems Interface (SCSI-2). It uses compact cables and connectors, and it has better performance, connectivity, and reliability. However, to facilitate migration, SSA retains much of the SCSI-2 logical protocol.

Current SSA implementations such as the IBM 7133 Disk Subsystem [1] provide a peak data rate of 20 MB/s in each direction. However, a typical loop configuration with one host adapter can provide a total sustained bandwidth of up to 73 MB/s, and higher speeds are becoming available. The physical medium is usually a copper cable up to 20 meters long, but fiber optics can also be used for longer distances.

This paper describes the link architecture and its implementation in a single CMOS chip.

Why serial?

Serial links offer many advantages over traditional parallel interfaces:

- *Compact cables and connectors* This is becoming increasingly important with the trend to small-form-factor devices, e.g., 3.5-in. and 2.5-in. hard disk drives.
- *Improved connectivity* A single host interface can attach many devices. The devices can also be shared among several host computers.
- *Higher performance* A serial interface can provide full-duplex communication at little extra cost. Frame multiplexing allows the interface to be shared efficiently among several concurrent operations.
- *Higher reliability* Error detection is improved by using a cyclic redundancy check (CRC) rather than a simple bus parity check. Point-to-point cables allow faults to be isolated more easily than in a multidrop bus. Devices can readily be dual-ported to provide alternate paths and allow continuous operation in the event of a fault. Individual devices can be "hot-swapped" for concurrent maintenance.

Architecture overview

SSA is defined in three layers:

- *SSA-PH1* [2] defines the electrical specifications, cables, and connectors.
- *SSA-TL1* [3] is a general-purpose transport layer. It defines the transmission protocol, configuration, and error recovery.
- *SSA-S2P* [4] is a mapping of the SCSI-2 queuing model, command set, status, and sense bytes.

Transport layer (SSA-TL1)

- *Nodes*

An SSA *node* is an addressable unit such as a host adapter card, a switch, or a hard disk drive. In some cases a

0018-8646/96/$5.00 © 1996 IBM

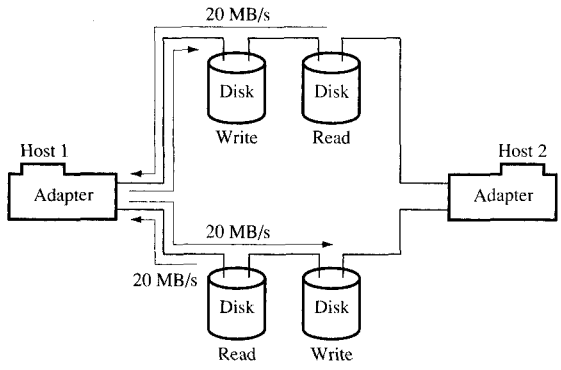

Figure 1

SSA loop showing full-duplex transfers with spatial reuse.

physical unit may consist of more than one SSA node. For example, an array controller may contain one SSA node that connects to the host computer and a second node that connects to the disk drives. For addressing purposes, SSA nodes are classified as follows:

- A *single-port* node has only one SSA port. A port is the hardware and firmware that controls one end of a link.
- A *dual-port* node has two SSA ports. In addition to its internal function, it contains a hardware *router* to allow a frame received on either port to be forwarded via the other port.
- A *switch* node has 4, 6, 8, $\cdots$ 126 ports and a hardware router. It can forward a frame received by one port to any other port, and it may have internal functions as well.

Dual-port nodes are by far the most common. For example, an SSA disk drive is nearly always a dual-port node.

- *Networks*

SSA nodes can be interconnected by *links* to create an arbitrary network called a *web*:

- A *string* is a linear web with 2–129 nodes. The two end nodes can be single-port, dual-port, or switch nodes. Any additional intermediate nodes must be dual-port nodes.
- A *loop* is a cyclic web containing 2–128 dual-port nodes *only*.
- A *complex web* contains one or more switch nodes and a number of strings.

Some typical SSA networks are shown in **Figures 1** and **2**.

The unit of information transfer on a link is a *frame*, which contains up to 128 data bytes. Each frame contains an address field to route it through the web from the source node to the destination node. The address also allows a single link to be multiplexed among many concurrent operations.

If a dual-port or switch node receives a frame that is addressed to another node, the router can forward the frame with a minimum delay of only 5 bytes. This is called *cut-through* routing. Compared to store-and-forward routing, it greatly reduces the latency in large strings or loops. However, if the addressed output port is currently busy, the receiving port must still be able to buffer the entire frame until the output port is available.

Since each link in a web operates independently of the other links, there can be several simultaneous data transfers, each at the full link bandwidth. For example, the adapter card for Host 1 in Figure 1 provides a peak bandwidth of $2 \times 20 = 40$ MB/s for read data. This feature is called *spatial reuse* [5]. Because the links also provide full-duplex communication, a further 40 MB/s is available for write data. Allowing for overheads, the total sustained bandwidth is 73 MB/s.

An SSA web can be configured with alternate paths to protect against hardware failures. A failure is automatically bypassed if an alternate path is available. For example, a loop provides two alternate paths, one clockwise and the other counterclockwise; thus, the remaining nodes can still communicate after any single failure. Since a disk array with parity protection provides similar redundancy, all of the disk drives in such an array can be connected in a single SSA loop. There is no need to provide a separate interface for each disk drive, as required by a multidrop parallel interface. This makes the link cabling independent of the array configuration.

