# HOTLink™-Based Data-Mover

## Introduction

This application note details the design and implementation of a generic data-mover based on the Cypress HOTLink™ high-speed serial interface transmitter/receiver ICs. The design is implemented using clocked FIFOs for data storage, a pASIC FPGA (field programmable gate array) as the system and serial-link controller, and HOTLink for the serial transmission and reception of the data. The pASIC controller design is implemented in a hierarchical structure of VHDL files to allow simple reuse or modification of the various modules to meet user-specific implementation requirements.

**What is a Data-Mover?**

A data-mover is that portion of a design or piece of equipment whose sole purpose is to move significant quantities of data from one physical location to another. Data-movers are quite common in most digital equipment. Common computer interfaces like printer ports, serial ports, and modems are all forms of data movers. Other parts of systems that also fall under the data-mover umbrella are backplanes, buses, sensor connections, etc.

The design described in this application note is applicable to those types of data-movers that can benefit from the speed, distance, power, and cost advantages of serial data transmission.

## Data-Mover Design

The first part of any design effort is to define the requirements of the design. For this design the requirements are

- Simultaneous bidirectional transmission of data
- Guaranteed reliable data transfer
- Hardware-based link-error recovery
- Parity-protected data paths
- Self-contained link diagnostics including
  - — Local and remote data loopback
  - — Local and remote BIST
- FIFO-based data storage

In addition to these fundamental requirements, some decisions must be made as to the functionality of the parallel or host-side interface of the design. A byte-wide interface was selected for this design to simplify the buffer resource requirements. This host interface can easily be changed to multi-byte widths to match up with specific host bus requirements.

This design was implemented using Cypress CY9266 HOTLink evaluation boards for the media interface. These cards provide a documented and proven serial media interface, and allow a variety of optical and copper media to be used for the serial connection. The complete design and functionality of these cards are documented in the Cypress "CY9266 HOTLink Evaluation Board User's Guide."

A complete parts list and interconnect schematic for the data buffer and control portions of this data-mover may be found in *Appendices A* through *G* of this application note.

**Top-Level Block Diagram**

This data-mover operates similar to a digital modem and is capable of moving continuous streams of data bidirectionally between two points. The design is symmetrical in nature, with both ends of the data-mover built with identical hardware. A top-level block diagram is shown in *Figure 1*. Because of the symmetrical or peer-to-peer type of design, *Figure 1* only shows one half of a full system.

The Control-PLD is the center or hub of the data-mover. It manages the transmit and receive FIFOs, error management, and diagnostic functions. All data passes through the Control-PLD, with link protocol and error check characters added and removed automatically.

This Control-PLD is almost entirely synchronous with the transmit path clock. This allows the internal state machines for both the transmit and receive data paths to interact without large numbers of metastable prevention, speed matching, or handshake circuits. The internal design hierarchy of VHDL modules used to create this Control-PLD is shown in *Appendix A*, while the external pin connections of the part are shown in *Appendix C*.

*FIFO Buffers*

FIFO memories are used for all data buffers in the design. This removes the need for large numbers of address counters and address-pin connections. By their very nature, FIFOs are also dual ported. This allows simultaneous read and write access to the data buffers by both the host bus and the Control-PLD.

All FIFOs used in this data-mover are of the clocked or synchronous variety. This selection was made to maximize the timing margins in the design. It allows the data-mover to operate at full speed (33 MBytes/second/direction) without the constraint of requiring special read and write pulse shapes. It also allows the design to operate at its maximum speed using the slowest available clocked FIFOs.

A total of seven FIFOs are present in each end of the system. These are:

- Receive Input FIFO—This FIFO is used as a speed matching buffer between the HOTLink receiver and the Control-PLD. It uses the HOTLink receiver $\overline{\text{RDY}}$ signal to filter out excess fill characters and prevent a FIFO overflow. This FIFO is referenced as U14 in the schematic in *Appendix C*.
- Receive Packet FIFO-A—This FIFO is used to receive a packet (with parity) and hold the packet until it has been validated. Following validation, the packet is transferred to the bulk receive FIFO. This FIFO is referenced as U12 in the schematic in *Appendix D*.

