The IBM Enterprise Systems Connection (ESCON) channel— A versatile building block

by J. R. Flanagan
T. A. Gregg
D. F. Casper

The IBM Enterprise Systems Connection (ESCON™) environment required the design of a single channel that could be attached to the entire line of Enterprise System/9000™ processors and deliver the performance required by the top of that line. In addition to the channel, other functions were needed, such as the ESCON channel-to-channel adapter. All of these functions were required to be implemented using the same channel hardware. This paper describes the key elements of the IBM ESCON channel design.

Introduction

Several (often conflicting) requirements were placed on the design of the IBM Enterprise Systems Connection (ESCON™) channel:

- *Performance:* The channel for the IBM high-end systems had to provide a high level of performance—not only sustaining the maximum data transfer rate allowed by the ESCON architecture [1], but also performing chaining and block reconnection [2] in a timely manner. A typical approach might have been to implement these time-critical functions in hardware; however, such a design must be customized for a particular interface protocol (the rules for sending and receiving messages) and message structure, and is very intolerant of changes [3].
- *Flexibility:* The ESCON connection strategy required two basic channel types: the native ESCON channel and the ESCON Converter Model 1 channel. These two channels use different message structures and interface protocols. In addition, the ESCON channel-to-channel adapter (CTC) and the ESCON channel-test-vehicle control unit functions were needed. The designers were given the requirement that a single hardware design must

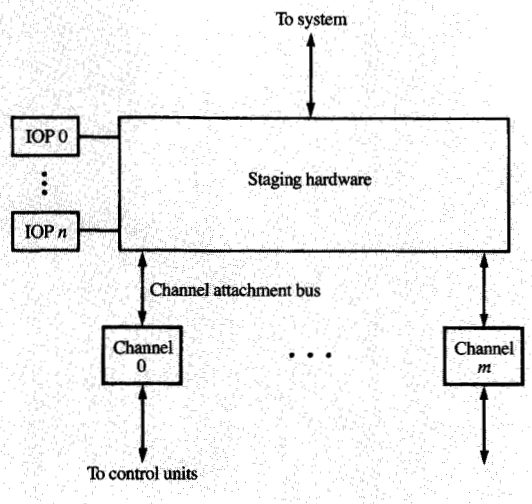

Figure 1

General structure of an ES/9000 channel subsystem.

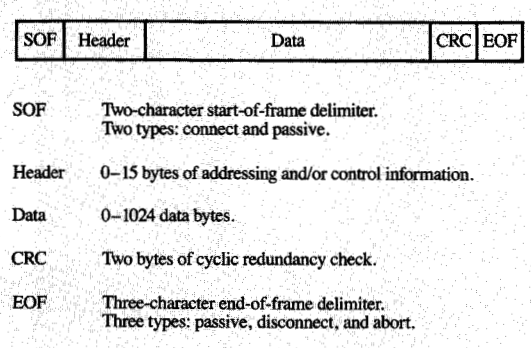

Figure 2

General structure of a data frame.

be able to perform all of the above functions through different code ("licensed internal code"). Also, all of the above interface protocols were being developed concurrently with the channel, so changes had to be accommodated easily. To achieve this degree of flexibility, it was necessary that many functions be implemented in code, which is readily modified. However, a code implementation is inherently slower (and thus poorer in performance) than a hardware implementation [4, 5].

- *Common system attachment:* The channel had to be attachable to every system in the Enterprise System/9000™ (ES/9000™) line, from the smallest Model 120 to the largest Model 900. The system interface was required to use a minimum number of wires yet provide sufficient bandwidth to support the maximum ESCON data rate.
- *Simulated I/O (SIMIO) capability:* Because of the large numbers of channels required on the high-end systems, it is impractical to attach real control units to every channel during system testing or manufacturing testing. There was a requirement to run real channel programs on all the channels without using any external control unit attachment.
- *Low-cost, high-reliability package:* Again, because of the large number of channels required, it was necessary to minimize the physical size of the channel. The channels also had to meet the very stringent reliability, availability, and serviceability requirements of the top-of-the-line processors.

The following sections discuss how all of these requirements were met in the design of the ESCON channel.

Background

- *Channel subsystem overview*

The basic function performed by an ES/9000 channel subsystem is to manage the transfer of data and control information between system storage and the attached I/O devices, thus freeing the central processors (CPs) of this burden. The program requiring I/O must 1) build, in system storage, a channel program, consisting of one or more channel command words (CCWs), that describes the data areas and provides the I/O-device commands to be used; 2) build an operation request block (ORB) specifying the channel-program address and other parameters; and 3) issue a Start-Subchannel instruction that specifies the I/O device. The channel subsystem then queues and executes the requested I/O operation and informs the program of the final status of the operation by means of an I/O interruption [2, 6].

The general structure of an ES/9000 channel substem is shown in **Figure 1.** It consists of three main elements:

- *Integrated off-load processors (IOPs)* perform all the communication with the CPs and maintain the work queues for the channel subsystem. They perform path selection, and they retry when busy conditions occur. They also perform initialization functions and aid in recovery from catastrophic channel errors.
- *Channels* are responsible for the execution of channel programs. They initiate channel programs, perform data transfer and chaining operations as appropriate, and provide final status information to the IOP. They also continue disconnected operations when so requested by an I/O device [2].

- *Staging hardware* provides communication paths among the IOPs, the channels, and the remainder of the system. Each channel is attached to the staging hardware via its own *channel attachment bus.* All communications with the other elements of the system (including maintenance function communications) are performed over this bus.

- *I/O interface*

The ESCON channel connects processors to ESCON control units, Directors [7], and converters by means of a full-duplex fiber optic serial link that operates at 200 megabits per second (Mb/s). This link is called the I/O interface. The data-encoding scheme is the character-oriented 8B/10B code described by Widmer and Franaszek [8]. This code converts 8-bit bytes into 10-bit characters that are transmitted on the link; in addition, several 10-bit control codes (called K-characters) are transmitted.

Communication on the interface is performed with sequences of characters called frames. **Figure 2** shows the structure of the frames used by the ESCON channel. The idle sequence, transmitted whenever no frames are being transmitted, is the repetition of one of the K-characters (K28.5). (Using a 10-bit idle character keeps the receiver in character synchronism at all times, which simplifies design of the receiver.)

All frames begin with an SOF (start-of-frame) delimiter comprising two K-characters. Two types of SOF delimiters are used: passive and connect. These delimiters control the connection state of the ESCON Director™. Data characters follow the SOF delimiter. The first group of data characters is the frame header; the second is the data field. All frames have a header, and frames used to transfer data also have a data field. In either case, the hardware that generates and receives frames can handle variable-length headers and data fields. The next two data characters of all frames comprise the CRC (cyclic redundancy check). Following the CRC is the EOF (end-of-frame) delimiter, which consists of three K-characters. There are three types of EOF delimiters: The first two are the passive and disconnect delimiters, used to control the connection state of the ESCON Director. The third EOF delimiter is the abort, used to signal the receiver to discard the current frame. The idle sequence resumes after the EOF delimiter. At least four idle characters must be transmitted between frames.

In addition to the normal idle sequence, a group of modified idle sequences, transmitted continuously, is provided to signal special link states. The states include ESCON Director disconnect, link failure, and link off-line. The modified idle sequences are composed of the idle character alternating with a data character [9]. The ESCON channel provides facilities for generating and receiving any of the 256 possible modified idle sequences (one for each of the 256 data characters).

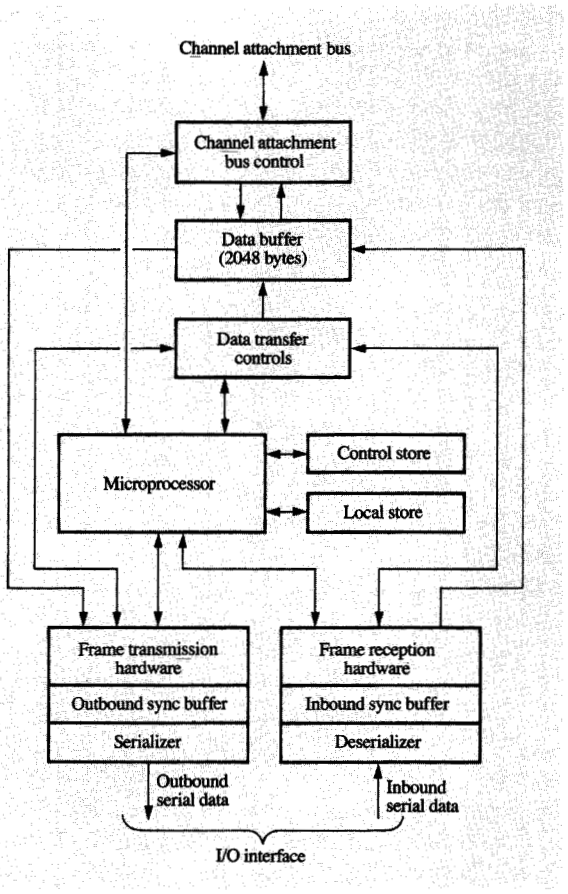

Figure 3

ESCON channel elements.

