Self-timed interface for S/390 I/O subsystem interconnection

by J. M. Hoke
P. W. Bond
T. Lo
F. S. Pidala
G. Steinbrueck

A high-speed interface has been designed for interconnection of the S/390® I/O subsystem to the IBM S/390 G5 processor. The self-timed interface (STI) provides high bandwidth, greater communication distances, and simpler timing within the S/390 servers than traditional interfaces. The STI communicates between the memory bus adapter and the expanded S/390 I/O subsystem, which now includes the new S/390 fiber channel offering (FICON™) and other network-based protocols such as ATM, Fast Ethernet, and Gigabit Ethernet. Also new for G5 is the use of STI for the integrated cluster bus (ICB), providing direct links among multiple G5 servers. The STI communicates over cables up to ten meters in length at a clock frequency of 167 MHz. Data is sent at twice the clock frequency (333 MB/s). The hardware implementation comprises specially designed logic macros, differential drivers and receivers, and the cables and connectors. The receive macro accommodates up to three bit-times of skew by retiming each data bit of the interface to the transmitted clock. This paper describes the STI logic, the characteristics of the link, and the transmission and reception of the data (and clock).

Introduction

The self-timed interface (STI) was first introduced with the third generation of S/390* CMOS servers (G3) to satisfy the increasing I/O bandwidth requirements of these improving large systems. As the performance of the processors increased, it became apparent that in order for IBM to be successful with its new line of CMOS servers, the bandwidth and connectivity of the I/O subsystem had to scale along with the processor. STI was developed to satisfy those requirements.

Figure 1 is a block diagram of an S/390 CMOS machine showing STI usage. The STI provides the interface between the memory bus adapters (MBAs) and the traditional I/O bridge chips (FIBBs), thus enabling legacy I/O such as parallel channels, ESCON*, and intersystem channels (ISC) for machines prior to G5. Today, with the delivery of the fifth generation of S/390 CMOS servers (G5), the role of the STI is further expanded as the primary interconnection method for the I/O subsystem. The STI is used for the direct connection of the new S/390 fiber channel offering (FICON*), as well as other network-based protocols such as ATM, Fast Ethernet, and Gigabit Ethernet, to the I/O subsystem through a PCI bridge chip. Also new with G5 is the ability to create direct communication links, using the STI, between other G5 servers via the Integrated Cluster Bus (ICB) [1].

0018-8646/99/$5.00 © 1999 IBM

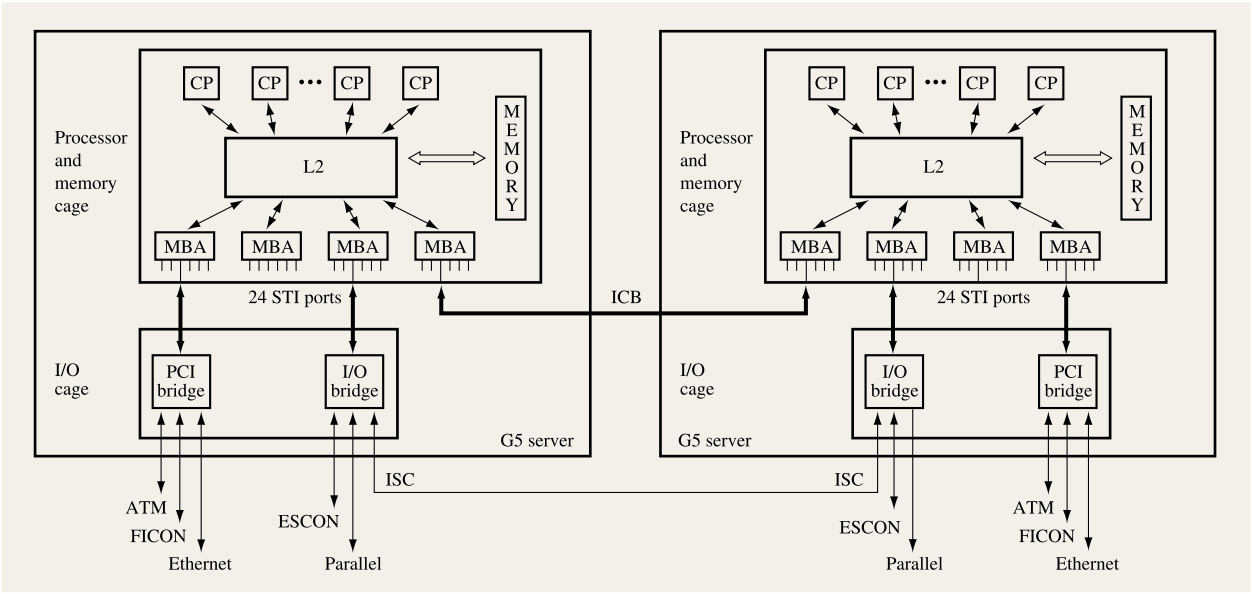

Figure 1

STI links as used in the G5 server.

To appreciate the advantages of the STI as an I/O subsystem interconnection, it is helpful to look at its predecessor. In the G1/G2 machines, the first generation of IBM CMOS mainframes, the I/O subsystem was connected to the processors and memory via the internal bus (IB) [2]. The IB interface was an 8-byte-wide bidirectional data bus running at a 27-MHz clock rate. This allowed a peak data transfer rate of less than 300 MB/s, but in only one direction at a time. Another limiting factor of the IB was its synchronous nature, which limited the maximum length of the cable to one meter. This limited the G1/G2 machines to a single I/O cage. Furthermore, the number of conductors alone made the IB cable very expensive.

Today's STI link uses a relatively inexpensive copper cable and can extend up to ten meters in distance depending upon the system topology. The upper distance limit is dictated by the electrical signal quality delivered to the receiver. (It is entirely realistic that STI links could be extended over much greater distances by the use of optical fiber in place of copper.) With the one-meter distance limitation of the IB eliminated, multiple I/O cages and ICBs were made possible. For example, G5 allows a maximum of three I/O cages.

The STI transmits one byte of data, a combination parity/flag bit and a half-speed clock signal in each direction. This requires 20 differential signal pairs (ten pairs in each direction) in the cable. The raw STI data rate is 333 MB/s (667 MB/s total), producing 3-ns data pulse widths using a half-speed 167-MHz clock signal. One major problem with operating a copper link at these speeds is that the skew between data signals on the link can exceed the bit-time.

The adoption of the STI as the I/O subsystem interconnection dramatically improved the total system bandwidth. An STI port requires 40 chip I/O pins in comparison with an IB port, which uses more than 80 chip I/O pins. Consequently, the G1/G2 MBA chip had only enough chip I/O pins for two IB ports. Aided by technology density improvements, a G5 MBA chip is able to support six STI ports. This gives the current S/390 CMOS servers more I/O subsystem bandwidth than was possible with even the most powerful bipolar mainframes [2].

The STI is implemented using specially designed semicustom macros to handle the STI physical-layer protocol. The function is partitioned into a physical send macro (PSM) and a physical receive macro (PRM), as shown in **Figure 2**. These entities interface with the STI logical-layer macros, LSM and LRM, respectively. The STI logical macros handle flow control and packet creation for the STI link [1, 2]. The primary function of the PSM is to serialize the word-wide interface from the LSM down to one byte for transmission onto the STI cable. The function of the PRM is to receive the signals from the cable, resynchronize the data signals to the transmitted clock, and, finally, deserialize the four bytes of data back into a

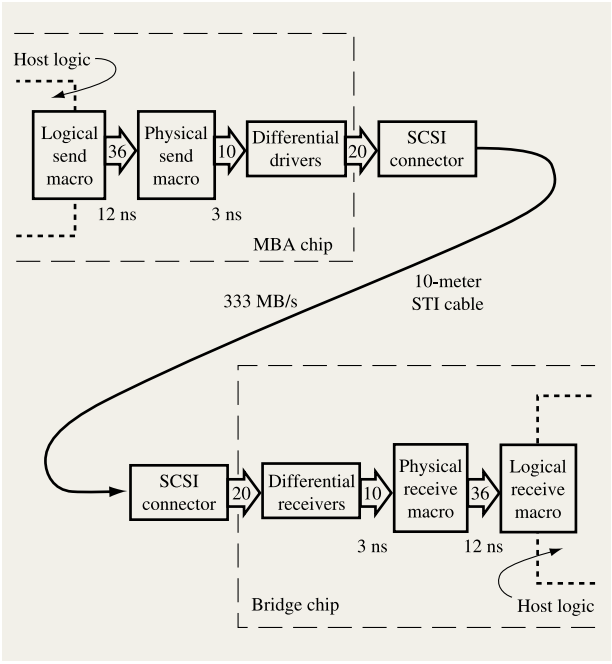

Figure 2

Components of the STI link.

word for presentation to the LRM. A detailed description of the PSM and PRM follows.

The STI combines both parallel and serialized data running at high speeds. This paper discusses the challenges encountered during the development of the STI physical link and details its implementation. The topics presented include the electrical characteristics of the copper cable and the resulting consequences, the problem of jitter and noise on link performance, the effects of silicon tolerances, and the ability of the STI to compensate for skew between data bits.