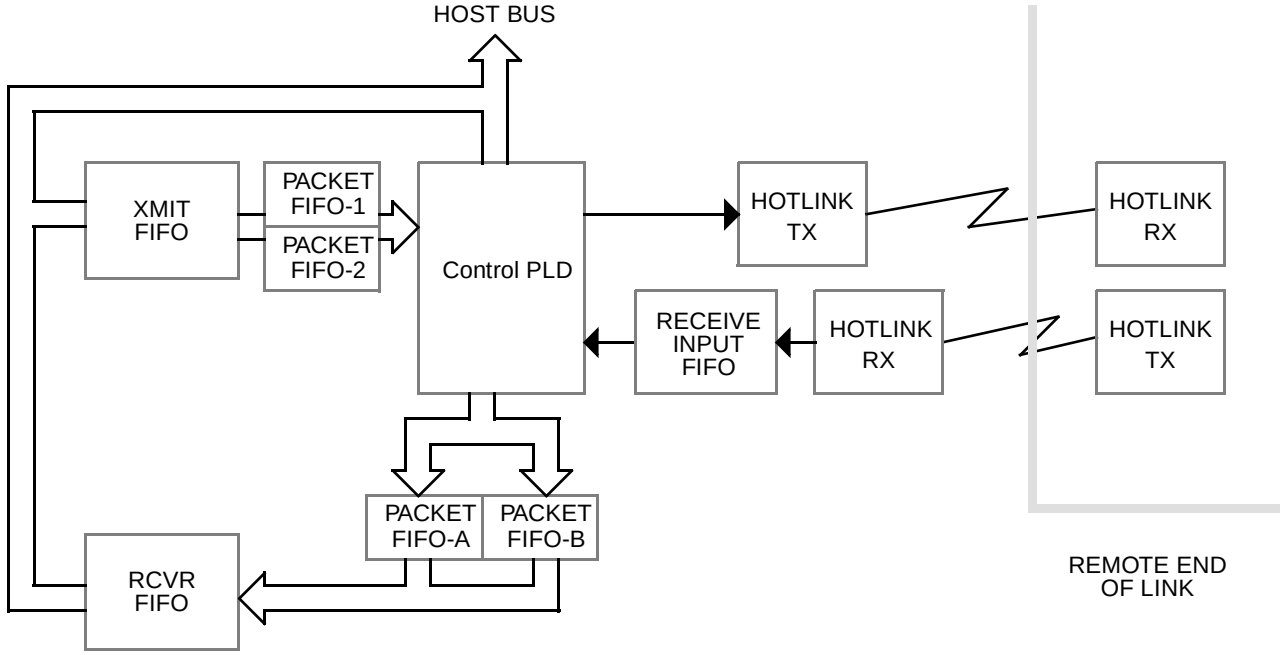



**Figure 1. Data Mover Top-Level Block Diagram**

- Receive Packet FIFO-B—This FIFO is identical in operation to the receive packet FIFO-A. Dual packet FIFOs are used to allow overlapped operation with the receive packet FIFO-A. This FIFO is referenced as U11 in the schematic in *Appendix D*.
- Bulk Receive FIFO—This FIFO accumulates the validated data for transfer to the host bus. It is capable of containing multiple packets to allow sustained operation at the maximum rate of the serial link. This FIFO is referenced as U10 in the schematic in *Appendix D.*
- Bulk Transmit FIFO—This FIFO operates similar to the bulk receive FIFO. It allows the host bus to load large amounts of data for transmission to the remote end of the link. This FIFO is referenced as U9 in the schematic in *Appendix D*.
- Transmit Packet FIFO-1—This FIFO accumulates a packet for transmission. Once sufficient data is loaded into the packet FIFO, the data is routed through the Control-PLD for transmission on the serial link. When transmitting, this FIFO is operated in a recirculate mode to allow immediate retransmission of the packet if errors are detected by the remote receiver. This FIFO is referenced as U5 in the schematic in *Appendix D*.
- Transmit Packet FIFO-2—This FIFO is identical in operation to the transmit packet FIFO-1. It allows overlapped load and transmit operations with transmit packet FIFO-1. This FIFO is referenced as U6 in the schematic in *Appendix D*.