Key elements of the channel design

At the heart of the ESCON channel, shown in **Figure 3,** is the microprocessor, which controls all of the channel elements. The channel attachment bus connects the channel to the system. The 2048-byte data buffer is used as a temporary staging area for data as they move between the I/O interface and the system. The data-transfer controls direct the transfer of data between the data buffer and the I/O interface. The frame transmission and frame reception hardware are independent elements. Each of these elements is described in the following sections.

- *Channel microprocessor*

All IBM large-system channels since the 1970s have been implemented using some form of microprocessor. However, the requirements for the microprocessor in the ESCON channel were somewhat more demanding than those of previous channels. Not only did two channel types have to be implemented, but the channel hardware

also had to be able to act as a control unit (e.g., the ESCON channel-to-channel adapter). This required complete control of the frame transmission and reception hardware by the microprocessor in order to allow different types of frames to be generated and received. Also, previous channel processors handled only one task at a time. As is discussed later, the ESCON channel processor had to be able to control the data transfer to or from system storage and to or from the I/O device simultaneously, and had to switch tasks very rapidly between the two. Finally, the ESCON architecture uses far more complex recovery algorithms than does the System/370™ OEMI parallel interface [10]. For example, the channel is required to request a retry of the current command under certain error conditions. Providing enough control storage in each channel to contain all of these recovery routines in addition to the primary routines would have been too costly. Therefore, a mechanism for paging non-time-critical code from main storage of the system was required.

The processor developed to meet these requirements is custom-designed, optimized for efficient control of the channel hardware facilities. It has a simple one-byte-wide data flow and has access to virtually all of the channel elements. The processor has a one-byte ALU, a shifter, and 256 bytes of working storage for general use. A 16-kilobyte local store provides space for control information for a large number of devices. Local store also holds trace data for problem determination purposes.

The basic processor instruction is a 38-bit microword. Thirty-five of the bits are divided into seven fields, each of which can be decoded independently into *micro-orders* to perform a different function. The last three bits are parity bits. Two of the fields control the ALU functions and the gating of data to and from the processor. One field is normally used as the next address. The remaining fields are used for various purposes, such as modifiers, array addresses, constant values, and micro-orders for setting and resetting control latches. Normally, one microword is executed each channel cycle. Since each of the fields can specify an independent function, multiple functions can be performed in each cycle.

The *writable control store (WCS) array* provides space for 8192 microwords. WCS is logically divided into 64 segments of 128 words each. The first 56 segments are *static*; that is, the code there remains resident after system initialization. These segments are used to contain the mainline code critical to performance. The last eight segments are *pageable*; these segments may be loaded from system storage during channel operation, permitting code that is not performance-oriented, such as recovery and link initialization, to be loaded when required. The paging function is performed by code, with hardware assistance.

Conditional branching is provided to test various conditions in the channel. A two-, four-, or eight-way branch is implemented by replacing the appropriate number of low-order bits of the next WCS address with bits representing the selected branch conditions. (*WCS address* means the address of the microword that is executed next.) Branch prediction is used to improve performance: If the actual branch condition agrees with the prediction made when the branch was coded, the next word is executed in the next cycle; if not, a one-cycle penalty is incurred while the proper word is fetched from WCS. A four-level link stack is provided for subroutines; however, at least one level must be reserved for preemptive traps (see the following).

Traps are the mechanism by which certain events are made known to the code without the code having to test for their occurrence explicitly. When a trap occurs, the WCS address is set to that of the first word of the segment that corresponds to the trap number. There are two types of traps: *normal* and *preemptive*. Normal traps can occur only when the code issues a specific micro-order that allows them. This is typically done when the code has completed a task and is going to idle while waiting for the next event.

Preemptive traps provide a single level of interrupt (i.e., a second preemptive trap cannot occur during a preemptive trap). They may occur any time they are not inhibited by the code. When a preemptive trap occurs, the current WCS address is pushed onto the stack. When the code in the trap routine has been completed, it issues a RETURN order, which causes the former WCS address to be popped off the stack and execution to resume at the interrupted point. Only two registers must be saved and restored by the preemptive trap routines, and other functions can typically be performed in the microwords that do the saving and restoring. Consequently, the preemptive traps allow very rapid task switching. This feature is exploited for data transfer, during which the normal traps service the storage interface, while preemptive traps break in as required to service the I/O interface.

The microprocessor is of key importance to the channel design. Its programmability provides the flexibility to implement multiple architectures, while the combination of multiple functions per cycle, branch prediction, and rapid task switching provides performance that equals or exceeds that of most all-hardware designs (see the data-transfer section for further discussion of performance).

- *Channel attachment bus*

Prior to the introduction of the ES/9000 line of compatible systems, it had been standard practice to design a new and completely different channel for each new system. Thus, the interface between the channels and the rest of the

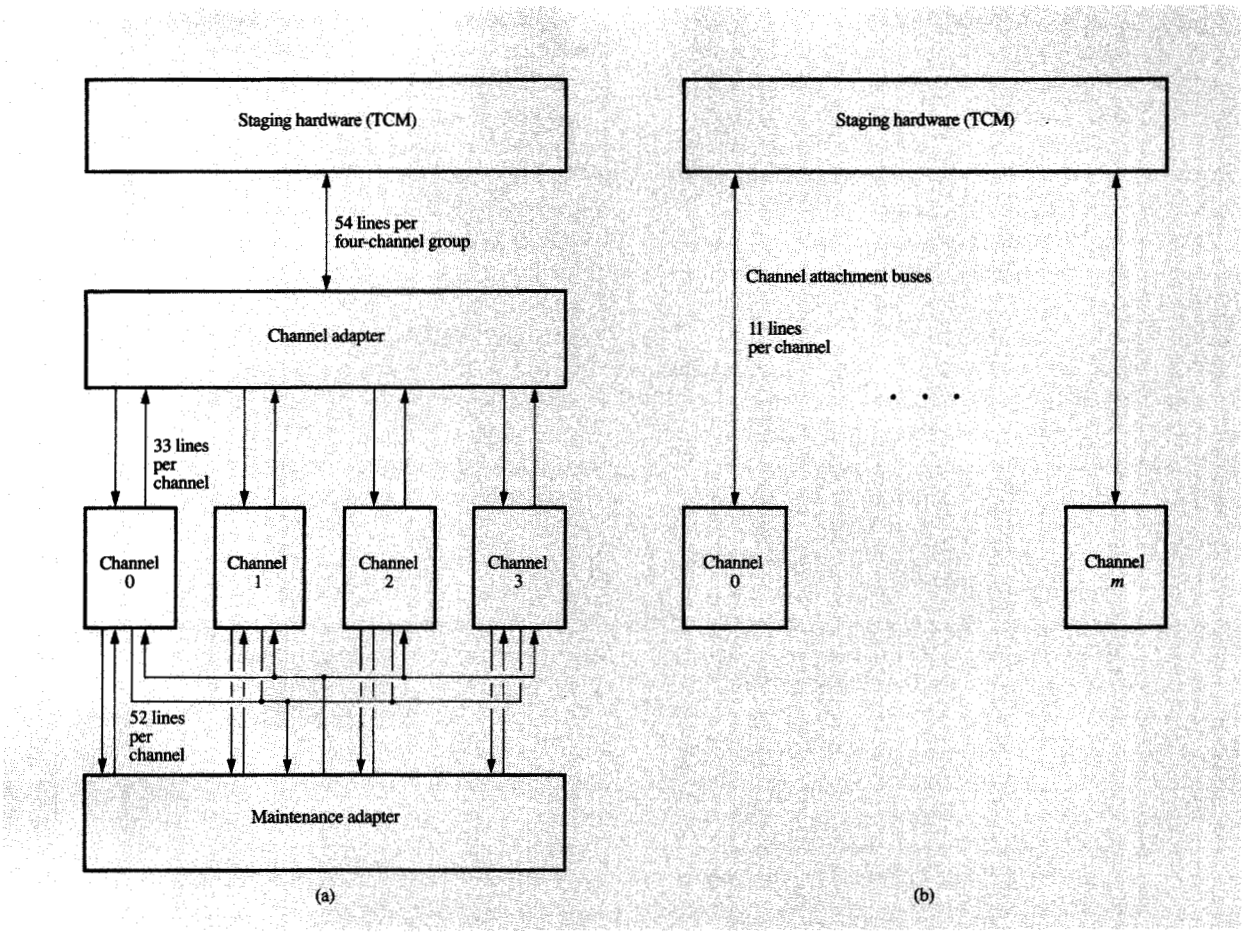

Figure 4

Channel attachments: (a) Early 3090 channel attachment; (b) ESCON channel attachment.

system was customized to the needs of the particular system. An example of such an interface, found in the early models of the 3090™ system, is shown in **Figure 4(a)**. Each of four channels is attached to a channel adapter by means of a 33-wire interface consisting of two unindirectional byte-wide buses and a multiplicity of individual control signals. The adapter multiplexes the data and control information from the four channels onto a 4-byte-wide bidirectional bus that is attached to the staging hardware. A separate maintenance interface, consisting of an additional 52 signal lines, is required for functions such as scanning, loading code, and logging trace arrays. Thus, a total of 85 wires are required to attach each channel to the system.