STI physical send macro

- *Description of the transmit logic*

A high-level block diagram of the PSM is shown in **Figure 3**. As mentioned earlier, the function of the PSM is to serialize a word of data received from the LSM and transmit it one byte at a time over the cable interface. Figure 3 shows the bit-slice nature of the PSM. The logic is designed for a single bit and then replicated eight times. This is referred to as a bit-slice or "one-bit" of the PSM. The details of the PSM one-bit are shown in **Figure 4(a)**. The serialization process first converts the four bits of data captured from the LSM into two-bit data fields using the capturing 12-ns oscillator as a controlling input to a

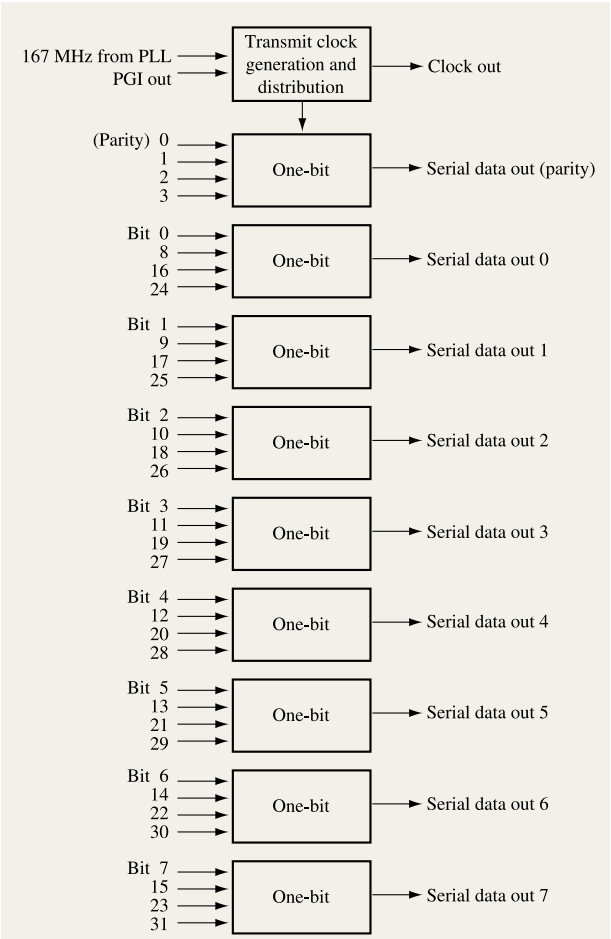

Figure 3

STI physical send macro.

pair of 2:1 selectors. When the 12-ns oscillator is low, bits 0:1 are captured into a two-bit register by a 6-ns clock; when the 12-ns oscillator is high, bits 2:3 are similarly captured.

The next stage of the serialization process, as shown in Figure 4(a), converts the two-bit fields of data into a serial data stream. A pair of latches (SRL0 and SRL1) whose clocks are 180° out of phase from each other capture the two bits of data, thus creating staggered data-valid windows for presentation to a custom-designed select-balanced selector (SBSEL). At this point, the most crucial step of the serialization process occurs. The design goal for the controlling input of the SBSEL is to achieve a perfectly balanced 50% duty-cycle oscillator signal. The symmetry of this signal will ultimately define the bit time or baud interval which is transmitted on the STI link.

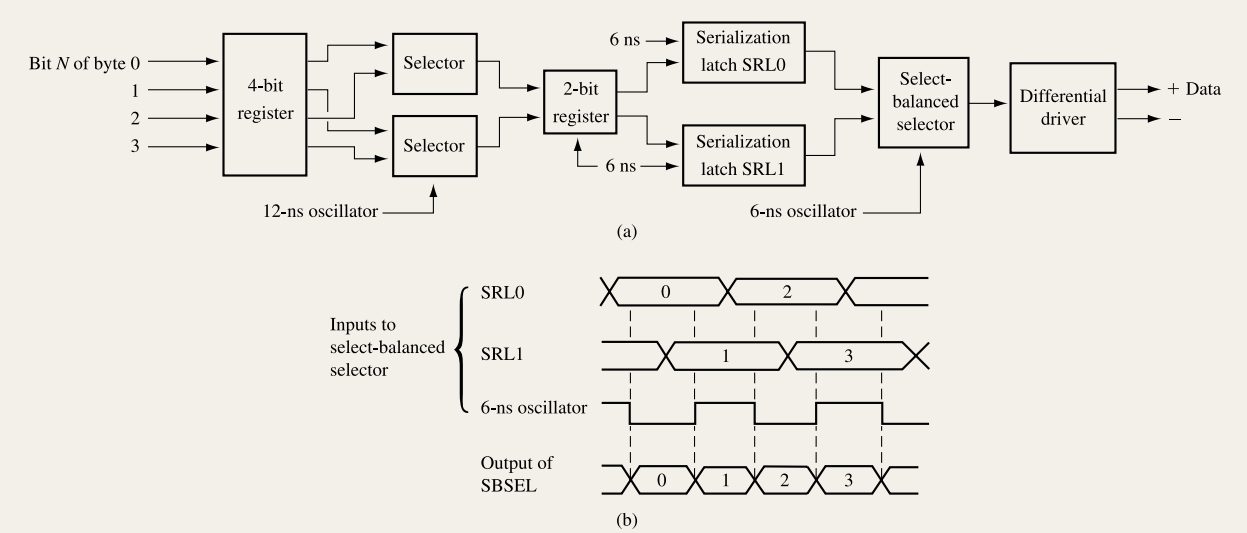

Figure 4

(a) STI physical send macro one-bit logic; (b) timing diagram for the final stage of the serialization process.

From the logic timing diagram in **Figure 4(b),** it can be seen how the STI is able to transmit data at 333 MB/s while using an oscillator that cycles at one half of the data rate. The serialization process is now completed by the PSM, and serial data is sent to the differential drivers for transmission on the STI cable. It should be understood that any deviation in the 50% duty cycle of the 6-ns oscillator signal will be perceived as data-bit jitter by the STI PRM at the other end of the cable.

- *Skew between the data bits*

In Figure 3, which shows the one-bits of the PSM, all data bits launched on the link have a common 167-MHz oscillator signal as the controlling input to the SBSELs inside the one-bits. The first source of data-bit skew is the inevitable variation in the 167-MHz oscillator distribution network to each one-bit. The next contributing factor is the difference in chip wiring from each PSM one-bit to its respective differential driver. Other contributors to skew are the differences in the chip wiring from the differential driver outputs to the C4 pads of the chip, differences in the substrate wire lengths from the C4 pads to the module I/O pins, differences in the board wire lengths from the module I/O to the connector pins, and, finally, differences in the conductors within the connector and cable itself. A similar set of contributors must be considered at the receiving end of the STI link. Naturally, silicon variations along the entire path must also be taken into consideration.

There is a distinct difference between data-bit skew and jitter. Although they are caused by similar sources, jitter results from the time-varying component of these sources and applies only to an individual data-bit path. Skew, on the other hand, is defined by the variations between one data-bit path and another. The largest contributor to skew is the physical difference in wiring paths from bit to bit. Jitter results primarily from noise sources that cause variation of silicon delays and from variations in transmission-line delays.

Transmission of electrical signals

The STI transmission system employs differential signaling with a swing of approximately 1 V for each phase of the signal (2 V peak to peak differentially). The common-mode voltage for the system is nominally 1.25 V. A typical path for the signal would include wiring on a chip and an MCM or SCM, the pins associated with the module, some circuit wiring on a card or board, a cable connector, and a length of cable, with a similar arrangement at the other end of the cable. In the case of the STI, the connector is a 50-pin SCSI connector, while the cable comprises 22 sections of 28-gauge “twinax” (twin-wire axial cable).

Twinax is ideally suited for differential signal propagation with a consistent differential impedance and low skew within a pair. Ten sections of twinax are used for transmission in one direction, while another ten sections are used in the opposite direction. The remaining sections are used by the system to sense cable connections. The

overall cable bundle is surrounded by a shield to reduce radiation from the cable. Finally, the entire cable is surrounded by a protective insulating outer layer.

The differential receiver, to a first order, responds only to differential signals at the two receiver inputs and is able to handle a common-mode excursion of about ±0.5 V. Further discussion of the receiver common-mode voltage is given in the context of receiver jitter.

As with all copper-wire transmission systems, there are physical limitations on the distance and speed of the system, and the STI is no exception. The primary problem with wire transmission systems is distortion of the signal due to high-frequency loss, as well as phase distortion caused by the cable and card wiring. Here the primary culprit is the well-known skin-effect loss in the card and cable wire [3]. Thus, the higher-frequency components of the signal are more attenuated than the low-frequency components, which in turn distorts the signal and leads to problems in decoding the data and maintaining synchronization. To a lesser extent, the reflections caused by discontinuities in the signal path (i.e., at connectors, module pins, vias, terminations, etc.) can also lead to signal degradation.

In general, the signal distortion problem is analyzed by modeling the various sections of the package. Transmission-line models that include skin-effect and dielectric losses have been developed for the card wiring and cable [3]. Models have also been developed for the connectors, the module, and the pins associated with the module package. The entire package is simulated in the time domain using the ASX electrical simulation program [4].

Jitter as the source of bit errors

Simply put, the job of the STI is to deliver error-free data over the link. From the standpoint of electrical and signal transmission, several effects can interfere with error-free data transmission. The first and perhaps the easiest to understand is noise induced from adjacent signal lines. If sufficient noise signal is induced when the data signal is sampled, a bit error can result. The STI is wired to minimize induced signal noise—differential pairs are separated as much as possible from one another on the cards and boards, coupling between the pairs is minimized in the cable bundle, and any induced coupling from vias, pins, module wiring, etc. is kept small enough to prevent significant coupling problems.

The most likely source of error in the STI link is the cumulative effect of jitter. Jitter is defined as the cycle-to-cycle variation in the "zero crossings" of the data. There are numerous sources of jitter in the STI link, and the combined effect of these various sources has the greatest potential to cause a bit error. The overall effect of jitter is to move the desired bit sufficiently in time to cause it to be incorrectly sampled by the clock.