Cypress CY7C451 FIFOs were selected for the packet FIFOs in this design. Their 512 x 9 size offers a reasonable maximum packet size. Other synchronous or clocked FIFOs may also be used in the design, often with no modification to the Control-PLD logic. Some possible FIFOs are listed in *Table 1*.

Not all FIFOs listed in *Table 1* will work directly in this design without modification, but all support the synchronous data and flag interface required by the Control-PLD. Specific capabilities (three-state outputs, programmable flags, retransmit capability, etc.) are present in some, but not all, of these components. Selection of appropriate FIFOs should be based on the specific implementation and performance requirements of the user's design.

**Table 1. Cypress Synchronous/Clocked FIFOs**

| Word Depth | 9-Bit Width | 18-Bit Width |
|---|---|---|
| 64 | CY7C4421 | CY7C4425 |
| 256 | CY7C4201 | CY7C4205 |
| 512 | CY7C4211, CY7C441, CY7C451 | CY7C4215, CY7C455 |
| 1K | CY7C4221 | CY7C4225, CY7C456 |
| 2K | CY7C4231, CY7C443, CY7C453 | CY7C4235, CY7C457 |
| 4K | CY7C4241 | CY7C4245 |
| 8K | CY7C4251 | CY7C4255 |
| 16K | CY7C4261 | CY7C4265 |
| 32K | CY7C4271 | |

*Host Bus Interface*

The Control-PLD requires a host processor or other external control to direct it into various diagnostic modes, and to act as a source and sink for data transferred across the link. In this design, this interface is presented as a single byte-wide port. The transmit and receive data-path bulk FIFOs can be kept on separate buses, or all placed on a common bus.

The interface to the transmit data path is through a CY7C453 clocked FIFO. This FIFO can be loaded at speeds of up to 70

MHz (70 MBytes/second). This is substantially faster than the rate at which data can be transmitted through the serial link (33 MBytes/second). If the host system is capable of transferring data at this fast rate, it will have sufficient bandwidth available to service the receive bulk FIFO through a shared port with the transmit FIFO.

The host bus interface to these transmit and receive FIFOs is implemented as a single byte-wide interface. Alternate word widths (16-bit, 32-bit, etc.) are also possible using additional FIFOs (or FIFOs with wider data paths), and should be selected based on the specific host bus configuration. Selection of alternate bus widths will affect other portions of the design that presently use a byte-wide data path.

This design assumes the transmit FIFO is loaded with data having odd parity. This parity is checked as the data is read from the packet FIFOs. If odd parity is not present on the data, the Control-PLD will still transmit the data, but a transmission error will be forced at the HOTLink transmitter by asserting SVS.

A small change to the Control-PLD can remove this parity requirement; however, such a change will not free up any significant resources within the Control-PLD. A large majority of the XOR structures used to implement the parity tree are shared with the CRC generator.

The receive path appends odd parity to each received character prior to loading that data into the receive packet FIFOs. This parity information is carried through the bulk receive FIFO, and may be checked or disregarded by the host bus interface hardware.

*HOTLink Interface*

This data mover design was configured around the capabilities present in the CY9266 HOTLink Evaluation Boards. These boards provide a standardized parallel interface to the HOTLink transmitter and receiver, as well as access to various physical media interfaces for fiber-optic, coax, and shielded twisted pair media. Additional information on these CY9266 boards is available in the “CY9266 HOTLink Evaluation Board User’s Guide.” The physical connection to these cards is through a 60-pin board edge connector. The schematic of the interconnect between the Control-PLD and the CY9266 cards is shown in *Appendix C*.

For use with this data-mover design, the HOTLink transmitter is configured to use its internal 8B/10B encoder, and is enabled to accept a character on every clock cycle ($\overline{ENA}$ is tied LOW with $\overline{ENN}$ tied HIGH). This allows the transmitter to start generation of the BIST sequence by assertion of only the $\overline{BISTEN}$ signal.