This structure was not considered acceptable for the ESCON channel, for a variety of reasons. The configuration could not provide sufficient bandwidth for each channel to sustain the maximum data rate of the ESCON link. This was primarily because of the multiplexing of the four channels using logic running at the same cycle time as the channels themselves. As a result, the multiplexing had to be done with the high-speed bipolar logic of the staging hardware, where the cycle time is approximately one fourth that of the channels. This required that each channel have a separate interface to the staging hardware. However, the requirement for an 85-wire interface per channel far exceeded the number of available signal pins on both the channel card and the thermal conduction module (TCM) [11] of the staging hardware. In addition, the 85-wire interface contained many signals, for maintenance functions, that were common to multiple channels. This made isolating errors to a failing channel very difficult.

The channel attachment bus, shown in **Figure 4(b)**, was developed to solve these problems. It provides a high-bandwidth communication path between each channel

and the staging hardware, using only ten signal lines and a clock line. This bus is used for both normal and maintenance functions; a separate maintenance interface is not required. The ten signals consist of an 8-bit byte, a parity bit, and a data-continue signal, which allows the staging hardware to pause during the transfer of data if its buffers become temporarily unavailable.

When a bus is inactive, the staging hardware controls six of the eight bits and the parity bit, and the channel controls the remaining two bits. The staging hardware always controls the data-continue line.

When the staging hardware uses the bus to communicate with the channel, it first determines that the bus is inactive, then places an encoded request on its portion of the bus in the next cycle, called the *request cycle*. In most cases, all of the information required for unsolicited requests from the staging hardware to the channel can be contained in this encoded value, so no additional bus cycles are necessary. If an additional cycle is required to pass the entire message, this is also specified by the encoded value passed during the request cycle.

When the channel requests use of the bus, it loads an internal bus-control register with the appropriate bus command and sets a latch. When channel hardware determines that the bus is inactive, it activates both of the bits it controls during the following cycle (the channel request cycle). If the six bits controlled by the staging hardware are inactive on the request cycle, the channel places the contents of the control register on the bus during the next cycle, called the *command cycle*. Additional *data cycles* may follow, depending on the command.

The staging hardware always has priority for use of the bus. If the channel, during its request cycle, detects a concurrent request from the staging hardware that requires control of the entire bus, the channel deactivates its request and presents the request again when the bus becomes idle. In this case, the staging hardware maintains its request during the following cycle. If during its request cycle the channel detects a concurrent request from the staging hardware that does not require control of the entire bus, the channel accepts the request and maintains its request during the following cycle.

The maximum data rate any channel can sustain is limited by the rate at which its system interface can move data between its buffers and system storage. The protocols implemented on the channel attachment bus allow the transfer of 128 bytes in only 147 cycles, which includes the overhead of passing the storage address to or from which the data are to be moved, and receiving an acknowledgment that the operation was completed successfully. This allows a bandwidth in excess of 20 megabytes per second (MB/s), which is more than adequate to provide the maximum data rate of the ESCON I/O interface.

Maintenance functions are also performed using the bus. To set the channel into maintenance mode, the staging hardware issues a request in its request cycle. This stops the channel clock. Thereafter, one of the bus bits is used to hold the channel in maintenance mode, while the remainder of the bits are used for signaling resets, providing data in, data out, and clock lines for scan rings [12], etc. The channel leaves maintenance mode on detecting all bits inactive on the bus. If the channel detects a catastrophic error that forces it to stop its own clock, it signals this condition to the staging hardware by activating only one of the two bits it controls.

• *Sync buffers*

Attachment of the ESCON channel to the system is simplified by running the channel synchronously to the system. The channel and system clocking rate is controlled by an independent oscillator in the processor and is not synchronous to the ESCON I/O interface, which has a fixed bit rate of 200 Mb/s. This difference between the channel and I/O clock speeds is handled by the channel sync buffers.

Asynchronous first-in-first-out buffers are used to synchronize the arrival and departure interface data with the channel clock. Two buffers are used: the outbound sync buffer and the inbound sync buffer (see Figure 3). The outbound buffer is between the frame-transmission hardware and the serializer, while the inbound buffer is between the deserializer and the frame-reception hardware. The primary function of the sync buffers is to store data temporarily while metastabilities [13] caused by the asynchronous sampling of signals are resolved. Placing these buffers close to the I/O interface minimizes the number of asynchronous signals required for sending and receiving data. The placement of these buffers also maximizes the amount of logic that can be clocked synchronously to the rest of the system. Developing synchronously clocked logic is easier than developing asynchronously clocked logic, because it allows the most effective use of hardware design tools such as simulation, timing analysis, clock generation, and test pattern generation.

The outbound sync buffer has an additional function used during certain recovery actions in the ESCON channel that cause the channel clock to be temporarily stopped. When the channel clock is running, all of the characters transmitted on the link (including the idle sequence) are supplied by the frame-transmission hardware and are sent through the outbound sync buffer. When the channel clock is stopped, a conventional sync buffer loses its source of characters and transmits unintelligible data. When these unintelligible data are received by the other end of the link, a link failure condition is detected. This is an unacceptable situation, because a link failure causes a

reset of the attached control units, which may lead to a loss of data. This problem is solved by a feature of the outbound sync buffer that detects the termination of the channel clock and automatically generates the idle sequence [14].

When the outbound sync buffer switches between its own automatically generated idle sequence and the character stream supplied by the frame-transmission hardware, the running disparity of these two sources must be the same. (Running disparity is the difference between the total number of 1s and 0s transmitted. A positive running disparity indicates that more 1s than 0s have been transmitted, and a negative running disparity indicates that more 0s than 1s have been transmitted.) The 8B/10B code achieves dc balance by choosing between two alternative 10-bit encodings of an 8-bit byte according to the running disparity of the bits transmitted on the link. The disparity of an encoding is the difference between the number of 1s and 0s in that code. Some of the encodings from an 8-bit to a 10-bit character have only one value, which has a zero disparity. The rest of the encodings have two alternatives: one with six 1s and four 0s (positive disparity) and one with four 1s and six 0s (negative disparity). When the running disparity is positive and the byte to be transmitted has two alternative encodings, the 8B/10B encoder chooses the one with negative disparity; when the running disparity is negative, the alternative with positive disparity is chosen. The idle character (K28.5) has two encodings: one with positive disparity and an alternate with negative disparity. These two idle character code points are sent alternately when the idle sequence is transmitted.

The outbound sync buffer keeps track of the running disparity of the character stream supplied by the frame-transmission hardware. When the channel clock stops, the sync buffer chooses the first idle character of its own automatically generated idle sequence according to the running disparity calculated following the last character supplied by the frame-transmission hardware. When the channel clock is restarted, the sync buffer delays switching to the character stream supplied by the frame-transmission hardware until the running disparity of its own automatically generated idle stream equals the running disparity of the first character supplied by the frame-transmission hardware. Through this process of stopping and starting the channel clock, the receiver at the other end of the link sees no disparity errors and does not detect a link failure condition.