• Data-pattern-dependent jitter	1440 ps
• Delay-line resolution	500 ps
• Delay-line jitter	90 ps
• PLL jitter	100 ps
• Transmit data duty cycle variation	(RMS) 100 ps
• Transmit clock duty cycle variation	(RMS) 100 ps
• Data-receiver-induced jitter	(RMS) 50 ps
• Clock-receiver-induced jitter	(RMS) 50 ps
• Data-coupled-noise jitter	(RMS) 400 ps
• Clock-coupled-noise jitter	(RMS) 400 ps
• Sum	2130 ps
• RMS sum	587 ps
• Total (sum + RMS)	2717 ps

Figure 5

Jitter budget for 333-MB/s STI.

Many jitter sources must be considered in the analysis of the link. All of these sources are analyzed and then placed in a "jitter budget" to ensure proper operation of the link (see **Figure 5**). First and perhaps foremost is the effect of high-frequency losses on the signal. The phase and frequency response of the card wire and cable spread the bits out in time, causing intersymbol interference. Intersymbol interference, in turn, causes the effective boundaries, or zero crossings, of bits to be dependent on the previous data. This problem, in which the present bit is affected by the previous bits, is well known as data-pattern-dependent jitter [4]. By itself, data-pattern-dependent jitter is not capable of causing a bit error, but it can be a contributing cause when its effects are combined with the other sources of jitter in the STI link.

To assess the amount of data-pattern-dependent jitter, simulation is done in the time domain using frequency-dependent models for each segment of the signal transmission path. The actual driver and receiver circuits are included in the simulation in order to assess their performance under anticipated conditions. The link is driven by various data patterns—some random, some fixed—to evaluate the jitter of the entire link. The resulting waveforms are "folded over" in time at a multiple of the bit rate to produce an "eye diagram" [5]. From these diagrams, the amount of data-pattern-dependent jitter for the system can be analyzed.

Although their effects are minimized by design, coupling between differential pairs on the package can contribute to jitter. As previously mentioned, the coupling that occurs comes from parallel module wiring, inside connectors, adjacent vias and pins, etc. The induced voltage will alter the zero crossing in time, and the

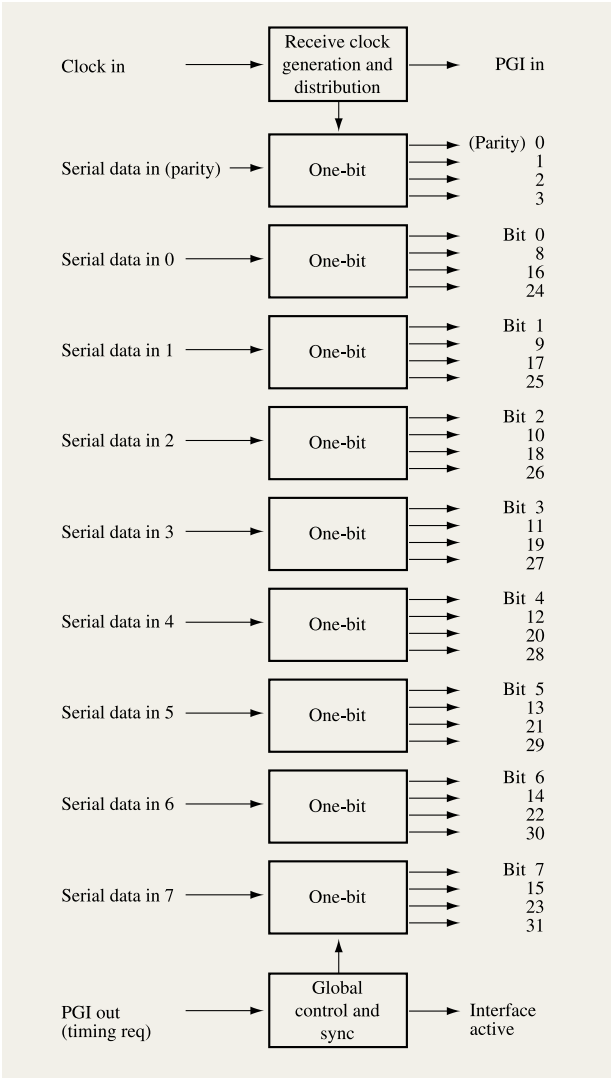

Figure 6

STI physical receive macro.

cumulative effect of coupling throughout the net can cause jitter on both the data and the clock nets; therefore, these items are included in the jitter budget.

Extensive measures are taken to strive for a perfect 50% duty cycle on the clock and data signals. As mentioned previously, any deviation from this goal contributes to the jitter. Since we cannot achieve perfect signal symmetry, we include these items in the jitter budget.

Finally, within-pair skew, or the difference in the arrival time between the two signals in the differential pair, will affect the performance of the receiver. It is virtually impossible to keep the propagation delay of the two phases of the differential signal equal throughout the entire path. Differences in wiring, imperfections in the cable, etc. will contribute to the within-pair skew, which will cause the voltage at which the two phases become equal (zero differential voltage) to vary from cycle to cycle. The varying crossover voltage will affect the delay through the receiver, causing jitter. Stringent wiring rules attempt to control the within-pair skew to some reasonable number—usually a small fraction of the rise and fall times at the receiver. This skew effect is factored into the design of the receiver. The crossover transitions will usually not occur at the nominal common-mode voltage, and the receiver must be able to handle the varying voltage crossover. From a differential standpoint, it should be pointed out that the timing from crossover to crossover is not affected by the within-pair skew. Jitter contributions to the data and clock caused by within-pair skew result primarily from the propagation-delay variation through the receiver in response to the varying crossover voltages.

Jitter budget

The jitter budget in Figure 5 shows that there are many causes of jitter other than the packaging. These other sources arise in the STI electronics. For instance, there is a fixed increment of time resolution in the delay line which leads to a resolution error in determining the exact edges of the data bit. This in turn leads to an error in locating the center of the data bit. The error in the bit center contributes to jitter by leaving less margin for data and clock perturbations in time. This is the origin of the delay-line resolution component of the jitter budget. Power-supply noise on the chip also affects the delay-line resolution, since power-supply noise modulates the delay through the delay line. This component is referred to as delay-line jitter. These two jitter sources are particular to the STI logic and will be better understood after the operation of STI is presented.

With all of the jitter components identified and analyzed, an acceptable method of combining all of the components must be developed. Perhaps the simplest method would be to add all of the various components. Since the components probably have truncated worst-case distributions, this would guarantee that the system would work. However, that approach is far too conservative. For example, not all links will have the worst-case coupled jitter combined with the worst-case duty cycle for both clock and data, etc. On the other hand, items such as data-pattern-dependent jitter, delay-line resolution, delay-line jitter, and PLL jitter are very common. A more desirable approach is to add the data-pattern-dependent jitter, delay-line resolution, delay-line jitter, and PLL jitter. The remaining components are root-mean-square (RMS) summed together and then added to the sum

above. The criterion for concluding whether or not the link will function is to require that the total jitter be less than the bit interval. Figure 5 shows a total jitter budget of 2717 ps, less than the 3000-ps bit interval.

STI physical receive macro

Thus far we have discussed the STI transmit function, differential signaling, the electrical characteristics of the copper cables, and the components contributing to skew and jitter. We next describe what occurs in the STI at the receiving end of the link. The physical receive macro (PRM) receives the signals from the cable, resynchronizes the data signals with the transmitted clock, and deserializes the byte-wide data bus back into a word-wide interface for delivery to the logical receive macro (LRM). Again, Figure 2 illustrates this dataflow.

Figure 6 shows a high-level block diagram of the PRM; the figure shows the bit-slice nature of the PRM, which is similar to that of the PSM. The logic is designed for a single bit and then replicated eight times. The details of the PRM one-bit are shown in **Figure 7**.

Serial data first enters the bulk delay chain, where delay is added to each data bit on an individual basis, with the primary goal being to center the data-bit window for sampling within the fine delay chain. The elements of both delay chains are custom-designed, balanced inverters.

Next the serial data enters the fine delay chain. One way to view the fine delay chain is as a memory in which each element contains slightly different phase information about the data signal. Three address registers address this memory element—early edge, late edge, and data. These registers select particular elements of the delay chain for output to be sampled and then presented to the edge detectors.

The edge detectors locate and lock onto the edges of the data bit to be sampled. After the edges are located, the position of the data sample is calculated as the center between the early edge and late edge addresses. This identifies the ideal point at which to sample the incoming serial data stream.