With $\overline{ENA}$ hardwired active, all Sync/Fill characters (C5.0) must be generated by the Control-PLD. Generation of this character is controlled by the transmit data path mux in the Control-PLD, such that the default state (when no data is available) is to output a C5.0 SYNC code.

The HOTLink receiver is also configured to use its internal 8B/10B decoder. To simplify framing operations, the transmission protocol is set up to ensure a minimum of two C5.0 characters between packets. This allows the receiver RF (reframe) control line to be hardwired HIGH, enabling the receiver’s multi-byte framer.

The HOTLink receiver data bus is routed through an arbitrarily sized receive input clocked FIFO. This FIFO is used to change the data timing reference from a recovered receive data clock (CKR) to the same clock used for transmitting data (CKW). The CKR clock is used for the write clock for this FIFO, with the read clock being the CKW clock.

The packet protocol is constructed such that not all characters received are written into this Receive Input FIFO. Since a read occurs on every CKW clock cycle, and the read and write clock rates are nearly identical, the FIFO can never overflow. In fact, this FIFO remains at (or within one or two bytes of) empty at all times. For those skilled in the art of reliable FIFO design, it could be possible to include this speed matching function within the Control-PLD.

The receiver CKR clock is still used in the Control-PLD, but only for those state machines monitoring BIST progress in the receiver. These state machines operate using signals ($\overline{RDY}$ and RVS) that are synchronous to the CKR clock, with their output information routed through metastable prevention circuits internal to the Control-PLD.

**Control Codes**

The serial interface is both controlled and maintained by the selective use of control codes. These control codes are characters that can be sent and received across the serial interface the same way that data characters are transmitted and received.

The 8B/10B code built into HOTLink allows the use of twelve different control codes, identified as C0.0 through C11.0 (for an explanation of the character naming conventions please see the CY7B923/CY7B933 data sheet or the Cypress *HOTLink User’s Guide*). Because these control codes exist separate from the codes used to represent the normal 256 data characters, they may be freely intermixed with the data stream and easily separated at the receive end of the link.

This capability to intermix control codes with data characters allows these control codes to be used as part of a low-level protocol to control and maintain the interface. For this data-mover, each of the twelve available control codes has been mapped to a specific function or meaning. Those functions selected are listed in *Table 2.*

**Table 2. Control Codes**

| Code | Name | Function |
|---|---|---|
| C0.0 | EOP | End Of Packet |
| C1.0 | ACK | Acknowledge Packet |
| C2.0 | NAK | Not Acknowledge Packet |
| C3.0 | XON | Enable Transmission |
| C4.0 | XOFF | Disable Transmission |
| C5.0 | SYNC | Synchronize/Fill |
| C6.0 | RLBON | Enable Remote Loopback |
| C7.0 | RLBOFF | Disable Remote Loopback |
| C8.0 | RBISTON | Enable Remote BIST |
| C9.0 | RBISTOFF | Disable Remote BIST |
| C10.0 | RRESET | Reset Remote Controller |
| C11.0 | RCOMP | Reset Complete |

The assignment of a specific control code to a specific function is for the most part arbitrary. The one exception to this is the C5.0 code. This control code is used for two functions: character-synchronization of the HOTLink receiver, and as a filler character to be sent when no other information is pending. In any type of HOTLink-based serial link this control code must always be used for this function, because the synchronization operation is performed in the HOTLink receiver, not in the Control-PLD.

For applications that require more than twelve functions to develop or support a protocol, it is possible to create additional function/code-mappings by grouping a control code with one or more additional control codes or data characters. This specific sequence of characters is then used to determine the desired function.

This is similar to the use of escape sequences to control printers, whereby a single control code (usually a hexadecimal 1B) is combined with one or more characters that do not print, but direct the printer to enter other modes of operation.

**Packet Generation**

This data-mover allows operation using either a continuous or variable-rate data stream. As data is placed into the bulk transmit FIFO it is automatically segmented into packets for transmission. This automatic packetization is used to minimize latency on transfers of short or slow data-rate information, while maximizing throughput on faster data-rate transfers.