- *Frame-transmission hardware*

The frame-transmission hardware is the key channel component providing the flexibility described in the Introduction. In addition to achieving the full data rate for both the native ESCON channel and the ESCON converter protocols, the hardware can generate special

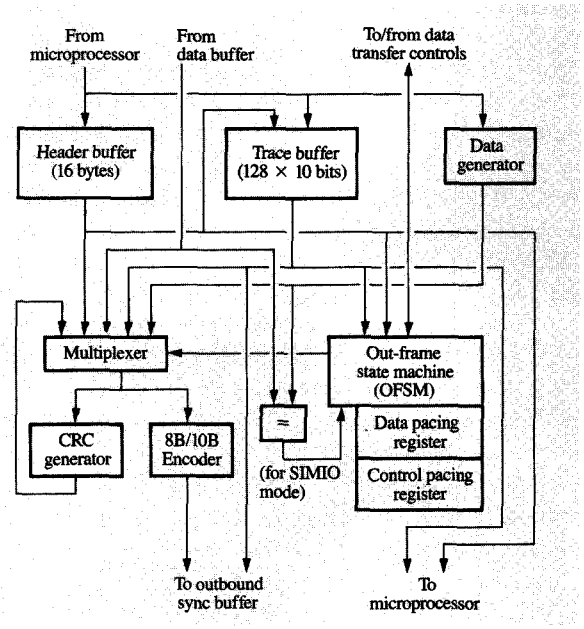

Figure 5

Frame-transmission hardware.

test frames for SIMIO, diagnostics, simulation, and the ESCON channel test vehicle. The channel test vehicle is an ESCON channel with code that emulates an ESCON control unit. The flexibility of this frame-transmission hardware has saved the cost of developing additional functional and test hardware and reduced the overall product development time. The major elements of the frame-transmission hardware are shown in **Figure 5**, and its various modes of operation are described in the following paragraphs.

To initiate a frame, the code first updates either the header buffer or the trace buffer. (The header buffer is used when generating normal frames used in the native ESCON channel and ESCON converter protocols, while the trace buffer is provided for the test modes to insert erroneous frames into the normal stream.) The first 15 bytes of these buffers are reserved for the frame header information, and the 16th byte is a control byte specifying the frame attributes, which include the types of delimiters and the length of the frame header. With all of the frame attributes in a single control byte, the code has only one byte to update when initiating a frame.

After the header buffer or trace buffer is updated, the code starts the out-frame state machine (OFSM), which controls the frame-transmission hardware. The OFSM reads the control byte from either the header buffer or the trace buffer, starts the frame by transmitting the start-of-frame delimiter specified in the control byte, and then

transmits the header. After the header is transmitted, the OFSM examines another bit in the control byte to determine whether data should be appended to the header. (Frames with data following the header are called data frames, and frames with no data following the header are called control frames.) Finally, the CRC is transmitted, followed by the end-of-frame delimiter, which is also specified in the control byte. As transmission proceeds, the OFSM selects the source of the bytes by controlling the multiplexer.

Including the control byte with the header data in the header buffer and the trace buffer speeds frame generation by allowing overlap of frame transmission with frame preparation. The code can update the control byte of the next frame while the data field, CRC, and end-of-frame delimiter of the current frame are being transmitted. The maximum data rate can then be realized, and the frames are transmitted with the minimum number (four) of idle characters between them.

The ESCON channel can transmit data faster than the data can be received by some of the attached control units; to solve this problem, the ESCON architecture provides several methods of limiting the data transfer rate. One of these methods, called data pacing, requires the transmitter to insert extra idle characters between data frames. The frame-transmission hardware automatically inserts these extra idle characters and frees the code from having to determine when to start the next data frame. Along with data pacing, specified by the architecture, a control-pacing function (not part of the architecture) is also implemented. This control-pacing function is used by the ESCON channel test vehicle and SIMIO to regulate the rate of data-request frames, which are control frames. The data-pacing and control-pacing registers determine the minimum number of idle characters that must be inserted between frames. When the code starts a new frame, the OFSM automatically delays frame transmission until the proper number of idle characters have been transmitted. The OFSM selects between data pacing and control pacing by comparing the type of the previous frame (control or data) to that of the current frame. Data pacing is used if both frames are data frames; control pacing is used for all other combinations.

The transmission of a data frame cannot start until all data for that frame are available. The frame-transmission hardware automatically starts the frame when the data are available and frees the code from making this determination. The assembled frame leaves the multiplexer and enters the 8B/10B encoder. The 10-bit encoded characters leave the encoder and enter the outbound sync buffer, which is discussed in the section on sync buffers, above.

The trace buffer provides functions that give the ESCON channel much of its flexibility. Providing useful information for problem determination is its primary function. As frames are transmitted, they are stored in the trace buffer, which contains 16 bytes for each of the last eight frames transmitted: the control byte and the first 15 bytes. In another mode of operation, the trace buffer is used by the SIMIO function as the source of information for frame generation. As described in the section on I/O simulation, below, the first 16 bytes of the trace buffer provide the control and header information, and the last 64 bytes can be the data source for data frames.

The ESCON channel test vehicle, diagnostics, and simulation use the trace buffer in yet another mode, which is called 10-bit mode. The microprocessor loads the buffer with 10-bit encoded characters. When the microprocessor begins frame generation, the OFSM reads the encoded characters from the buffer, replacing the idle sequence and bypassing the encoder and CRC generator. A hardware disparity control delays transmission of the 10-bit characters from the trace buffer until after the next idle character with positive disparity has been transmitted. After the 10-bit characters from the trace buffer have been transmitted, the idle sequence resumes with a negative-disparity idle character. Any valid or invalid bit stream can be generated. This control of the disparity of the bit stream leads to reproducible results of diagnostic tests. The ESCON channel test vehicle, driven by special channel-test software, uses 10-bit mode to predictably inject errors into the channel under test to determine whether the proper error-reporting and error-recovery actions are being performed. The diagnostic code uses 10-bit mode when the serial output is "wrapped back" to the serial input, electrically or optically. The diagnostics generate error sequences to determine whether the errors are being properly detected by the frame-reception hardware. Simulation test cases also use the diagnostics to demonstrate the correctness of the frame-reception hardware. This capability of the trace buffer avoided the necessity of developing special test hardware and writing separate simulation test cases and diagnostic code.

The ESCON channel test vehicle and SIMIO both require the generation and verification of data patterns within data frames. The data generator register meets this requirement by generating a simple sequence (i.e., 0, 1, 2, 3, $\cdots$) or a pseudorandom pattern. The ESCON channel test vehicle uses this register as the source of data for frames sent to the channel under test. It also checks the validity of data from the channel under test by comparing the data received to the data generated by this register. The SIMIO function uses the data generator similarly; it is described later, under the heading I/O simulation.

• *Frame-reception hardware*

The frame-reception hardware [15] complements the frame-transmission hardware by recognizing all frames that

follow the rules described in the I/O interface section above and by detecting and characterizing link errors, all of which can be simulated by the frame-transmission hardware. The division of functions performed by hardware and code was chosen to keep the code uninvolved in the mechanics of frame reception while giving the code the maximum information, encoded in a compact form, in error situations. To this end, the frame-reception hardware automatically decodes the 10-bit characters, checks for character synchronism, detects modified idle sequences used to signal special link states, detects start-of-frame and end-of-frame delimiters, checks CRC, examines frame headers, and stores frame headers and data. As the hardware performs these functions, it collects information describing their progress and informs the microprocessor when appropriate. The frame-reception hardware (**Figure 6**) is controlled by the in-frame state machine (IFSM).

The deserializer converts the bit stream received from the link into 10-bit groups that are sent over a 10-bit parallel bus through the inbound sync buffer to the 10B/8B decoder, where they are decoded into data bytes, special control characters, and invalid characters (code violations). The output of the decoder is sent to the character sync detection logic, CRC checker, and IFSM.

When these 10-bit groups are aligned on character boundaries, the 10B/8B decoder detects valid characters. When the character alignment is not on character boundaries (i.e., the 10-bit groups contain bits from two adjacent characters), invalid characters are detected. The character sync detection logic calculates the frequency of invalid characters to determine whether the deserializer has the correct character alignment. The frequency of invalid characters is calculated by hardware specified by the ESCON architecture [16, 17]. Briefly, there are two counters: One counts valid characters and the other counts invalid characters. When the ratio of valid to invalid characters is less than 15, the character sync detection logic indicates that the deserializer does not have the correct character alignment.

Character synchronism is achieved by adjusting the character alignment of the deserializer. A signal controlled by the microprocessor is sent to the deserializer instructing it to discard one bit of the incoming serial bit stream, thus changing the deserializer character alignment by one bit. The discarding process continues until the alignment of data from the deserializer is on character boundaries.