The STI has two distinct modes of operation: timing mode and operational mode. During timing mode, both ends of the STI link participate in the self-timing of the interface. The PSM transmits a predetermined pattern called the timing pattern, which consists of a string of zeros followed by a periodic one. The PRM expects to see the timing pattern while in timing mode, and the bulk delay is adjusted in this mode. After timing mode is complete, the bulk delay is frozen; however, the edge detectors continue to update the early, late, and data address for the fine delay chain. Through the feedback path surrounding the fine delay chain (i.e., data sample, edge detection process, filter, and finally edge address register update), the STI is capable of maintaining a lock

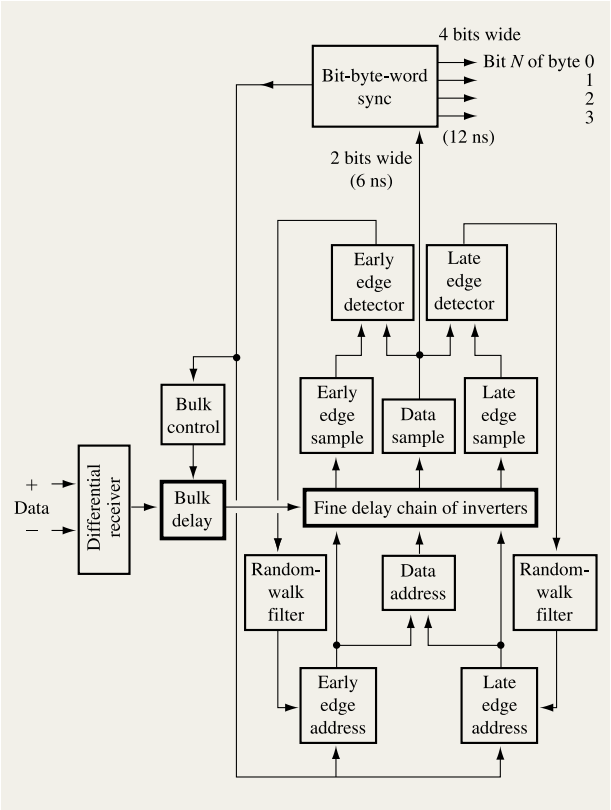

Figure 7

STI physical receive macro one-bit logic.

on the incoming serial data stream. As the environment of the system changes because of temperature and voltage variations, the STI is able to track these variations, thereby maintaining error-free sampling of the incoming serial data stream. The details of each of these functions are presented next.

- *The problem of receiving skewed data*

Consider the data transmission system shown in **Figure 8**. Data is launched by a clock from a set of latches (six shown) on the transmitting end of the link through a set of off-chip drivers (OCD) into a set of conductors. In this case, the launching clock is sent along with the data. At the receiving end of the conductors, a corresponding set of off-chip receivers (OCR) processes the incoming signals, which are then captured by receiving latches using the received clock.

Although the data for each conductor is launched at the same time, the arrival times at the receiver are skewed because of the variations of the individual transmission paths, as previously discussed. In addition to illustrating the physical link, Figure 8 also shows the hypothetical

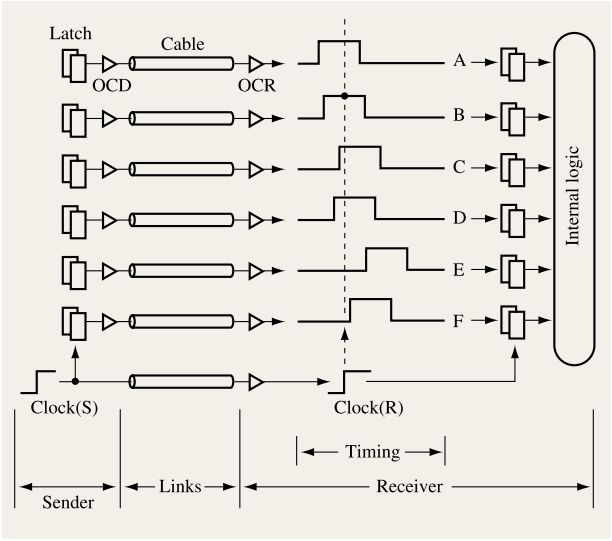

Figure 8

Data transmission problem.

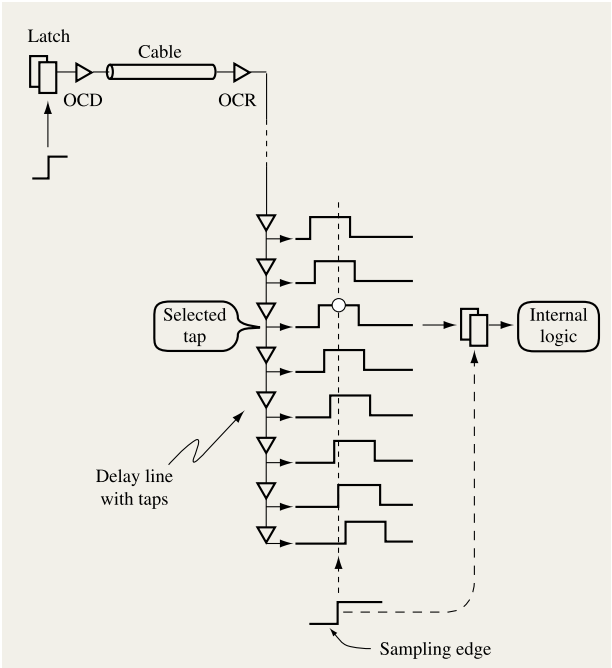

Figure 9

Basic concept of phase selection.

skewed arrival times at the receiving end of the link. The timing diagram shows that the signal on Conductor A arrives at the receiving chip the earliest, and the signal on Conductor E arrives the latest. If nothing is done, Conductor A is sampled late and Conductor C and Conductor D are sampled early. Conductor E and Conductor F are sampled incorrectly. Only Conductor B happens to be properly centered with the sampling edge of the received clock. Clearly, the received signals must be properly phase-aligned with respect to the sampling edge of the clock in order to be accurately captured into the receiving latches. Jitter also exists on the data edges, which further compounds the problem. The STI provides a solution to this problem by automatically aligning the incoming data bits with the received clock.

- *The STI solution*

The STI concept is novel and straightforward. Each data input signal to the receiver chip is fed to a delay line with multiple taps, as shown in **Figure 9**. The delay line contains many identical delay elements, with the output of each delay element representing a unique phase of the incoming data signal. At the core of the STI is the phase-selection logic, which is responsible for selecting the preferred phase of the data bit to be sampled—at the center of its data-valid window. Once identified, a built-in servo-mechanism locks the data phase selection and makes dynamic adjustments to maintain the lock; hence the name "self-timed interface."

Central to the phase-selection process is centering the data-valid window with respect to the sampling clock edge. Consider a data pulse as it propagates down the delay line, as illustrated in **Figure 10**. First the edges of the data bit must be found by means of the edge-detection process, to be discussed later in detail. As shown, the leading edge of the C2 clock aligns with the trailing edge of the data window and identifies the corresponding tap as tap *E*. In a similar way, the leading edge of the C2 clock aligns with the leading edge of the data window, which occurs at tap *L*. Finding the center of the data window becomes a simple matter of choosing a tap midway between tap *E* and tap *L*. Tap *D* is chosen as the tap from which data is sampled, because the sampling edge of the C2 clock falls directly into the center of the data window. This process is also known as bit synchronization. Tap *E* is located earlier in the delay chain than tap *D*; hence it is called the early guard-band (EGB) tap. Similarly, tap *L* is located later in the delay chain, and is thus called the late guard-band (LGB) tap.

- *Description of the receive logic*

Fine delay line and bulk delay line

The fine delay line consists of 32 precisely controlled delay stages. A convenient way to implement these delay stages is with custom-designed inverters having symmetrical rise

and fall times. The delay-line taps mentioned earlier represent not just one single inverter output, but the outputs of a pair of adjacent inverters. These inverter pairs are selected by an address register in a "leapfrog" manner. For example, if the address is "0000," inverter 0 and inverter 1 are selected as the first pair. When the address is advanced to "0001," inverter 1 and inverter 2 (not 2 and 3) are selected as the second pair, and so on. The selected pair presents a true output phase and a complement output phase of the input data. The reason for selecting adjacent inverter pairs is explained shortly.

As mentioned earlier, the principal function of the bulk delay line is to add sufficient delay so that the data bit window is centered within the fine delay line. As shown in Figure 10, it is important that when the sampling edge (positive-going edge of C2 in this example) occurs, the entire data window of bit(i) must be stored within the fine delay line. In the event that this condition is not met, such as in the case when half of bit(i) and half of bit(i + 1) are stored within the fine delay line at the time of sampling, an additional amount of bulk delay must be added so that the entire bit(i) will be shifted into the fine delay line, as shown in **Figure 11**. Like the fine delay element, the bulk delay element is a custom-designed inverter having symmetrical rise and fall times. The bulk delay line has multiple taps to provide delay from a fraction of a bit-time (1/10, 1/5, etc.) up to slightly more than one bit-time. This is done to ensure that, for all possible phase relationships between clock and data, the data can be centered and sampled in the fine delay line.

For a fast-process chip, which requires more inverters to hold a bit-time, our design goal is to ensure that no fewer than 1.5 bits are stored in the fine delay line. This allows plus or minus one quarter bit-time for dynamic tracking of the data edges. For a slow-process chip, which requires fewer inverters to hold a bit-time, our design goal is to ensure that no fewer than eight inverters exist between the data edges stored in the fine delay line. Figure 11 shows the fine delay line and bulk delay line cascaded together. The process of determining the proper amount of bulk delay is carried out during timing mode, which is discussed in detail later in this paper.

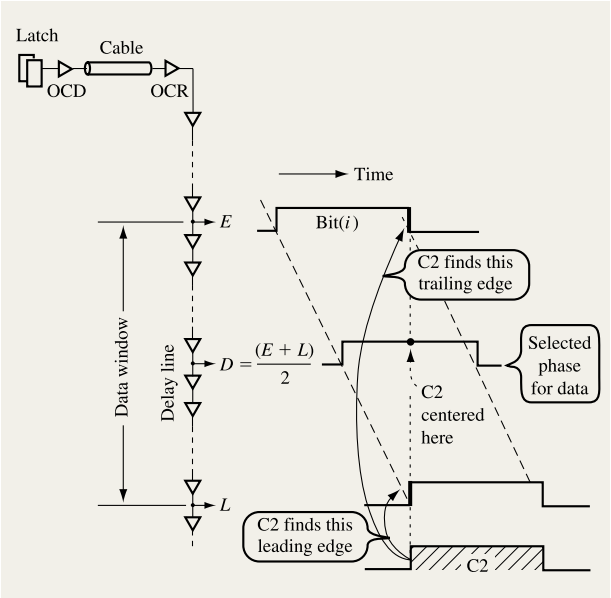

Figure 10

Data window centering.