- *8B/10B code*

SSA uses an 8B/10B code [6] to convert each byte into a 10-bit *character* for transmission on a link. Therefore, to achieve a peak data rate of 20 MB/s, the signaling rate on the serial link is 200 Mbaud.

The 8B/10B code has a maximum run length of five consecutive *0*s or *1*s. This allows the receiver to recover a clock from the transitions in the data. The code also has a maximum digital sum variation from -3 to $+3$. (The digital sum is a running count of the *0*s and *1*s, counting a *0* as -1 and a *1* as $+1$.) Consequently, the serial data are *dc-balanced*. This permits ac-coupling of the signal, and it allows the logic threshold to be determined simply from the average value of the received signal. These features facilitate the use of fiber optics where longer distances are required.

The 8B/10B code provides twelve special characters in addition to the 256 data characters. Three of the special characters have the additional property that they do not occur in any combination of the other characters. They allow character synchronization to be established and are called *comma* characters.

SSA uses the special characters as follows:

- SSA-TL1 has eight *protocol* characters, e.g., for framing and flow control.
- SSA-S2P uses one special character for synchronizing the rotation of disk drives, e.g., in a disk array. This allows the synchronization signal to be multiplexed onto the SSA link rather than using a separate cable.
- Two other special characters are made available as user-defined characters.
- The remaining special character can cause false synchronization in certain combinations and is defined as invalid.

- *Frames*

As mentioned previously, information is transmitted on an SSA link in frames. A frame consists of at least six data characters delimited at each end by a *flag*. The flag is a comma character which is also sent continuously when the link is idle. To minimize overheads, the trailing flag character of one frame can also be the leading flag character of the next frame.

A frame may contain some data or a message. Messages are used for control information, e.g., a command or status. A frame normally contains four fields, as shown in **Figure 3**: a control field, an address field, a data field, and a CRC field.

Control field The control field is always the first byte following the leading flag character of a frame. It indicates the frame type as follows:

- *Control* frames are used only by SSA-TL1 to reset a link or a node, e.g., after an error.
- *Privileged* frames are also used only by SSA-TL1 for network configuration and error recovery.
- *Application* frames are used by SSA-S2P.

The control field in privileged and application frames also contains a 2-bit frame sequence number (FSN) which increments modulo-4 in each successive frame. The sequence number detects a missing frame caused by a link error, and it is also used in the subsequent recovery procedure.

Address field The address field follows the control field. It normally contains 2 to 6 bytes, and it is divided into a *path* address followed by a *channel* address. (Control frames do not contain a channel address.)

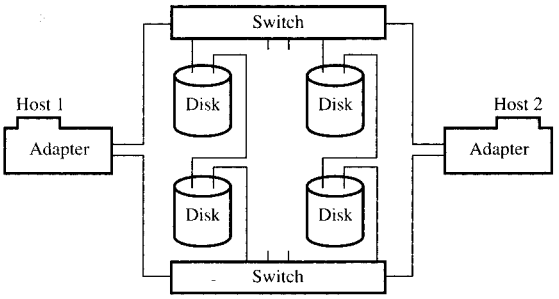

Figure 2

Complex web using switches.

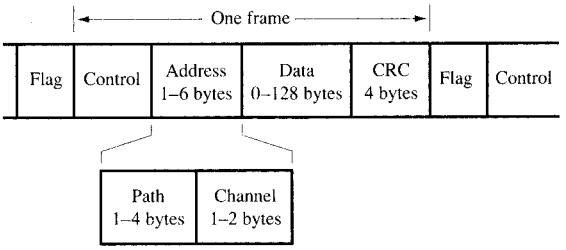

Figure 3

Frame format.

The path address is used to route the frame through the web from the source node to the destination node. The path address for a string or loop takes 1 byte, but it can be extended up to 4 bytes for a complex web containing switches. SSA uses a source routing algorithm. The router in each node only needs to manipulate the first byte of the path address. If the first byte is *0*, the current node is the destination node. Otherwise, a dual-port node decrements the first byte and forwards the frame out of the other port. A switch node uses the first byte to select an output port. Then it normally deletes the first byte.

The channel address is used to route the frame within the destination node. It is usually 1 byte, but it can be extended to 2 bytes if required. The channel addressed by the single byte *0* is predefined to receive messages. All other channels are dynamically allocated by the destination node to receive data transfers. Typically a data channel consists of a hardware direct memory access (DMA) facility.

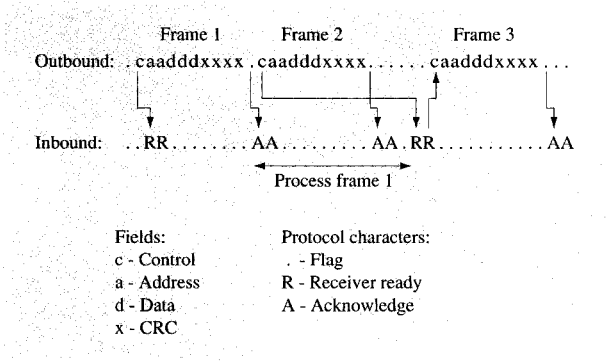

Figure 4

Flow control.

Data field The data field follows the address field; it contains the payload of the frame. This may be a message with a maximum length of 32 bytes or some data with a maximum length of 128 bytes. The data field is not present in control frames.