When the character sync detection logic determines that the frequency of invalid characters is high enough that the character alignment of the deserializer may not be correct, it causes a microprocessor trap. Since large noise bursts on the link may damage many characters, transforming them into invalid characters, the character sync logic may falsely detect a loss-of-character-sync condition even

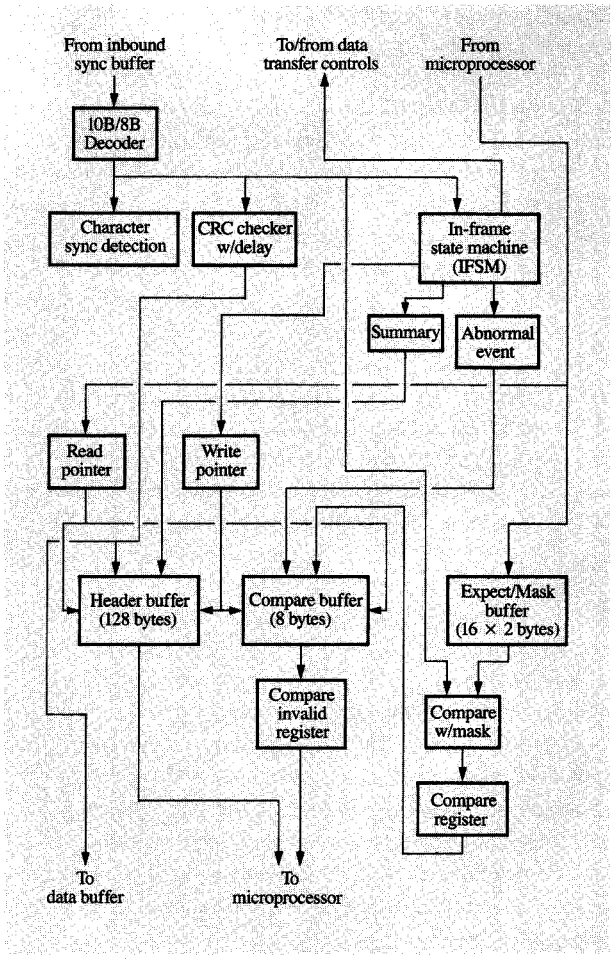

Figure 6

Frame-reception hardware.

though the deserializer is still in the proper character alignment. Because of this effect of a large noise burst and because of the relatively long time it takes to readjust the deserializer character alignment, the code does not immediately start to readjust the deserializer character alignment after the microprocessor is trapped because of a loss-of-character-sync trap. Instead, it waits for about 10 μs (2000 bit times) before examining the signal from the character sync detection hardware that indicates loss of character sync. If the character sync signal is off after this wait time, the link has received a noise burst, and no deserializer character realignment is required. If the loss-of-character-sync signal is still on, the code starts the deserializer character alignment procedure.

The alignment procedure consists of repeatedly discarding one bit from the incoming serial bit stream and waiting for about 3 μs (600 bit times) before checking the

loss-of-character-sync signal. Eventually, either the proper deserializer character alignment is established and operations continue, or a time-out is encountered, indicating a link failure. In either case, the code logs the loss of character sync and includes information describing the duration of the condition and the number of bits that were discarded in order to reacquire sync. This information is used to determine the performance of the inbound link. Indeed, during testing of the channel, this information proved very useful in correcting a defect in the phase-lock loop of the deserializer.

The CRC checker receives data from the decoder, verifies the contents of all frames, and supplies data to the header and data buffers. The CRC bytes are not written into the buffers. Because the frame length is generally unpredictable and because the end is indicated by the end-of-frame delimiter only after the receipt of the two CRC bytes, the CRC checker has a 2-byte delay that prevents writing the CRC bytes into the buffers.

The output of the 10B/8B decoder is also sent to the IFSM logic, where the sequence of characters is examined. This logic controls loading the header buffer and the compare buffer (discussed below), reading the Expect/Mask buffer (also discussed below), and setting status information describing what was received on the inbound interface. The reception of anything on the inbound interface of a frame, modified idle sequence, error, or any of certain other sequences is called an *event*; each event causes an entry into the header buffer and compare buffer.

During an I/O operation, most of the fields within the received frame headers can be anticipated by the ESCON channel, and very few of these fields change from frame to frame as the operation proceeds. This characteristic of the architecture is exploited by the Expect/Mask buffer. As frames are received, they are automatically compared with bit patterns stored in this buffer. At the beginning of each I/O operation, the Expect/Mask buffer is initialized by the microprocessor. Each element of the buffer has two bytes: The first byte is the expected frame data or summary information (discussed below), and the second is a mask of the bits to be compared. As a frame is received, bytes are read from the buffer and compared with the received frame bytes. When the bits compared are equal, bits in a compare register are set to zero. The contents of the compare register are written into the compare buffer at the end of the event. To check the validity of a received frame, code simply reads the compare buffer entry for the frame; if the entry is zero, the frame has passed the Expect/Mask test. Nonzero bits in the entry direct the code to the header bytes or summary byte that were not equal to the Expect/Mask values. The Expect/Mask facilities are also used to recognize data frames, as discussed in the next section.

Since the code cannot always service the inbound interface immediately, a pair of FIFO buffers, the header and compare buffers, store information from the last eight events detected on the inbound interface. Each event occupies 16 bytes of the header buffer and one byte of the compare buffer. The most common event is the receipt of a frame. In this case, the frame header (up to 15 bytes) and a summary byte are stored in the header buffer. The result of the Expect/Mask compare operation is written into the compare buffer. The write pointer is an address register for the header and compare buffers and points to the current event being stored in these buffers. After each event, the write pointer is incremented by the IFSM.

The read pointer, another address register for the header and compare buffers, points to the current event being read by the microprocessor. The read pointer is controlled by the microprocessor. When the read and write pointers are equal, the header and compare buffers are logically empty and there is no work pending for the code. When an event is detected by the IFSM, the header and compare buffers are loaded, and the write pointer is incremented as described above. Now the read and write pointers are no longer equal, which causes a microprocessor trap. After the code is finished processing the event, it increments the read pointer. If no other events were received during processing, the read and write pointers are again equal, indicating that there is no pending work for the microprocessor. The header and compare buffers also provide trace information for problem determination, since data for the last eight events are always in these buffers.

Sometimes events occur on the inbound interface faster than they can be processed by the code. This is usually caused by multiple errors on the inbound interface or by error recovery operations occurring within the ESCON channel. In either case, the header and compare buffers may become full, and the IFSM must discard subsequent events. When events have been discarded, the code is notified. It must then assume that the connection state of the ESCON Director is unknown and perform the appropriate recovery action.

Each event entered into the header buffer includes a byte of summary information that is written into the 16th byte of storage for the event, as described above. One bit of this byte indicates that the event is a normal frame. In this case, the other seven bits describe the frame. The bits indicate the type of start-of-frame and end-of-frame delimiters and the length of the frame or its header (up to 15 bytes). The IFSM normally writes the summary information and increments the write pointer after the end of the frame has been received.

When the ESCON channel is receiving data, the IFSM makes the data frame header information available to the code before the end of the frame is received. This

improves performance by allowing the code to begin processing the header as soon as possible. However, it presents a problem when an error is detected in the frame after the header information has been presented to the microprocessor. If this kind of error is detected, the IFSM creates another event and, by setting one of the summary bits, indicates that it is associated with the previous event. When the code is informed of this condition, it assumes that the previous header information is suspect and performs the appropriate recovery.

When the event summary byte indicates an abnormal event, the compare buffer entry does not hold the result of the compare operation, but contains instead a description of the abnormal event. A 4-bit field of the compare buffer entry indicates such conditions as a loss of character synchronism, a short frame, a long frame, a CRC error, or a data-buffer overrun. Since more than one of these conditions can occur within a single event, only the most important one is presented by this 4-bit field. Two more bits of the compare buffer entry indicate the validity of the delimiters. These validity indicators are useful in determining the state of the ESCON Director in error situations. Another of the bits indicates that an invalid character was received. In this case, the four bits of the summary byte that are normally used to indicate the frame length give attributes of the invalid character. These include code violations, undefined control codes, and invalid character sequences.

The compare invalid register keeps track of the validity of Expect/Mask compare operations. Whenever the code updates the Expect/Mask buffer, all entries in the compare buffer become invalid, because the code cannot determine whether a compare operation was performed before or after the Expect/Mask buffer was updated. The compare invalid register has eight bits, one associated with each compare buffer entry. Whenever the Expect/Mask buffer is changed, all eight bits of the compare invalid register are set to 1 (indicating that all entries are invalid). When the code reads the compare buffer and when the corresponding bit in the compare invalid register is on, the code receives a compare buffer value of all 1s, indicating that none of the frame header bytes were equal to the contents of the Expect/Mask buffer. The code can read the compare buffer when it contains a description of an abnormal event. After an update of the Expect/Mask buffer, as new events are received, the bits of the compare invalid register are reset by the IFSM. A 0 value of a compare invalid register bit allows the corresponding compare buffer entry to be read without modification.