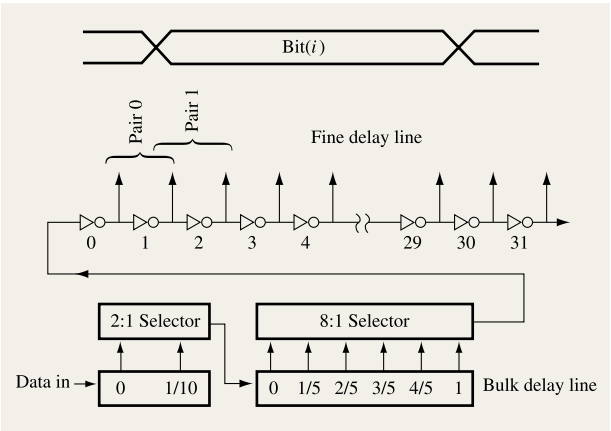

Figure 11

Fine delay line and bulk delay line.

Early and late guard-band tap selection

Figure 12 shows a more detailed block diagram of the STI one-bit, with particular attention paid to the phase-selection logic. The fine delay line has 32 output taps, which are divided into three groups. The first group, comprising inverters 0–15, is assigned to the EGB; the second group, comprising inverters 16–31, is assigned to the LGB. A third group of inverters, 8–23, is assigned to the data tap and shared with the upper half of the EGB and the lower half of the LGB, respectively. The three groups of inverter outputs are fed as inputs to three pair-selector blocks. The pair-selector blocks are built using a series of custom-designed, balanced-output multiplexors. One pair-selector on the left chooses the EGB tap, or tap E, which is controlled by the early address register. Another pair-selector on the right selects the LGB tap, or tap L, whose selection is controlled by the late address register. A pair-selector block receives a four-bit address code from the address registers. The pair-selector will

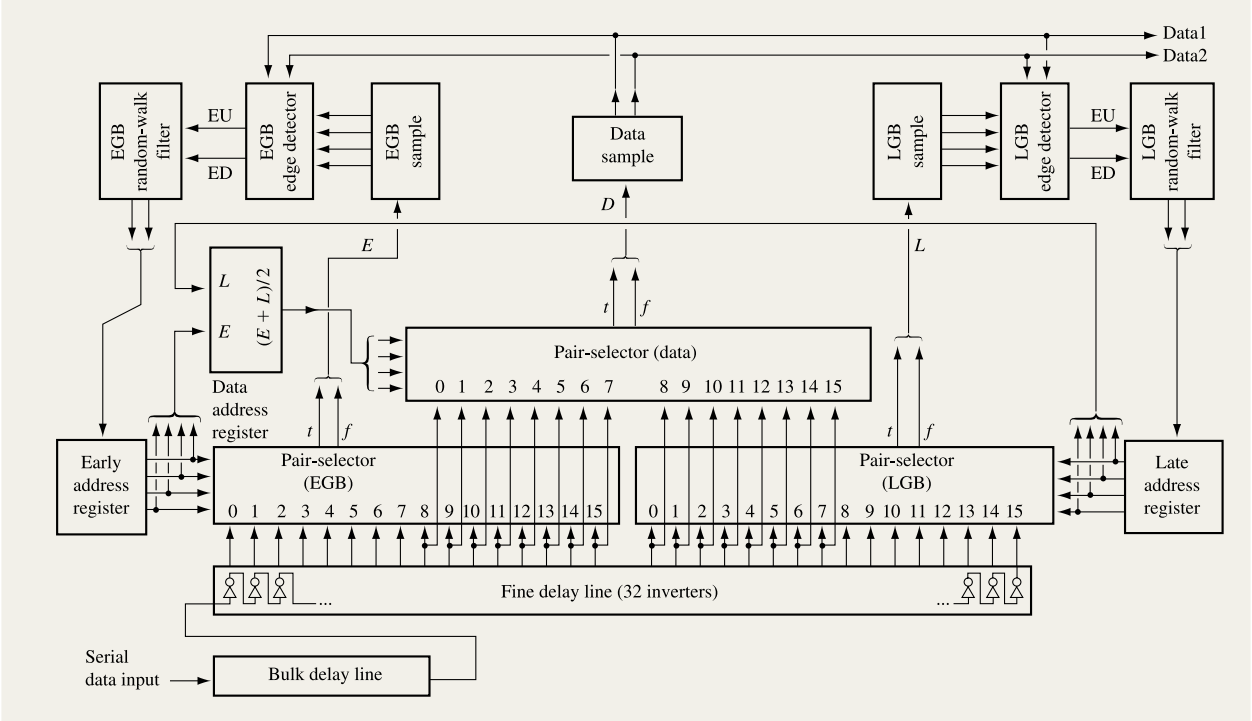

Figure 12

Detailed block diagram of the physical receive macro one-bit.

select a pair of inverter outputs from the 15 possible pairs formed by the 16 inverters in the EGB, LGB, and data groups. The inverter pairs chosen by a pair-selector are labeled *t* and *f*. As we explain shortly, the edge detectors use both true and false taps of EGB and LGB at the same time.

Data tap selection

The EGB address and the LGB address correspond to the fine delay line taps, where the trailing and the leading edges of the data window occur. The data tap control logic, also shown on the left in Figure 12, averages the EGB and the LGB addresses to generate a third address for the data tap. Tap *D* is obtained by feeding the calculated data address into another pair-selector. Therefore, finding the optimum tap within the fine delay line from which to sample data becomes a matter of finding the early (tap *E*) and late (tap *L*) edges of the data bit and then finding their midpoint.

Unlike the EGB and LGB, only one tap of the true/false data samples is used at a time by the edge detectors. The reason for collecting both true and false data samples, even though only one data sample can be used at a time, is to smooth the process of changing from one tap to the next. These two phases are selected alternately as the address register selects "leapfrog" pairs of inverters. When the data address changes, the STI never switches to a new tap immediately for fear of corrupting the data. The next tap is always available and stable before it is used.

Sampling the serial data

The three selected pairs of true and false taps are fed to their respective sampling blocks to generate samples for the edge detectors. The three sampling blocks shown in Figure 12 are the EGB sample, the LGB sample, and the data sample. The data sampling block also serves as the first step of the deserialization process, converting the serial data stream into a two-bit-wide data stream labeled as Data1 and Data2. The edge-detection process is an iterative feedback process that will eventually converge on the proper settings for the three address registers. Before describing the edge detector, it is necessary to describe how the STI samples the incoming serial data. Details of the sampling logic are shown in **Figure 13**. A serial data stream is fed to the inputs of both an L1/L2 (master/slave) latch and an L2* (pronounced L2 star) latch. It is given that the bit time of the data stream is one half of the

clock cycle. As stated earlier, the PSM transmits data using both edges of the clock; hence, two bits of data must be captured during each clock cycle, one using the positive edge of the clock and the other from the negative edge of the clock.

The received-link clock shown in Figure 13 is connected to the input of a clock splitter that generates two complementary-phase clock outputs, C1 and C2, which drive master/slave latches. C1 is an out-of-phase clock gating the master (L1) part of the L1/L2 latch. C2 is an in-phase clock gating both the slave (L2) part of the L1/L2 latch and the L2* latch. The L2* latch is like an L1/L2 latch except that the L2 portion is directly accessible and the L1 portion of the L2* need not be used.

In the timing diagram of Figure 13, the positive-going edge (marked X) of the clock points to the "current" data bit Data(i). When the C1 clock is closed while Data(i) is active, Data(i) is sampled or captured into the L1 portion of the L1/L2 latch. When C2 is on, the data latched in L1 will be transferred to the L2 portion of the L1/L2 latch. As a result, Data(i) appears at the output of the L1/L2 latch beginning at the positive-going edge of C2 and remains valid for an entire cycle, labeled as Cycle(i).

The L2* latch works somewhat differently. When the C2 clock is high, the input data is flushed directly to the output of the L2* latch, resulting in part of the Data(i) appearing as "noise" at the L2* output. Since the C2 clock is closed while Data($i + 1$) is active, Data($i + 1$) is captured into the L2* latch and remains there for the remainder of Cycle(i). The "noise" at the output of the L2* latch can be removed by adding another L1/L2 latch (shown as SRL2). To match the timing, SRL1 is added to the output of the L1/L2 latch. As a result, two stable data samples, Data(i) and Data($i + 1$), appear at Cycle($i + 1$).

In summary, Data(i) is sampled by the positive-going edge of the clock, and Data($i + 1$) is sampled by the negative-going edge of the clock. The data samples of different bit-times in the serial data stream (one half cycle apart) are now lined up in the same cycle so that in Figure 12, the data sample block converts the serial data into two-bit-wide parallel data, Data1 and Data2. As shown, Figure 13 assumes that Data in is from the "true" tap of the fine delay line. If Data in were from the "false" tap of the fine delay line, inverters would be added between L2 and SRL1 and between L2* and SRL2. It should be mentioned that the EGB and LGB samples use the same L1/L2 and L2* configuration to sample data from the fine delay line. Also worth noting is that by design, the EGB and LGB samples operate at the transitions of the data to be captured. Therefore, standard design techniques are employed to avoid metastability hazards.