The maximum length of the data field has been deliberately restricted in order to minimize the size of the frame buffers needed in each port. (Typically a port will have two receive frame buffers and two transmit frame buffers, which are normally implemented in RAM on the protocol chip.) However, since the framing overhead for a loop is only eight characters, an efficiency of 94% can still be achieved.

CRC field The CRC field consists of the last four bytes of a frame before the trailing flag character. It is calculated using a degree-32 polynomial.

The CRC covers the preceding control, address, and data fields, but it does not include the flag characters or any other intervening protocol characters. Since the control and address fields are updated as a frame is routed through the web, the CRC must be recalculated at each node.

- *Flow control*

The SSA-TL1 protocol for flow control is robust and efficient but very simple. The protocol operates independently on each link in a web.

When a port transmits a frame, it expects to receive two responses from the remote port:

- An *acknowledgment* indicates that the frame was received correctly, e.g., that the FSN and CRC were correct. It informs the transmitter that it can free the corresponding frame buffer. The acknowledgment must be received within 25 μs; otherwise, the transmitter may detect a time-out error.
- A *receiver_ready* indicates that the remote port has another frame buffer free to accept the next frame. It allows the transmitter to send another frame. If all of the frame buffers are full, the receiver_ready may be delayed indefinitely.

To permit continuous data transfer, the remote port may send receiver_ready anytime after it has received the control field of the current frame. Consequently, to operate at full speed, the two-way delay of the cable is limited to a little less than the time to send one frame. This is about 650 meters with 128-byte frames at 20 MB/s.

The protocol is illustrated in **Figure 4**. Each response is signaled using a pair of protocol characters (AA or RR) rather than a frame. This allows responses to be interleaved within a frame to reduce latency during full-duplex transfers. The receiver can easily filter out the responses, since they are represented by protocol characters. The responses are checked by duplication, rather than a CRC.

An acknowledgment must be unambiguously associated with a particular frame, even in the presence of transmission errors. This requires the transmitter to insert null protocol characters before the trailing flag of the current frame if it has not yet received an acknowledgment for the previous frame. Consequently, there can never be more than one frame for which an acknowledgment is outstanding.

- *Configuration*

An SSA web is configured automatically after power-on. There are no switches or jumpers to assign node addresses. Nodes and links can also be connected and disconnected dynamically.

Each SSA node is either a *configutor* or a *responder*. A configutor (such as a host adapter) understands the topology of the web; a responder (such as a disk drive) does not.

Each SSA node must be assigned an 8-byte *unique_ID* during manufacturing. The unique_ID is usually stored in an erasable programmable read-only memory (EPROM). It is used to detect cyclic paths during the configuration process.

During the configuration process each configutor must build a configuration table. This is essentially a list of every node in the network with pointers to represent the links. Each configutor builds its table by "walking" the web one node at a time. A configutor first issues a query_node message to an adjacent node. That node returns a query_node_reply message which indicates its unique_ID, the number of ports it has, and which ports

are operational. This provides the information needed to walk to the next node, and so on, until all of the nodes have been visited.

During configuration the configutor with the highest unique_ID is automatically elected as the *master* node. The master is responsible for configuring each port in the web with various parameters, e.g., the SAT quotas described later. It also coordinates the processing of asynchronous events such as dynamic changes in the network configuration.

During configuration a responder assigns a *return_path_ID* to each configutor that wishes to communicate using an upper-level protocol (ULP). The responder also saves the port and path address that it will use to return ULP messages to the configutor.

- *Reliability*

Each SSA port has a wrap mode which routes the serializer output directly to the deserializer input. This provides a local power-on self-test independently of the cable or the remote node.

During normal operation each port detects the following errors:

- Cable faults.
- Loss of synchronization.
- Violation of the 8B/10B code.
- Incorrect CRC.
- Incorrect FSN.
- Acknowledge time-out.
- Various protocol errors.

When a port detects an error, it enters the check state and sends a link_reset control frame to the remote port. This causes the remote port also to enter the check state and return a link_reset frame. The link_reset contains the sequence number of the last frame correctly received by the port which is sending the link_reset. The receiving port uses this to determine whether a frame which it previously transmitted has been lost. If so, the receiving port can simply retransmit the frame, since a transmit buffer is not freed until the corresponding frame has been successfully acknowledged.

- *SAT algorithm*

The SAT algorithm operates on a string or loop to ensure that each port has a reasonable share of the available bandwidth for originating frames. It also allows spatial reuse when concurrent transfers are not competing for the same links.

During configuration, the master allocates each port two quotas for originating frames, A and B ($A \leq B$). Typically, for a loop with ten nodes, $A = 2$ and $B = 8$. The operation of the algorithm for a loop is as follows:

- Two SAT protocol characters circulate around the loop, one in each direction. Each SAT character controls the frames traveling in the opposite direction.
- Except when it is holding a SAT character, a port gives priority for transmission to frames being forwarded from the other port. This minimizes latency in a large loop.
- A port is allowed to originate a frame only if it is holding a SAT character or if there are no frames waiting to be forwarded from the other port and it has originated fewer than B frames since it last forwarded the SAT character.
- When a port receives the SAT character, it forwards the SAT via the other port if it has originated at least A frames since it last forwarded the SAT (i.e., it is SATisfied), or if it has no frames to originate.
- Otherwise the port holds the SAT character. This gives priority to the frames being originated by the port and allows it to originate its A quota. It will then forward the SAT character.