• *Data frame recognition*

When the ESCON channel performs a read operation, a mixture of data and control frames is received. One way of extracting the read data would be to use a very fast processor to examine the frames at link speeds. Because this is a very costly solution, the ESCON channel uses the Expect/Mask hardware to automatically distinguish control frames from data frames and extract the read data from the frames as they are being received. This hardware must be programmable enough to handle protocols for both the native ESCON channel and the ESCON Converter; the frame header contents and length are not the same in these two protocols.

The first group of header bytes in data frames used in both the native ESCON channel and the ESCON Converter identifies the frames as data frames. This group contains X bytes. The second group contains flags that describe the status of the data transfer operation. The total length of the data frame header is Y.

The Expect/Mask hardware is used to test the frame headers. The code initializes the Expect/Mask buffer so that the headers of the anticipated incoming data frames equal the contents of the buffer. The code also sets two 4-bit registers to X and Y. The first X bytes of the incoming frame must be equal to the first X elements of the Expect/Mask buffer for the hardware to recognize the frame as a data frame. Once the frame is recognized as a data frame, the hardware uses Y to determine the end of the header and the beginning of the data field. Recall from the previous section that the summary information stored in the 16th byte of a header buffer entry is compared with the 16th element in the Expect/Mask buffer. The summary byte must also be equal to the Expect/Mask buffer element for the frame to be recognized as a data frame.

Code involvement in data transfer is further reduced by conditionally interrupting the microprocessor. As described above, the second group of bytes in the data frame header contains flags describing the status of data transfer. These flags indicate the ability of the sender to receive another data request (RDY) and the end of data transfer. Code must be interrupted if a data frame with one of these flags is received. The hardware compares the group of bytes containing the flags (from byte $X + 1$ to byte Y) with the contents of the Expect/Mask buffer. If these bytes are not equal (i.e., one of these unusual flags is received), the write pointer for the header buffer and compare buffer is incremented and a microprocessor trap is generated.

Once the data frame recognition hardware is initialized by the code, the majority of data frames are received automatically, without causing the microprocessor to be interrupted. Since the code does not have to process each data frame, these frames can be received with the minimum number of idle characters (four) between them. This means that the ESCON channel never requires additional data pacing (discussed in the section on frame-transmission hardware), regardless of how small the data frames may be.

• *Data transfer*

The data transfer process can be thought of as two distinct operations that occur simultaneously. *Storage data transfer* is the movement of data between system storage and the 2048-byte data buffer of the channel. *Interface data transfer* is the movement of data between this data buffer and the ESCON interface.

Prior to the advent of ESCON, all System/360™ and System/370 channels used the standard parallel interface [10]. Data transfer on this interface is relatively simple: The control unit provides or requests each byte of data for read or write operations respectively by raising the appropriate control line to the channel. The channel provides or accepts the data byte and raises the corresponding control line to the control unit. This process is well known and has remained relatively unchanged over the years. Since performance is important and flexibility is not required, all channels implement the data transfer on the parallel interface using hardware state machines. The function of the channel code during data transfer is limited to the movement of data between system storage and the data buffer. There is little or no involvement of the code in the transfer of the data between the data buffer and the parallel interface.

In the ESCON channel, however, data are transferred in frames that contain, in addition to the data bytes, control information in the header. This control information is different for each function implemented with the channel hardware (native ESCON channel, ESCON Converter channel, ESCON control unit, etc.) and even varies from frame to frame for a given function. In addition, data request frames flow in the opposite direction from data transfer and must be generated or processed concurrently with the data frames. For example, the native ESCON channel must be capable of receiving a data request frame at any time during a write operation. It must process this data request and then set the RDY bit in the next outbound data frame.

To provide this kind of flexibility, it was necessary that the code generate and interpret the information in the data frame headers and data request frames in addition to handling the movement of the data to and from system storage. The challenge was to create a structure that would permit this while also allowing a continuous flow of data frames to be maintained on the link. The key was to exploit the preemptive trap mechanism and the Expect/Mask and data frame recognition facilities previously described, and to provide additional hardware assistance to enable the code to perform frame analysis and setup in the minimum number of cycles [18].

To this end, two preemptive trap routines were provided for the management of the data transfer on the serial interface: the *outbound-data service trap* and the *inbound-data service trap*. The inbound-data service trap is taken whenever the inbound header buffer is not empty during data transfer. It processes data request frames (writes) and the data frame headers that do not agree with the expected values (reads). The outbound-data service trap controls the transmission of data frames (writes) and data request frames (reads). It occurs as described in the following sections.

Write operations Before the start of data transfer for a write operation, the code sets up a "template" of the expected data request frame in the Expect/Mask arrays. Therefore, when the inbound-data service trap is taken, the code has merely to check the compare byte (stored in the computer buffer) generated by the hardware to verify that the data request frame was valid. The only remaining check is the inspection of the received request count to ensure that it conforms to the requirements of the architecture being implemented. The code then loads this count into a register "stack" in the data transfer control logic. This facility consists of a counter that is decremented for each byte of data sent on the serial interface and a backup register in which request counts received while data are being transmitted on the interface can be added. If the counter is decremented to zero during the transmission of a frame, the transmission of data bytes is terminated and the CRC is appended to the frame. The backup register values are transferred to the counter by hardware each time a new data frame is started when the counter equals zero.

To enable a continuous stream of data frames to be maintained on the interface, all processing necessary to set up the next data frame must be overlapped with the transmission of the current data frame on the link. For this reason, the outbound-data service trap is taken for writes when the transmission of the header portion of each frame has been completed. It is further required that the number of cycles necessary for this processing must be less than that required to transmit the data portion of the frame. This allows only about 16 cycles for the smallest permitted data frame.

To meet this latter requirement, it is necessary for the code to identify quickly which frame it is setting up relative to the data request that is being satisfied by that frame. Two examples of the need for this information are as follows: A RDY bit may be required in the first frame satisfying a data request, but not in any of the other frames. An end (E) bit is required to be set only in the last frame of the last data request of a channel command word [1].

Several hardware comparators in the data transfer controls are provided to accomplish the overlapped processing. The first two compare the value in the request counter at the end of the header transmission with the maximum data frame size specified by the control unit, and with twice this value. If the value of the request counter at

this time is greater than twice the frame size, the frame being set up is a continuation of the current request. If the value of the request counter is less than or equal to twice the frame size, but greater than the frame size, when the transmission of the current frame is complete, the request count will have been decremented to be less than or equal to a frame size. Thus, the frame header being set up during this occurrence of the outbound-data service trap will be the last one for the current request. If the value in the request counter is less than or equal to the frame size at the end of the header transmission, the current frame transmission will cause the request count to be decremented to zero. Thus, the frame being set up by the outbound-data service trap in this case is the first one for the next request. The comparators allow the code to determine which of the above conditions holds, with a single four-way branch.

A special case exists for the second condition when the new request is the last one for the channel command word. In this case, in order to properly set the E bit, it must be determined whether the last request will be satisfied by one frame only. A third hardware comparison of the value in the backup register and the maximum frame size is provided for this purpose.

The code sets the appropriate bits in the outbound header buffer and sets a latch. Once the latch has been set, the hardware transmits the new frame when the current frame has been transmitted and the pacing and data availability requirements for the new frame have been met. When the header for the new frame has been transmitted, the outbound-data service trap is taken again and the process is repeated.

Because of the extensive use of the hardware comparators to assist the code, the outbound-data service routine takes only seven hardware cycles to modify a bit (for example, the E bit or RDY bit) in the outbound header buffer and set the latch to send the frame. The hardware cycle time is always less than the 50-ns interface byte time, so the next frame is normally completely set up prior to the transmission of the seventh byte of the current frame. Thus, the design can readily support the transmission of "back-to-back" data frames (i.e., only four idles between the EOF of one to the SOF of the next)—even at the minimum frame size of 16 bytes. For the native ESCON channel using 16-byte frames and data requests of 256 bytes, the interface data transfer routines use just 30% of the available microprocessor bandwidth, leaving the rest for fetching data from storage, IOP communications, running timers, etc. For a frame size of 256 bytes (necessary for an 18MB/s data rate), this load drops to 15% of capacity.