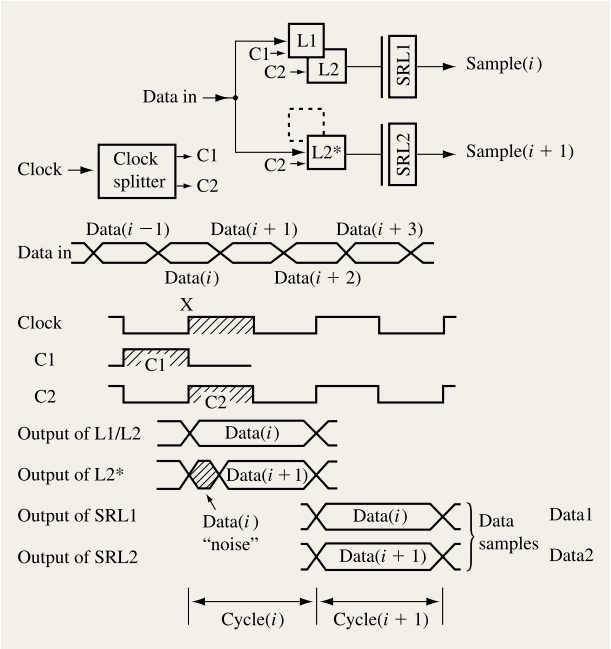

Figure 13

Sampling logic.

The edge-detection process: Transition detector

Understanding how STI captures data from a serial data stream and aligns two samples in the same cycle, we are now able to use the sampled data in downstream logic. **Figure 14** shows an example of a transition detector which is used by the edge detector described later. Let S(i) be the sample of Data(i) and S($i + 1$) be the sample of Data($i + 1$). The late edge (LE) exists when S($i + 1$) differs from S(i). Therefore, performing an exclusive-OR operation between these two samples will produce a signal, LE_exist, indicating the presence of the late transition.

Similarly, the early edge (EE) exists when Data($i - 1$) is different from Data(i). The sample of Data($i - 1$) labeled as S($i - 1$) is generated by the cycle preceding the one corresponding to Data(i). The added latch SRL3 will bring that sample into the same time slot as S(i) and S($i + 1$) so that a comparison can be readily made. Samples S(i) and S($i - 1$) will be connected to another exclusive-OR to produce a signal, EE_exist, indicating the existence of the early edge. It is imperative that the edge detectors be able to distinguish the difference between the case in which either EGB or LGB is incorrectly positioned and the case in which the serial data simply did not change from bit to bit.

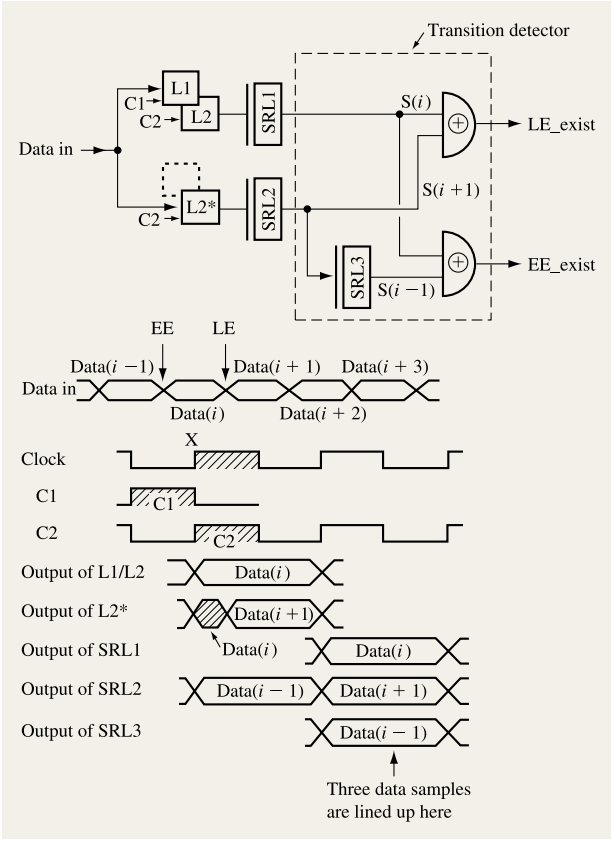

Figure 14

Transition detector.

The edge-detection process: Edge detector

Detecting where the edges occur is the critical step in the phase-selection process. Figure 12 shows two edge detectors: the early guard-band (EGB) edge detector and the late guard-band (LGB) edge detector. Each edge detector is independent from the other and has as its inputs one data sample and two adjacent-edge samples. Under typical operating conditions, the data edge would be trapped between the adjacent-edge inverters. The principle of operation is the same for both, but the implementations are slightly different. Let us look at the LGB edge detector to illustrate the concepts.

In the timing diagram of **Figure 15**, there are four cases to be considered. They differ in terms of the relationship between the serial data and the sampling edge of the clock (C2). Figure 15 shows a data stream with all zeros followed by a single one which is followed by all zeros again. The single one is a half cycle wide and is represented by the shaded area. *D* is the data tap (top middle of Figure 12). *L* is a selected true/false pair of samples from the LGB (assume that all necessary inversions have been introduced). Tap *L* has more delay than tap *D*, as shown in both Figure 12 and Figure 15. The objective is to locate an edge within the half-cycle window. For example, for the leading edge of the one, the window of search is between the two vertical dotted lines.

Consider Case 1 of Figure 15 in which data transition edges (EE and LE) are found. By this we mean that tap *L*, which represents a pair of adjacent phases, is properly chosen such that the transition of one phase falls immediately to the left side of the sampling clock edge and the transition of the next phase falls immediately to the right side of the sampling clock edge. Since there are two sampling edges, the positive-going edge of the C2

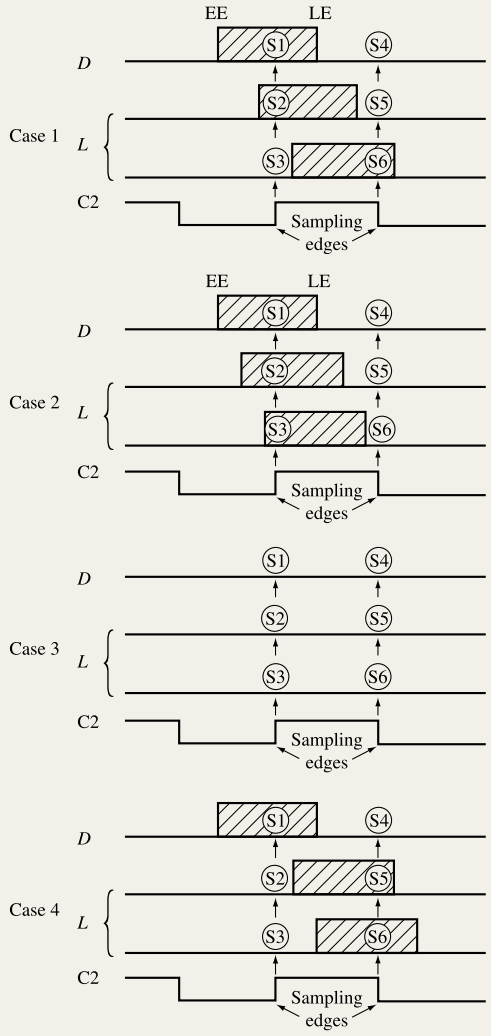

Figure 15

Concept of edge detector.

clock and the negative-going edge of the C2 clock, there are six tap samples for the edge-detection process; these are labeled in circles in Figure 15. Sample 1 (or S1) is captured by the positive-going edge of the C2 clock and taken from one of the two phases (true or false) of tap *D*. Samples 2 and 3 (S2 and S3) are also generated by the positive-going edge of the C2 clock and taken from the phase pair of tap *L*. Samples 4, 5, and 6 (S4, S5, and S6) are all generated by the negative-going edge of the C2 clock in a like manner. From the timing diagram, we learn that in order for this condition to be met, the samples must have the following characteristics:

Case 1:

- S2 = S1.
- S3 ≠ S1.
- S5 = S4.
- S6 ≠ S4.

Action: None, because the late edges have been found.

If Case 1 is detected, no action is taken by the edge detector because tap *L* has been located such that the data edges fall right on the sampling edges of the clock. Note that in this implementation of the edge detector, all samples are lined up in the same cycle by the sampling logic described earlier. The desired logic comparisons can be readily made by means of exclusive-OR gates.

Consider Case 2 of Figure 15, in which the data transitions (EE and LE) are not found because an earlier tap *L* pair has been selected. The corrective action is to select a later tap, which has more delay, from the fine delay line. This situation has the following characteristics:

Case 2:

- S2 = S1.
- S3 = S1.
- S5 = S4.
- S6 = S4.

Action: The LGB edge detector will generate an Edge_up (or EU) pulse in an attempt to select a tap pair with more delay (i.e., a later tap). This is done by incrementing the LGB address register by one. This action is taken only when transitions exist.

Case 3 of Figure 15 shows the situation for which there are no data transitions in the serial data. The STI receives a constant logic level (either all ones or all zeros) for the duration of the sampling interval of interest. The samples obtained have the same characteristics as in Case 2, but no corrective action should be taken because no edge transitions are found. This is why the action taken in Case 2 must depend on whether or not the serial data has had a transition.

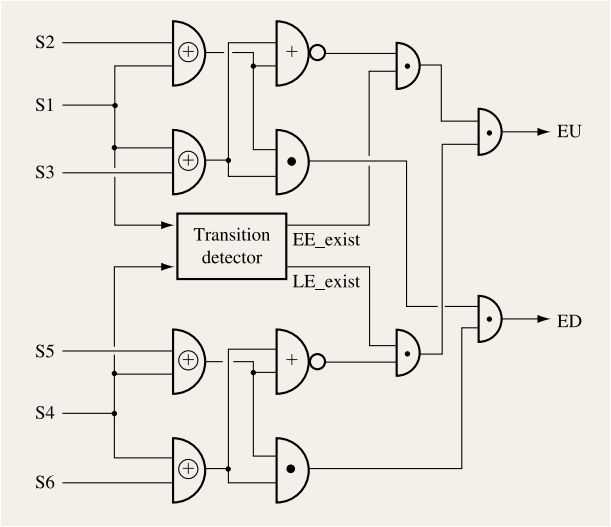

Figure 16

Edge detector.