This algorithm is extended to a string by using an additional SAT′ protocol character. When a SAT character reaches the end of the string, it is reflected as SAT′. The SAT′ is then forwarded unconditionally to the other end of the string, where it is reflected as a new SAT character.

SCSI-2 mapping (SSA-S2P)

SSA-S2P is designed to facilitate the migration of existing firmware from a parallel SCSI-2 environment. It retains the SCSI-2 queuing model, the command descriptor block (CDB), the status byte, and the architected sense bytes. SSA-S2P is concerned primarily with mapping the SCSI-2 bus phases and messages to the SSA-PH transport layer. Also, since parallel SCSI uses individual bits of the bus to address the initiators and targets, SSA-S2P has a different addressing model.

In SCSI-2 an *initiator* (such as a a host adapter) issues commands and a *target* (such as a disk drive) executes them. In SSA terms, an initiator is a configutor and a target is a responder.

SSA-S2P has several features that allow higher performance than parallel SCSI-2:

- Frame multiplexing replaces arbitration, disconnection, and reselection. This reduces overhead and also allows a new command to be issued promptly during a long data transfer.
- SSA-S2P supports out-of-order data transfers, e.g., for zero-latency reads or disk arrays.
- SSA-S2P allows concurrent commands on the same device, e.g., a read from the cache concurrently with a write to the disk.

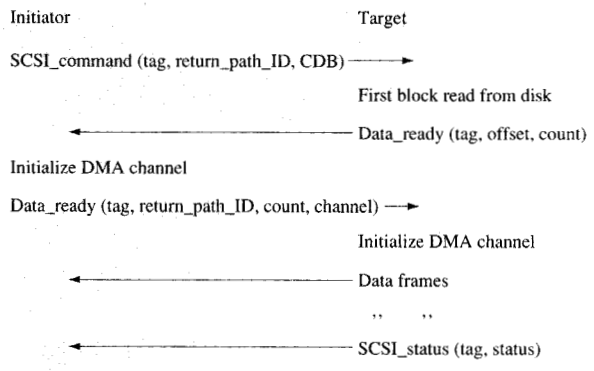

Figure 5

SSA-S2P read command.

- The message protocol has been designed to minimize the number of initiator–target exchanges. Typically a write command requires just three messages and a read requires four messages.

- *Messages*

Both SSA-S2P and parallel SCSI-2 use messages for control information. However, there is no one-to-one mapping, and the two systems should not be confused. In some cases a single SSA-S2P message performs the function of several messages in parallel SCSI-2. As mentioned above, other functions which are performed by messages in parallel SCSI-2 are not required in SSA-S2P.

Each SSA-S2P message must be contained in a separate frame with a maximum data field of 32 bytes. The first byte in the data field always contains a code to indicate the function of the message. The destination node can easily differentiate a message frame from data because a message is always addressed to channel *0*.

Each message also contains a 2-byte tag which is usually allocated by the initiator when it issues an SCSI command. The tag must be unique among all of the active tags from that initiator. When a tag is returned in a message from a target to an initiator, it identifies the relevant target and command. When it is sent in a subsequent message from the initiator to the target, a tag identifies the relevant command. A tag is normally freed when the target returns status for the corresponding command.

If an initiator sends a message to a target, it also supplies the return_path_ID as one of the parameters of the message. When the target wishes to send a frame in reply, it simply inserts the corresponding path address of the initiator into the path component of the frame address field. Consequently, the target need not understand the topology of the web. The return_path_ID also serves to identify a particular initiator to the target, since tags are not unique among different initiators.

- *Read command*

Figure 5 shows the SSA-S2P protocol for a typical read command:

1. The initiator sends the target an *SCSI_command* message containing a 6-, 10-, or 12-byte CDB, as architected by SCSI-2. The target decodes the SCSI_command message and enters the command into its queue.
2. When the target executes the command and has read the first block from the disk into its buffer, it returns a *data_ready* message to the initiator. The offset parameter allows out-of-order transfers, and the count specifies the number of bytes that the target is currently offering to send.
3. The initiator sets up a hardware DMA channel to receive the data. It then sends the target a *data_reply* message containing the channel address corresponding to that DMA channel.
4. The target sets up a DMA channel to send the data. Each frame contains the channel address supplied in the data_reply message.
5. Finally, when the target has finished sending the data, it returns an *SCSI_status* message to the initiator. This contains the status byte, as architected by SCSI-2.

The disk drive may be able to perform a zero-latency read; i.e., it can start reading as soon as the head has settled rather than waiting for the first block requested by the host. In this case there would be two pairs of data_ready and data_reply messages, one for the "tail" of the data and a second for the "head."

If the initiator had limited buffer space, it could restrict the count in the data_reply message to be less than the number of bytes offered by the target. Later the initiator would send a further data_reply message to obtain some or all of the remaining data.

Implementation

Serial links with performance similar to SSA have been available for some time. However, previous designs are often bulky, expensive, and power-hungry. Some use separate chips for the line driver and receiver, or the serializer and deserializer. Few are suitable for a small disk drive, where the hardware has to be compact, cheap, and consume little power. Ideally for this application, the interface should be capable of integration as part of a larger chip that includes other functions such as the disk formatter.