Read operations The data-streaming feature was added to the parallel interface in the late 1970s [10], allowing the control unit to "stream" data to the channel without waiting for a response. This feature permits higher data rates, independent of the cable length. However, if the channel is temporarily unable to accept data bytes at the rate requested by the control unit, it must abort the data transfer and signal the control unit that an overrun has occurred. Thus, with the data-streaming feature, overruns could be encountered simply because the data-streaming protocol eliminated the interlocks previously used to prevent overruns, and fully buffered control units that ran with no overruns in dc-interlock mode [10] had to cope with overruns for the first time.

Early in the development of the ESCON channel, it was decided to implement the interface protocol such that it would not be exposed to overruns. This means that there must always be space in the data buffer to accommodate all data bytes requested but not yet received. Thus, the maximum data request that can be made for each new read CCW is the data buffer size. If the CCW count exceeds this value, subsequent requests must be made as data are received and moved out of the buffer to storage. The delay between the time at which the last byte of a data block is moved out of the buffer and the time at which a data request to "backfill" this block can be transmitted directly affects the cable length at which the maximum data rate can be supported. An important objective of the design was to minimize this delay, in order to achieve the maximum possible "full-bandwidth" cable length. Of course, complete code control of the processing of data and data request frames was still required for maximum flexibility.

It was thus decided to exploit once again the rapid task switching of the preemptive trap mechanism by using the outbound-data service trap to allow the code to make data requests. A counter is provided to keep track of the number of bytes moved between the data buffer and the staging hardware. When this counter reaches a threshold value selected by the code, the outbound data service trap is taken. This threshold can be any value from 16 to 1024 in powers of two. When the trap is taken, a data request frame for the number of bytes given by the threshold has previously been set up in the outbound header buffer. The code has only to check whether certain conditions for sending a data request have been met before initiating the transmission of the frame. This can be accomplished within about nine hardware cycles after the space becomes available in the buffer. After sending the frame, the code computes and builds the next data request in the outbound header buffer.

The data frame recognition hardware described in the section on data frame recognition automatically controls the proper routing of headers and data to the correct buffers and checks for proper format all headers that do not contain control information. Only headers that contain

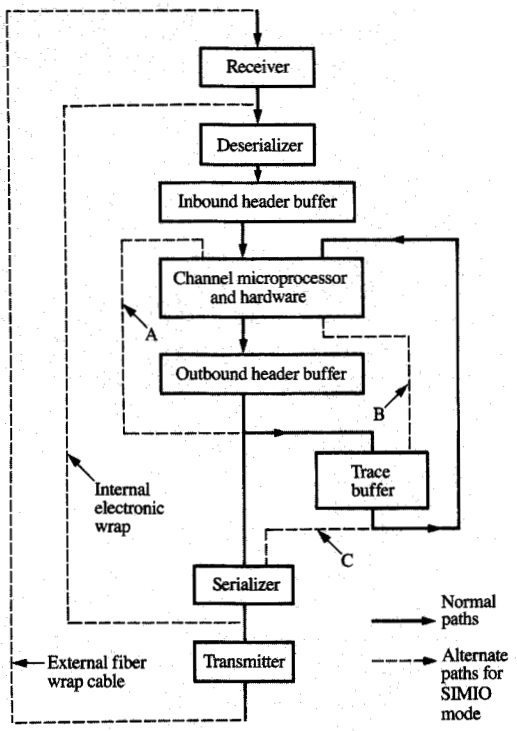

Figure 7

SIMIO control unit data flow.

control information cause the inbound-data service trap. The code in this routine uses the hardware-generated compare byte from the compare buffer to quickly identify the control bits that were set. The code then takes the appropriate action on the basis of these bits.

For a data request size of 256 bytes, the outbound- and inbound-data service routines use only about 15% of the available bandwidth of the microprocessor, leaving the remainder for moving data to storage, running timers, etc. This number is not dependent on the data frame size as it was for writes, since additional data frames that do not contain control information are handled entirely by hardware.

- *I/O simulation*

Simulated I/O (SIMIO) is the ability of the channel to behave as if it were physically attached to a control unit. Code and hardware in each channel provide the simulated control unit function. This allows a complete system test of the processor and channels without the attachment of a real control unit to each channel. Because of the complexities of cabling and configuring a large number of control units, this capability becomes more important as the number of channels provided on a system grows.

In parallel channels, the simulated control unit is implemented by adding hardware at the I/O interface. This hardware generates signals to the channel in response to signals from the channel. When the simulated control unit sends data to the channel for a read operation, the data pattern is supplied by a data generator. When the channel sends data to the simulated control unit for a write operation, the same data generator is used to check the data pattern.

In the ESCON channel, however, all communication between the channel and the control unit is done in frames. A large amount of hardware would be required to analyze frames from the channel and generate response frames to the channel. Also, the channel implements multiple architectures, and thus supports multiple frame formats.

For these reasons, control unit simulation in the ESCON channel is implemented using a combination of hardware and code [19]. The hardware provides alternate data paths, a data generator function similar to that used on the parallel channels, and a preemptive trap to invoke the SIMIO code. This code is a stand-alone routine, separate from the mainline code. It uses the hardware facilities to analyze the outbound frames from the channel and to generate the required response frames to the channel.

The alternate data paths that are enabled in the SIMIO mode are shown as dashed lines in **Figure 7**. The outbound header buffer is the source of frames for the SIMIO code, so it effectively becomes the inbound header buffer of the simulated control unit (path A in Figure 7). Similarly, the SIMIO code builds its response frames in the first 16 bytes of the outbound trace buffer, so it effectively becomes the outbound header buffer of the simulated control unit (paths B and C in Figure 7). Outbound frame tracing is disabled in SIMIO mode.

When mainline channel code sends a frame, transmission of the frame on the link is suppressed, and the SIMIO preemptive trap is taken instead. The code examines the frame in the outbound header buffer, builds the appropriate response in the trace buffer, and sets a latch. This causes the out-frame state machine (OFSM) to serialize and transmit the frame from the trace buffer exactly as it does from the outbound header buffer in normal mode.

In order for the frame transmitted by the simulated control unit to be received by the channel, the outbound link must be wrapped back to the inbound link. This may be accomplished either by an internal electronic wrap, which connects the output of the serializer to the input of the deserializer, or by an external fiber optic wrap cable. The latter method has the advantage of checking the fiber

optic components. Both of these paths are also shown as dashed lines in Figure 7.

In the SIMIO mode, data for the data frames are generated and examined by means of the data generator and the trace buffer. The data generator is capable of generating both an incrementing and a pseudorandom data pattern, and the last 64 bytes of the trace buffer can be loaded by the microprocessor with any data pattern. When the SIMIO code examines data from a data frame, it instructs the OFSM to compare data from the data buffer with data from either the data generator or the trace buffer. When the SIMIO code generates response data, it can specify the source as either the data generator or the trace buffer.

• *Technology and package*

Figure 8 is a photograph of a Model 9021 ESCON channel card with an optical duplex jumper cable attached. At the lower right is an uncapped channel logic module, exposing the three logic and three array chips. All of the ES/9000 machines use similar components and card layouts. The card shown contains two physically separate channels. The card technology is "pin in hole" on 2.54-mm (100-mil) centers. There are three power and four signal planes; the card connector, on the right-hand side of the card, contains 144 pins. Without the optical connectors, the card is about 120 mm wide and 180 mm tall. The optical connectors are attached to the left-hand side of the card and add about 40 mm to its width.

The two gold-colored rectangular boxes next to each of the optical connectors are the transmitter and receiver. These components are connected to the serializer/deserializer modules, which are 28 mm square. Each serializer/deserializer contains two bipolar chips and several passive components. The transmitter, receiver, and serializer/deserializer operate at 200 MHz, thus requiring careful card layout. The card also contains many small components, such as crystals, decoupling capacitors, terminating resistors, and inductors.

The logic modules, measuring 50 mm on a side, are the largest components. Their multilayer ceramic (MLC) substrates have 21 layers and a thickness of almost 4 mm. The substrates can hold up to four logic chips (9.4 mm square) and six array chips (9.4 mm by 6.5 mm). Decoupling capacitors are provided on the substrates.

Each logic chip is fabricated using the IBM CMOS2 process [20, 21]. The chips contain random logic and a number of embedded arrays. The ability to define arrays within the logic chips was vital in the design of the ESCON channel. The array chips are CMOS static arrays, containing 144 kilobits each, with an access time of 15 ns.