Case 4 of Figure 15 shows the situation for which a very late tap *L* is selected. The sampling characteristic is as follows:

Case 4:

- S2 ≠ S1.
- S3 ≠ S1.
- S5 ≠ S4.
- S6 ≠ S4.

Action: The LGB edge detector will generate an Edge_down (or ED) pulse in an attempt to select a tap pair with less delay. The goal is to decrement the late-address register by one.

On the basis of this understanding, the LGB edge detector can be constructed. **Figure 16** shows a possible implementation in which six samples are taken to generate either Edge_up (EU) or Edge_down (ED) signals. The same principle of operation applies to the EGB edge detector, the only difference being that tap *L* is replaced by tap *E* (see Figure 12) and the EU and ED definitions are swapped.

The edge-detection feedback process is a matter of taking the proper actions to find the right tap *E* and tap *L* such that Case 1 is always maintained. However, in the presence of jitter (noise) and environmental changes in temperature and voltage, the data edges will drift within

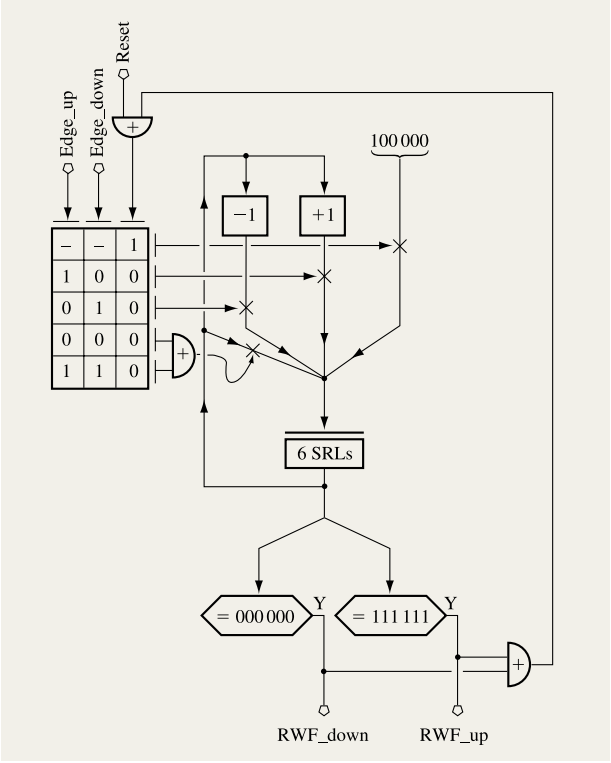

Figure 17

Random-walk filter.

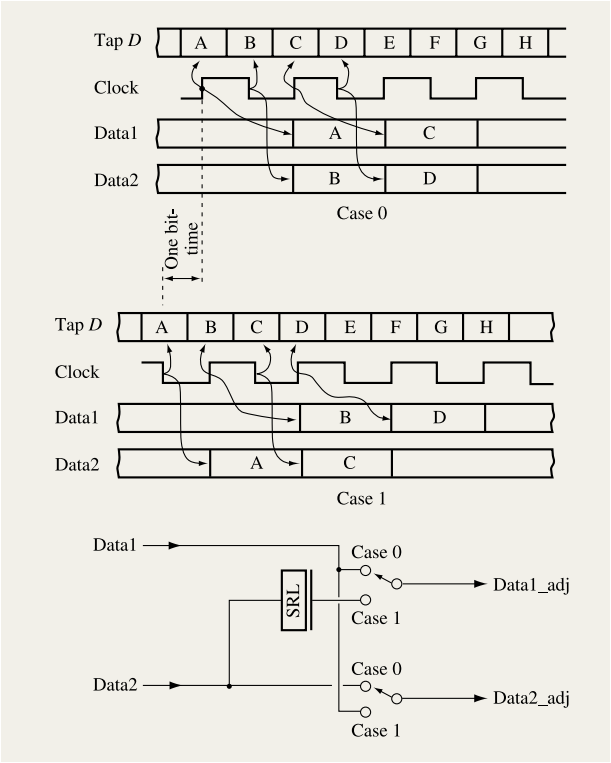

Figure 18

Step one of byte synchronization.

the delay lines, causing the edge detectors to move in and out of the Case 1 situation. Therefore, corrective actions will always be necessary. This dynamic tracking mechanism is desirable, since it guarantees stable data centering, making the system more robust and tolerant of environmental variations.

Random-walk filter

Because data on the STI link is subject to noise, jitter, and environmental changes, it is unavoidable that some edge samples taken at the transition region will be wrong. This will in turn cause the edge detectors to produce incorrect EU or ED signals. It follows, then, that instantaneous adjustment of the address registers is not prudent; rather, an adjustment based on an average of several transitions is in order. A random-walk filter (RWF) is used to smooth out the edge-detector outputs before updating the address registers. An individual RWF is assigned to each edge detector, as shown in Figure 12.

Figure 17 shows a logical view of the RWF consisting primarily of a 6-bit up/down counter. At the beginning of the edge-detection process, the counter is preset to its middle value, b'100000'. It can count up 31 positions or down 32 positions depending on whether Edge_up or Edge_down is received.

If a net of 31 Edge_up pulses have been received since reset, an RWF_up signal is generated. However, if a net of 32 Edge_down pulses have been received, an RWF_down signal is generated. The RWF_up or RWF_down signal will update the corresponding EGB or LGB address register, causing a new tap *E* or tap *L* to be selected from the fine delay line. When the counter reaches either all ones or all zeros, it resets itself to the middle again and produces an UP/DN signal. Thus, the RWF effectively averages and removes jitter from the edge-detector outputs.

Byte and word synchronization

The data-sampling logic (upper middle of Figure 12) deserializes the serial data from tap *D* and places it on two parallel data buses, Data1 and Data2. We assume at this point that the phase-selection logic has already selected the proper tap *D* such that the sampling edge of the clock falls in the middle of its data window. The next question is whether bits on the Data1 and Data2 buses are in the proper order.

Figure 18 shows a serial data stream ABCDEFGH . . . arriving from tap *D* at the data-sampling logic with the transitions of the clock aligned in the middle of the data window. Two possible relationships can exist, depending upon the arbitrary relationship between clock and data for a particular one-bit. Case 0 shows the positive-going edge of the clock lined up with bits A, C, E, and G. Case 1 shows the other possibility, in which the positive-going edge of the clock lines up with bits B, D, F, and H. As we have described earlier, the serial data sampled by the positive-going edge of the clock is put on the Data1 bus, and the serial data sampled by the negative-going edge of the clock is put on the Data2 bus. From Figure 18, it follows that Case 0 and Case 1 yield different results on the parallel data buses.

If we assume that Case 0 is desired, a mechanism must exist so that Case 0 can be derived from Case 1. This is the first step in a process called byte synchronization. From the timing diagram in Figure 18, the required mechanism can be seen: Delay Data2 by one cycle and swap the two buses. A high-level representation of the byte-alignment logic is shown in Figure 18 as an SRL and a pair of selectors. This logic transforms the Data1 and Data2 buses into a pair of new (Data1_adj and Data2_adj) buses capable of producing the correct data-bit order. As mentioned previously, during timing mode, a particular timing pattern is transmitted to help the PRM one-bits identify whether the phase-selection logic is producing Case 0 or Case 1. For example, if the data stream is a repeating pattern of 'X010' (A = *X*, B = 0, C = 1, D = 0, E = *X*, F = 0, G = 1, H = 0, and so on, where *X* can be either 1 or 0), then for Case 0, the Data2 bus is always at logic zero, and for Case 1, the Data1 bus is always zero. Hence, by simply detecting which bus is always zero, both cases can be identified.

The next step in the byte-synchronization process is to account for the anticipated three bit-times of skew between the data bits within the STI link. Nothing can be done to speed up the slower one-bits; therefore, the earlier-arriving bits must be delayed an appropriate amount until they are aligned with the latest-arriving bit. **Figure 19** shows the logic required to perform this alignment, as well as the final step in the deserialization process. The amount of delay, measured in bit-times, is controlled by Switch A and Switch B. Note that Switch A is the same one referred to in Figure 18. Switch A and Switch B are controlled by a two-bit control counter, S0 and S1, within the signature detector.

During timing mode, each one-bit goes through byte synchronization independently. The two-bit control counter is cycled until the signature detector recognizes a good signature (i.e., 'X010') as provided by the timing pattern. An example illustrates this point. Refer to the switch control truth table shown in **Figure 20**. If the slowest bit finds the timing pattern signature with S0:S1 = 00, a bit which is one bit-time faster will set S0:S1 = 01, and the one-bit which is two bit-times faster will set S0:S1 = 10. The fastest bit, potentially three bit-times earlier than the slowest one, will set S0:S1 = 11.

The final step in the synchronization process is word synchronization. Word synchronization is done at a global level by the global synchronization logic, which compares the most significant bit, $D_{out}0$, from each one-bit. This logic compares every cycle to ensure that the X bits of the timing pattern are identical for each one-bit. Switch C of each one-bit is controlled by the global synchronization logic and introduces an additional four bit-times of delay to the one-bits it affects.