Currently an entire dual-port SSA node has been implemented in a single serial interface chip (SIC). The SIC connects directly to the interface cable and includes the SSA protocol, the router, the frame buffers, and 24 DMA channels. It has a flexible back-end interface which can support adapter, controller, switch, and disk-drive applications. However, the following sections describe only the high-speed circuits which deal with the serial data.

Macros

The SIC measures 7.5 mm square; it is fabricated using a 0.45-μm (effective) technology having three layers of metal. This is a standard CMOS logic process with no enhancements, to enable the designer to use analog components such as resistors and bipolar devices. The choice of process was based more on the requirements of the logic than those of the analog circuits.

The SIC was designed as a set of macros using a standard-cell methodology. This means that in future the SSA node could become part of a larger chip that includes other functions. The following macros are associated with the serial data:

- Deserializer.
- Serializer.
- Phase detector.
- Voltage-controlled oscillator (VCO) and charge pump.
- Bias current generator.
- Line driver.
- Line receiver.

The first three are logic macros. Owing to the 200-MHz clock rate and critical timing, the logic gates and interconnections are fixed within each macro. However, the macros themselves can be moved around the chip. The last four macros are analog circuits which are located in the input/output (I/O) cells on the periphery of the chip. All of the macros use the same 3.3-V power supply as the logic, although some are fed via separate pins to reduce inductively coupled noise.

- *Serializer and deserializer*

The serializer and deserializer convert between 10-bit parallel data and the serial bit stream on the cable. The design challenges result from the speed at which this logic must operate. SSA currently operates at 200 Mbaud, which gives a clock period of only 5 ns and limits the design to about four levels of (lightly loaded) gates between latches.

The serializer is by far the simpler design, since it only has to clock bits out based on a local clock. It consists of a conventional parallel-load shift register running on one phase of the 200-MHz clock. An internal divide-by-10 counter controls parallel loading. The serializer is checked by computing parity serially on the data leaving the shift register; at the end of each character, this is compared with a parity bit on the incoming parallel data.

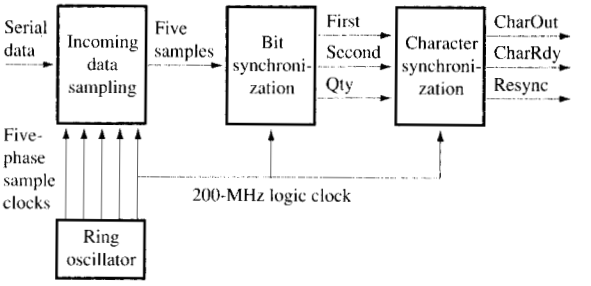

Figure 6

Deserializer.

The deserializer receives a data stream of unknown phase and slightly variable frequency owing to crystal tolerances between the transmitter and receiver. It must determine the locations of the boundaries between bits and the boundaries between encoded characters. The deserializer must also be tolerant of jitter in the signal edges due to distortion on an imperfect cable and in the line driver and receiver. Ultimately, deserializer performance has a significant effect on the maximum cable length between SSA nodes.

A common deserializer design is to regenerate a clock from the incoming data stream. The clock, running at the bit rate and with a known phase, can then be used directly to clock the data into a shift register. However, creating such a clock is not straightforward. An analog phase-locked loop (PLL) can be used, but allowance must be made for jitter and the unknown bit pattern in the incoming data. This approach requires precision analog circuitry, and this would be required separately for each SSA port.

The alternative is to use a digital scheme which samples the incoming data at a rate much higher than the bit rate and then deduces where the bit boundaries occur. This technique is common in low-speed serial links such as RS-232 serial receivers, where the sampling rate is often 16 times the bit rate. However, for high-speed links the clock rate soon becomes prohibitive. Even sampling at five times the bit rate, SSA would require a 1-GHz clock, which is very difficult with current CMOS technology. The design chosen recovers the data digitally, but without having to clock any logic at a rate above 200 MHz.

A block diagram of the deserializer is shown in **Figure 6**. The ring oscillator operates at 200 MHz and provides

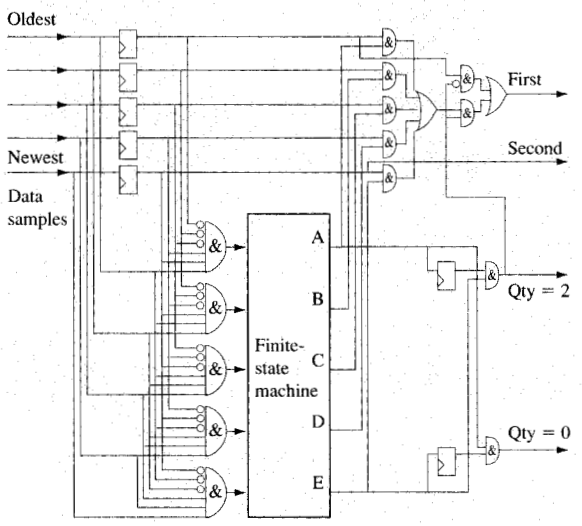

Figure 7

Bit synchronizer.

five clock phases. Note that the ring oscillator is not aware of the incoming data, and it is not phase-locked to the data or even at exactly the same frequency. A single ring oscillator can be shared among several SSA ports. One of the clock phases is used as the single 200-MHz clock for the bit and character synchronization logic.

The five clock phases are used to sample the incoming data stream at 1-ns intervals. Metastability must be considered on the input sampling latches, and our latch design has been carefully characterized to guarantee the error rate.