Conclusions

Because of its high degree of programmability and content-independent frame-transfer facilities, the ESCON channel has proven to be a versatile frame processor suitable for many different functions. The single hardware design provides two channel types for 18 different processor models in the ES/9000 computer line. The design is equally adaptable to control units: Three have been completed and others are planned. The channel attachment bus provides a minimum-wire attachment mechanism for integration of the channel in a system.

A significant saving in hardware development cost resulted from having a common design for all of these functions. In addition, further savings in development, debugging, and testing were realized by the ability to share code routines among the functions.

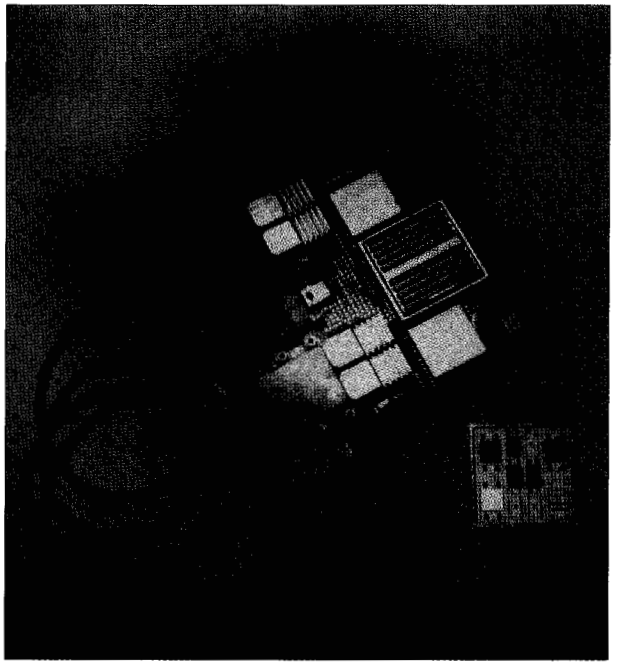

Figure 8

Model 9021 ESCON channel card and MLC module.

Acknowledgments

Design of the ESCON channel would not have been possible without the combined efforts of the entire channel development team in Poughkeepsie. We especially acknowledge the contributions of Cathy Huang, Matt Kalos, and Bjorn Liencres. We also thank Dave Meltzer and Lisa Spainhower for their thorough review of this manuscript.

References

1. J. C. Elliott and M. W. Sachs, "The IBM Enterprise Systems Connection (ESCON) Architecture," *IBM J. Res. Develop.* **36**, 577–591 (1992, this issue).
2. *IBM Enterprise Systems Architecture/390 Principles of Operation*, Order No. SA22-7201; available through IBM branch offices.
3. I. R. Radziejewski, E. Lo, R. H. S. Hardy, and A. M. Leung, "Improved Performance Token Ring Network Interface Adapter," *Proc. IEE, Part E* **137**, 421–426 (1990).
4. T. Yaguchi, K. Fujimoto, E. Katsumata, K. Tanaka, K. Tamaru, A. Kanuma, Y. Katagiri, A. Nishikawa, H. Shiraishi, T. Yamamoto, K. Kimura, Y. Terui, and T. Hamai, "Design of a CMOS Token Ring LAN Controller, TRC, Compatible with IEEE802.2 MAC Protocol," *Proceedings of the 1989 Symposium on VLSI Circuits*, pp. 129–130.
5. D. R. Scherbarth, "Designing an FDDI Adapter," *Wescon '90 Conf. Record* **34**, 118–120 (1990).
6. S. G. Tucker, "The IBM 3090 System: An Overview," *IBM Syst. J.* **25**, 15–16 (1986).
7. C. J. Georgiou, T. A. Larsen, P. W. Oakhill, and B. Salimi, "The IBM Enterprise Systems Connection (ESCON) Director: A Dynamic Switch for 200Mb/s Fiber Optic Links," *IBM J. Res. Develop.* **36**, 593–616 (1992, this issue).
8. A. X. Widmer and P. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," *IBM J. Res. Develop.* **27**, 440–451 (1983).
9. T. A. Gregg and L. Skarshinski, "Transmitting Commands Over a Serial Link," U.S. Patent 5,048,061, September 10, 1991.
10. *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*, Order No. GA22-6974; available through IBM branch offices.
11. D. P. Seraphim and I. Feinberg, "Electronic Packaging Evolution in IBM," *IBM J. Res. Develop.* **25**, 617–629 (1981).
12. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the Design Automation Conference*, IEEE Computer Society, New Orleans, 1977, pp. 462–467.
13. G. Brent, T. Gregg, P. Oakhill, and M. S. Siegel, "Asynchronous Multi-Clock Bidirectional Buffer Controller," *IBM Tech. Disclosure Bull.* **24**, 4404–4406 (1982).
14. T. A. Gregg, "Data Synchronizing Buffers for Data Processing Channels," U.S. Patent 5,003,558, March 26, 1991.
15. D. F. Casper, J. R. Flanagan, T. A. Gregg, C. C. Huang, and M. J. Kalos, "Apparatus for Decoding Frames from a Data Link," U.S. Patent 5,025,458, June 18, 1991.
16. L. Skarshinski, "Character Synchronization Method," *IBM Tech. Disclosure Bull.* **28**, 5577–5579 (1986).
17. *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, Order No. SA22-7202; available through IBM branch offices.
18. D. F. Casper, J. R. Flanagan, T. A. Gregg, C. C. Huang, and M. J. Kalos, "System for High Speed Transfer of Data Frames Between a Channel and an Input/Output Device with Request and Backup Request Count Registers," U.S. Patent 5,101,477, March 31, 1992.
19. D. F. Casper, J. R. Flanagan, T. A. Gregg, and M. J. Kalos, "Control Unit Simulation on the Serial Channel," *IBM Tech. Disclosure Bull.* **33**, 170–172 (1991).
20. A. W. Aldridge, R. F. Keil, J. H. Panner, G. D. Pittman, and D. R. Thomas, "A 40K Equivalent Gate CMOS Standard Cell Chip," *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1987, pp. 248–251.
21. Robert Hornung, Martine Bonneau, Bernard Waymel, James Fiore, Elliot Gould, Ronald Piro, John Martin, Lance McAllister, and Sze Tom, "A Versatile VSLI Design System for Combining Gate Array and Standard Cell Circuits on the Same Chip," *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1987, pp. 245–247.

Received June 19, 1991; accepted for publication May 13, 1992

John R. Flanagan *IBM Enterprise Systems, P.O. Box 950, Poughkeepsie, New York 12602 (FLANAGAN at PK705VMA, flanagan@pk705vma.vnet.ibm.com).* Mr. Flanagan is a Senior Engineer in the Interconnect Products group. He received a B.S. degree in electrical engineering from Santa Clara University in 1973 and an M.S. degree in electrical engineering from Stanford University in 1974. He joined IBM at the Poughkeepsie Laboratory in 1974 and has held various technical positions in the areas of circuit design, cryptographic unit design, and channel design. Mr. Flanagan holds several patents relating to the ESCON channel design and has received an IBM Invention Achievement Award. He received an IBM Outstanding Innovation Award for his work on the development of the ESCON channel and ESCON architecture. Mr. Flanagan is a member of Tau Beta Pi and Eta Kappa Nu.

Thomas A. Gregg *IBM Enterprise Systems, P.O. Box 950, Poughkeepsie, New York 12602 (GREGG at PKEDVM9, greggtag@vnet.ibm.com).* Mr. Gregg is a Senior Engineer in the Interconnect Products group. He received an Sc.B. degree in engineering from Brown University in 1972 and continued there under a University Fellowship, receiving an Sc.M. degree in electrical engineering in 1974. He joined IBM at the Poughkeepsie Laboratory in 1973. Mr. Gregg has held various technical positions in the area of I/O subsystem design. He holds patents utilized in various IBM serial channel products and has received three IBM Invention Achievement Awards. He received an IBM Outstanding Innovation Award for work in the architecture, design, and implementation of ESCON products.

Daniel F. Casper *IBM Enterprise Systems, P.O. Box 950, Poughkeepsie, New York 12602 (CASPER at PKSMRVM).* Mr. Casper is a Senior Technical Staff Member in the Interconnect Products group. He received a B.S. degree in electrical engineering from the University of Wisconsin at Madison in 1970, joining IBM at the Kingston Laboratory that same year. Mr. Casper has held various technical positions in the areas of channel and system-control-element development. He holds numerous patents relating to channel design and has received two IBM Invention Achievement Awards. He has received several other formal awards, including an IBM Outstanding Technical Achievement Award for his work on the IBM 3090 system channels and an IBM Technical Excellence Award. Mr. Casper is a member of the Institute of Electrical and Electronics Engineers.