An example illustrates this function. Assume that Switch C is initially set to position 0, as shown in Figure 19. If the slowest one-bit sets S0:S1 = 01 instead of 00, the

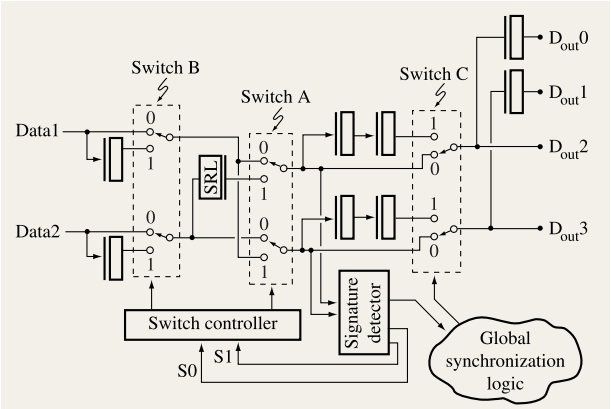

Figure 19

Byte and word synchronization.

(S0,S1)	Switch A	Switch B	Comments
00	0	0	delay = 0 (reference)
01	1	0	delay = +1 bit-time
10	0	1	delay = +2 bit-times
11	1	1	delay = +3 bit-times

Figure 20

Switch control truth table.

fastest bit will wrap back and set S0:S1 = 00. However, this will put the fastest bit ahead of the slowest by four bit-times, or one word-time, because now the fast bit is one bit-time ahead (00 versus 01) plus the three bit-times by which it arrived ahead of the slow bit originally. To achieve word synchronization, Switch C of the fast bit is thrown to position 1, thereby delaying it back into the same word as the slow bit. Similarly, if the slowest bit sets S0:S1 = 10, the fastest bit will wrap back and set S0:S1 = 01. Again, the fastest bit will be four bit-times ahead of the slowest. The global synchronization logic will throw Switch C of the fast bit to position 1 to realign it with the slowest bit.

Summarizing byte and word synchronization, the byte-alignment logic will adjust until the third bit of the 'X010' timing pattern is always one. If this is not the case, it advances the two-bit counter until the condition is attained. The global synchronization logic compares all most significant bits X (one or zero) of all one-bits to verify that they are all the same. If not, the corresponding Switch C will be toggled to position 1 and rechecked.

Timing mode and functional mode

When the system is powered on or reset, the STI must enter timing mode. Timing mode is initiated by the host logic and controlled by a timer in the PRM, in conjunction with state machines in the LRM and LSM. During timing mode, the timing pattern is generated by the LSM and transmitted by the PSM. Timing mode is typically completed in less than two milliseconds, and after a few "handshaking" signals, the STI enters functional mode.

The timing-mode timing pattern as sent from the STI LSM to PSM is

FF 00 FF 00 00 00 FF 00 00 00 FF 00 00 00 FF 00.

This translates to each one-bit seeing

1010 0010 0010 0010.

At the beginning of timing mode, the bulk delay of each one-bit is reset to zero, and the related tap *E* and tap *L* are reset to near the middle of the fine delay line. In order for the edge-detector algorithm to work, tap *E* and tap *L* must never allow more than one data bit to fall between them. For example, the early-address register is preset to 10, and the late-address register is preset to 2. It then follows that the data address register for tap *D* will be calculated as (10 + 2)/2, or 6. As the bit-synchronization process proceeds, tap *E* will "walk" toward the beginning (moving down by ED) of the delay line and tap *L* will walk toward the far end (moving up by EU) of the delay line, until the data edges are found.

It is desirable to have the entire data window centered in the middle of the fine delay line to allow maximum tracking range. To prevent one edge from being too close to the middle and the other edge being too close to either end of the delay line, the EGB range for tap *E* and the LGB range for tap *L* must be defined. If one of the edges is not found, or if the found edge falls outside the specified guard-band ranges, this is an indication that the data window for that particular one-bit is not properly centered inside its fine delay line. If this is the case, a unit of bulk delay (Figure 11) is added for each one-bit that falls into this category. Then, as before, tap *E* is again preset at 10 and LGB is preset at 2, and the bit-synchronization process is repeated once more. If this time the edges are still not found or the found edge still falls outside the specified guard-band ranges, another unit of bulk delay is added. This process is repeated as many times as needed on a per-bit basis until both edges are found and they lie within the specified guard-band ranges. Once timing mode is complete, the bulk delay values are frozen, as are the byte-alignment and word-alignment logic, leaving the fine delay line and edge-detection process to continue into operational mode.

Conclusions

The introduction of STI as the interconnection for the I/O subsystem of S/390 has enabled new interfaces such as ICB, FICON, ATM, Fast Ethernet, and Gigabit Ethernet to be integrated with the IBM legacy channels ISC, ESCON, and parallel channel. With STI as the I/O subsystem interconnection, S/390 CMOS servers now have an I/O subsystem that provides the necessary bandwidth and connectivity to scale with the performance improvements being realized in processor speed.

This paper has presented the design goals and details of the self-timed interface. As the evolution of S/390 CMOS servers continues, so must STI, by exploiting the latest offerings in circuit, cable, and connector technologies to achieve even higher data rates.

Acknowledgments

The authors wish to acknowledge the pioneers of the STI, Frank Ferraiolo and Daniel Casper. Also contributing to the advancement of STI were Richard Jordan, Anthony Perri, Mark Fischer, and Jack Yarolin. We also wish to thank Evan Davidson, Frank Ferraiolo, and Daniel Stigliani for reviewing this paper.

References

1. T. A. Gregg, K. M. Pandey, and R. K. Errickson, "The Integrated Cluster Bus for the IBM S/390 Parallel Sysplex," *IBM J. Res. Develop.* **43,** No. 5/6, 795–806 (1999, this issue).
2. T. A. Gregg, "S/390 CMOS Server I/O: The Continuing Evolution," *IBM J. Res. Develop.* **41,** No. 4/5, 449–462 (1997).

3. A. Deutsch, G. V. Kopcsay, V. A. Ranieri, J. K. Cataldo, E. A. Galligan, W. S. Graham, R. P. McGouey, S. L. Nunes, J. R. Paraszczak, J. J. Ritsko, R. J. Serino, D. Y. Shih, and J. S. Wilczynski, "High-Speed Signal Propagation on Lossy Transmission Lines," *IBM J. Res. Develop.* **34,** No. 4, 601–615 (1990).
4. John G. Proakis, *Digital Communications*, McGraw-Hill Book Co., Inc., New York, 1995.
5. A. Bruce Carlson, *Communications Systems*, McGraw-Hill Book Co., Inc., New York, 1986.

Received March 2, 1999; accepted for publication June 28, 1999

Joseph M. Hoke *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (jmhoke@us.ibm.com).* Mr. Hoke is an Advisory Engineer in the S/390 Connectivity Solutions Development group. He received the B.S. degree in electrical engineering from the University of Illinois at Chicago in 1987 and continued his studies under a university fellowship, receiving the M.S. degree in electrical engineering from Northwestern University in 1989. He joined IBM at Poughkeepsie, New York, in 1989 and has held various technical positions in S/390 I/O development. Mr. Hoke holds several patents used in IBM ESCON and sysplex products and has received two IBM Invention Achievement Awards. He has received an IBM Outstanding Technical Achievement Award for his work on ESCON and another for his contributions to the S/390 G5 Server.

Paul W. Bond *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (pwbond@us.ibm.com).* Mr. Bond is an Advisory Engineer in the Mid-Hudson Valley High-Performance Design Center. He received his B.S. degree in electrical engineering from Rensselaer Polytechnic Institute in 1972 and an M.E. degree in electrical engineering, also from Rensselaer Polytechnic Institute, in 1973. He joined IBM in Kingston, New York, in 1973. He is currently involved with the development of high-speed CMOS serial links.

Tin-chee (TC) Lo *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (tclo@us.ibm.com).* Mr. Lo received the M.S. degree in electrical engineering from Carnegie Mellon University. He joined IBM at East Fishkill, New York, in 1977, working on DRAM development projects. He became a Senior Engineer in 1984 and moved to IBM Poughkeepsie in 1985, working on assignments related to the design of System/390. Mr. Lo has engaged in different technical activities including bipolar and MOS device modeling, n-MOS and CMOS circuit design, static and dynamic memories, ABIST, and various logic design programs for high-end servers. His current interests are in high-speed interconnections and signal propagation. Mr. Lo holds 14 U.S. patents and has published numerous papers and invention disclosures in a variety of areas in the field of microelectronics and logic design. He has also received many technical awards during his career in IBM. Prior to joining IBM, he worked for American Micro-System and Fairchild Semiconductor, both in northern California.

Frank S. Pidala *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (pidala@us.ibm.com).* Mr. Pidala is a Staff Programmer Analyst in the Mid-Hudson Valley High-Performance Design Center. He joined IBM in East Fishkill, New York, in 1969. He is currently responsible for the physical design and release of self-timed interface (STI) macros. Mr. Pidala has received various recognition and informal awards, and more recently he received an IBM Outstanding Technical Achievement Award for his work on G5.

Gary Steinbrueck *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (steinbru@us.ibm.com).* Mr. Steinbrueck is a Senior Engineer in the Mid-Hudson Valley High-Performance Design Center. He received the B.S. degree in electrical engineering from the University of Missouri at Rolla in 1968. He joined IBM at East Fishkill,

New York, and has held a wide variety of technical positions, including thermal engineering, advanced development of power devices, CMOS and bipolar memories, microprocessors, and packaging and product development of bipolar and CMOS logic circuits; he is currently responsible for the design of circuits for high-performance communication links for S/390, Power Parallel Systems, and OEM products. He has received many technical awards, including IBM Outstanding Technical Achievement Awards for his contributions to the IBM S/390 G3 and G5 systems.