Five consecutive samples are then fed into the bit synchronizer, shown in detail in **Figure 7**. The bit synchronizer has five latches to store the five samples. These allow it to examine ten consecutive samples (the five latched from the previous cycle and the five current samples) for rising edges in the incoming data. The edges are detected by five AND gates looking for a *000111* pattern at five possible alignments within the ten samples.

The AND gate outputs drive a finite-state machine with five states which correspond to the five inputs. The state determines which data sample is closest to the center of a bit. The selected sample is output on the signal *First*. As the incoming phase drifts, the state machine will step to match the data. The state change is restricted to go from the current state only to an adjacent state; this improves the response to jitter.

An interesting situation occurs if the incoming bit rate differs slightly from the local clock rate, as it is bound to do. Eventually a local clock cycle will occur in which zero or two bits must be transferred, instead of the usual one bit per cycle. This situation is indicated when the state machine wraps around from its last state, E, to its first state, A (in which case zero bits are transferred), or from A to E (in which case two bits are transferred). The case where data arrive slightly faster than the local clock is illustrated in **Figure 8**. The *Qty* outputs from the bit synchronizer indicate the number of bits being output in the current clock cycle, which can be 0, 1, or 2. The bit(s) themselves appear on the *First* and *Second* signals.

The last block in the deserializer is the character synchronizer, which is shown in detail in **Figure 9**. It receives the *Qty*, *First*, and *Second* signals from the bit synchronizer and outputs 10-bit characters on the CharOut bus. To do this, the bits are fed into a special shift register which is capable of shifting 0, 1, or 2 places in each cycle, as indicated by the *Qty* signals.

A counter, counting up from 0 to 9, records the number of data bits currently held in the shift register. The value of *Qty* is added to the current count during each clock cycle. Whenever the counter wraps from 9 to 0, this indicates that a 10-bit character is complete, and it is loaded from the shift register into the output buffer register. Since a character can be formed with either 9 bits from the shift register and *First*, or with 8 bits plus *First* and *Second*, a multiplexor selects the input to the output buffer register.

A strobe, CharRdy, is output from the deserializer each time the output buffer register is loaded, to signal that a new character is available on the CharOut bus.

The character synchronizer also looks for the three comma characters to identify the boundaries between characters. Each comma character contains a unique 10-bit pattern which does not occur elsewhere. The logic examines the 9 bits in the shift register together with *First*, and activates the signal *Sync1* if they match a comma character. If *Qty* = 2, it also examines the last 8 bits in the shift register together with *First* and *Second* and activates *Sync2* for a comma character. When either *Sync1* or *Sync2* goes active, the counter is set to zero, unless *Sync1* is active and *Qty* = 2. The counter is then set to 1, since this indicates that the 9 bits in the shift register and *First* make up a character, and we have another incoming bit on *Second*.

Whenever *Sync1* or *Sync2* goes active, the counter is checked for 9 or 8, respectively. If this is not the case, a pulse is output on *Resync* to indicate that the character boundary has changed. This is ignored during initialization, but in normal operation it indicates a synchronization error.

This digital clock recovery scheme locks very quickly to the incoming data. The bit synchronizer will lock within

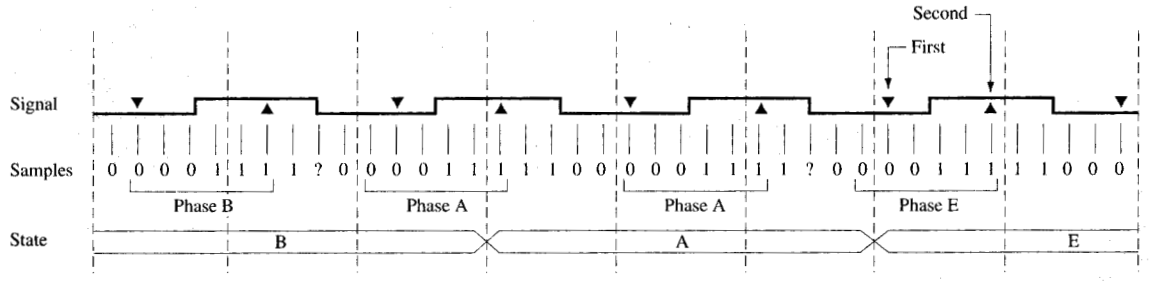

Figure 8

Example of data sampling.

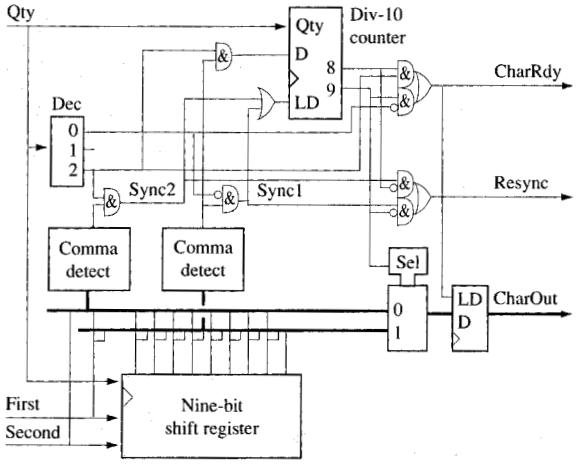

Figure 9

Character synchronizer.

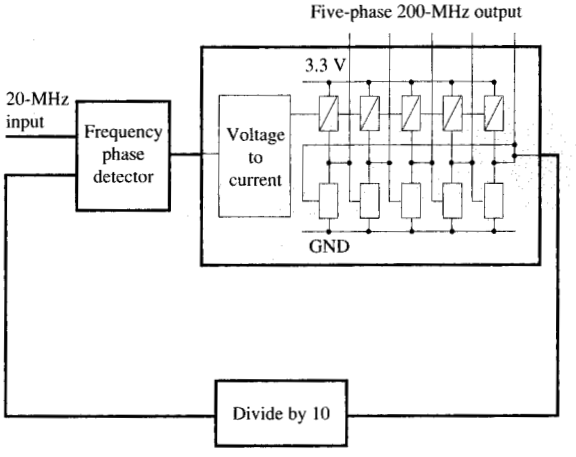

Figure 10

Phase-locked loop.

three positive transitions, and the character synchronizer requires two comma characters.

Phase-locked loop

The deserializer requires a five-phase clock input at 200 MHz. These clocks are generated on-chip from a 20-MHz reference clock using a PLL, as shown in **Figure 10**. The PLL consists of a voltage-controlled oscillator (VCO), a phase detector, a charge pump, and a divide-by-10 counter. The five clock phases are produced directly by the VCO, which is a five-stage ring oscillator.

To reduce jitter, the VCO has its own power pins to isolate it from inductively coupled logic noise. The voltage-to-current converter is also designed for high rejection of power supply noise. Finally, since the loop has to lock only once at power-on, it has a narrow bandwidth.

Line driver and receiver

Figure 11 shows the circuits of the line driver and receiver together with the cable and terminating resistors.

The cable contains four wires arranged as two differential pairs. Each pair has a characteristic impedance

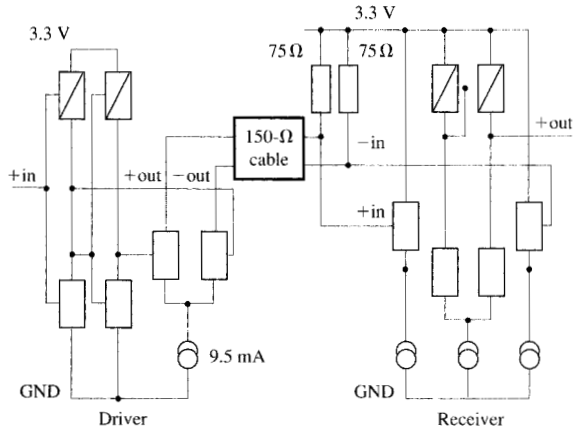

Figure 11

Driver, cable, terminators, and receiver.

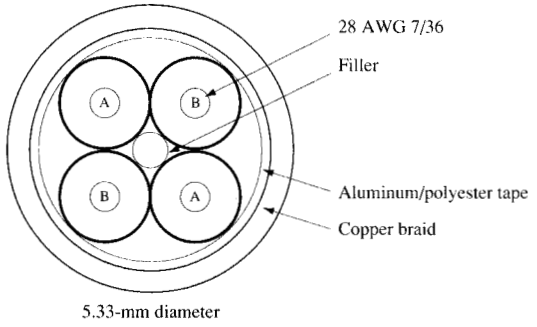

Figure 12

Cross section of external cable.

of 150 Ω. With respect to a particular port, one pair is always used to transmit and the other pair is always used to receive. Each wire is terminated at the receiver through a 75-Ω resistor to 3.3 V. There is no need to plug discrete terminators because they are built into each port.

It is difficult to design a voltage-mode driver with a low output impedance using only CMOS devices. Therefore, a simple long-tailed pair is used as a differential current sink. The driver draws 9.5 mA out of one wire or the other depending on whether it is transmitting a *0* or a *1*. The driver output voltage has a common-mode range of ±0.5 V. Since the cable is unterminated at the driver, common-mode noise between the driver and receiver grounds does not degrade the receiver input signal. However, this requires the cable to be carefully matched, because reflections arriving at the driver are reflected back to the receiver rather than being absorbed.

A good receiver should have exactly equal rising and falling delays in order to preserve the timing of the signal transitions. It should also have a low offset voltage and a high differential gain. Both of these parameters are difficult to achieve using only CMOS devices. The final design is thus a compromise. It is a simple differential amplifier, but particular attention was paid to the physical layout to minimize the offset voltage.

Both the line driver and receiver incorporate comparators to detect invalid voltages caused by a cable fault such as a wire shorted to ground. These are not shown in the circuits.

• *Bias current generator*

All of the analog macros must be biased by current sources. These currents are distributed from a central bias generator, which derives the currents from an external 1.2-V reference. Distributing current sources rather than voltages ensures that the effects of ground noise are kept to a minimum.

• *Layout*

The analog macros are laid out on the periphery of the chip, where logic I/O circuits would normally reside. The power for the VCO and charge pump is fed separately via adjacent pins. The line driver and receiver both take their power from the logic power supply. The two serializers and two deserializers are in the logic area. A serializer uses about 350 gates, and a deserializer uses 1390 gates.

• *Cable*

External cables are used for links between separate enclosures. (Internal cables are used only within a single enclosure and can generally use much thinner conductors with less shielding.)

The cable design is a compromise between minimizing the thickness and maximizing the operating distance. For longer distances the signal attenuation per unit length must be reduced, but this requires thicker conductors. Crosstalk may also have to be reduced by shielding each pair separately, but this further increases the overall cable diameter.

As with all engineering problems, a compromise was chosen. The specified SSA external cable uses 28 AWG conductors with a quad construction, as shown in **Figure 12**. The central filler and outer shields maintain the two pairs

accurately at 90° to one another. This avoids the need for individual shields to reduce crosstalk and gives the most compact construction possible with four wires. The maximum recommended length for this particular cable is 20 meters.

- *Results*

To date the macros have been used on four different chips, and all implementations have performed well within the SSA-PH specification.

Figure 13 shows a plot of the link error rate with no error recovery. The measured results are for a nominal process, a chip temperature of 55°C, and a supply voltage of 3.3 V. The graph also includes the expected worst-case error rate, which is estimated to be three orders of magnitude above the nominal value. This was derived from analysis of the error rate variation with process, supply voltage, temperature, ground noise, and frequency stress. With a maximum cable length of 20 meters, it can be seen that the worst-case error rate is better than 10^{-12} errors per bit.

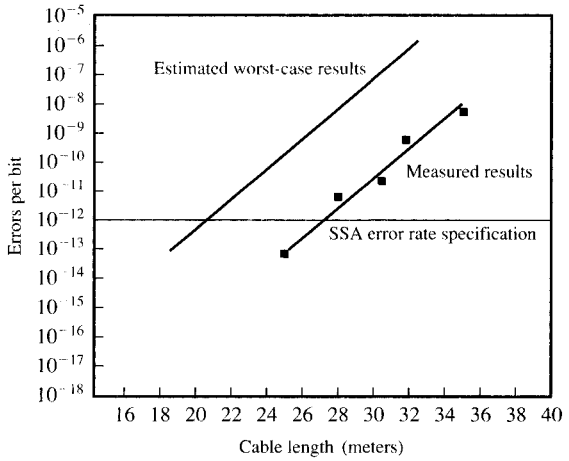

Figure 13

Error rates.

Conclusion

SSA provides a compact, high-performance interface using low-cost CMOS technology. It is ideal for connecting small-form-factor hard disk drives and array controllers with servers and workstations.

SSA is an open architecture, and other companies are using it in their products. Work is also nearly complete at the American National Standards Institute (ANSI) to produce a formal standard. In time this should further reduce costs and provide a wider choice of compatible products.

Acknowledgment

The authors would like to acknowledge the work of many colleagues in IBM and other companies who have contributed to the definition and implementation of SSA.

References

1. "7133 and 7131 Serial Storage Architecture (SSA) Disk Subsystems," IBM Publication G522-2416.
2. "Serial Storage Architecture—Physical, SSA-PH1," *ANSI Standard X3T10.1/1145D*. This specification is available from Global Engineering, telephone 1-800-854-7179.
3. "Serial Storage Architecture—Transport Layer, SSA-TL1," *ANSI Standard X3T10.1/989D*. This specification is also available from Global Engineering.
4. "Serial Storage Architecture, SSA-S2P (ANSI SCSI-2 Protocol)," *ANSI Standard X3T10.1/1121D*. This specification is also available from Global Engineering.
5. I. Cidon and Y. Ofek, "MetaRing: A Full-Duplex Ring with Fairness and Spatial Reuse," *IEEE Trans. Commun.* **COM-41**, No. 1, 110–120 (January 1993).
6. A. X. Widmer and P. A. Franaszek, "A DC-Balanced, Partitioned Block, 8B/10B Transmission Code," *IBM J. Res. Develop.* **27**, No. 5, 440–450 (September 1983).

Received February 16, 1995; accepted for publication July 31, 1996

Ian D. Judd *IBM Storage Systems Division, c/o XYRATEX Ltd., P.O. Box 6, Havant, Hants PO9 1SA, United Kingdom (ianjudd@vnet.ibm.com).* Mr. Judd is currently a senior engineer responsible for SSA and its implementation. He received a degree in mechanical and electrical engineering from Churchill College, Cambridge. He joined IBM in 1970 to develop the integrated file adapters for the System/370™ Model 135. His subsequent experience includes image processing and high-resolution graphic displays. Mr. Judd has received a Fourth-Level Invention Award and two Outstanding Innovation Awards. He was elected to the IBM Academy of Technology in 1993.

Philip J. Murfet *IBM Storage Systems Division, c/o XYRATEX Ltd., P.O. Box 6, Havant, Hants PO9 1SA, United Kingdom (pmurfet@vnet.ibm.com).* Mr. Murfet received a B.Sc. degree at Loughborough University, England. He joined IBM in 1974 and has spent the majority of his career designing analog and digital circuits. Most recently he has designed fast CMOS mixed-signal integrated circuits for displays and storage systems.

Michael J. Palmer *IBM Storage Systems Division, c/o XYRATEX Ltd., P.O. Box 6, Havant, Hants PO9 1SA, United Kingdom (mjpalmer@vnet.ibm.com).* Mr. Palmer graduated with a mathematics degree from Trinity College, Cambridge, in 1984, joining IBM Storage Development at Hursley as a logic designer working on disk storage controllers. He has been lead designer for three IBM VLSI chips, the largest and most complex being the serial interface chip for SSA. Mr. Palmer holds two patents relating to serial disk interfaces.