The two transmit packet FIFOs exist in one of two orthogonal states: loading and transferring. While one packet FIFO is enabled to load data from the bulk FIFO, the other is enabled to transfer its data as a packet across the serial link. Following acknowledgment of the transmitted packet, if any data is present in the loading FIFO, the functions of the two packet FIFOs are swapped. Now the FIFO that had been loading (and now contains data) is configured to transfer data, while the other packet FIFO (now empty) is configured to load data.

This three-FIFO structure (one bulk and two packet) allows packet data content to be limited to no less than one byte, and no more than the depth of the packet FIFO. The selection of the CY7C451 for the packet FIFOs in this design limits the maximum packet size to 512 bytes. The data-mover will also work with larger or smaller FIFOs in either the packet or bulk FIFO locations, with slight variations in system performance due to link and system latencies.

*Packet Format*

The default packet structure used by the data-mover is shown in the upper half of *Figure 2*. The start of a packet is set by the first data character (non-control code) transmitted on the interface. All remaining data bytes in the transmitting packet FIFO are sent as contiguous characters until the entire packet has been sent. The end of the packet is set by the transmission of the EOP (C0.0, End Of Packet) control code. The next two transmitted data characters are the Cyclic Redundancy Code (CRC) check characters for the packet.

This format also allows other command characters to be intermixed within the transmitted data stream, as shown in the lower half of *Figure 2*. These non-data characters are not considered part of the packet itself, and can include various combinations of packet responses or fill characters as directed by the receive path.

By allowing this mixing of data and control characters, the ACK/NAK responses for received packets can be transmitted as soon as they are available, rather than delaying the response until the end of the current transmit packet. This minimizes the delay between packets on the interface and allows for faster overall data transfers.

**Transfer Protocol**

Any individual packet transfer consists of moving data from a transmit node to a receive node. These data transfers are handled through a relatively simple protocol. The transfer of a single packet is diagrammed in *Figure 3.*

The transmit node remains in a waiting state until data is available in the loading packet FIFO. Once data is available, the functions of the two transmit packet FIFOs are swapped, making what was the loading FIFO into the transmitting FIFO. Next the number of bytes written into the FIFO is loaded into a packet-length counter.

The actual data transfer is started by reading data bytes from the transmit packet FIFO and presenting them to the HOTLink transmitter and CRC generator. This read and transmit activ-

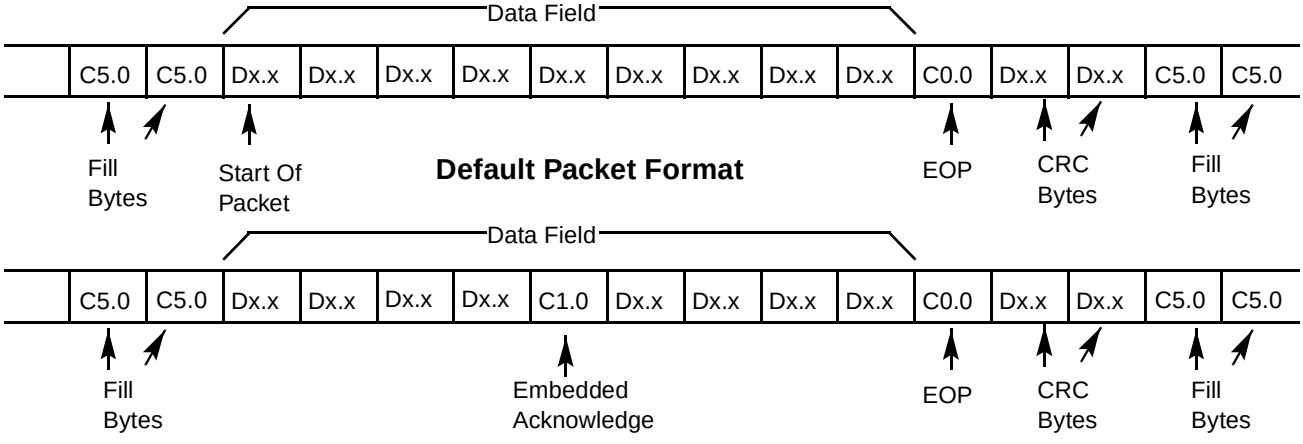



**Figure 2. Data-Mover Packet Format**

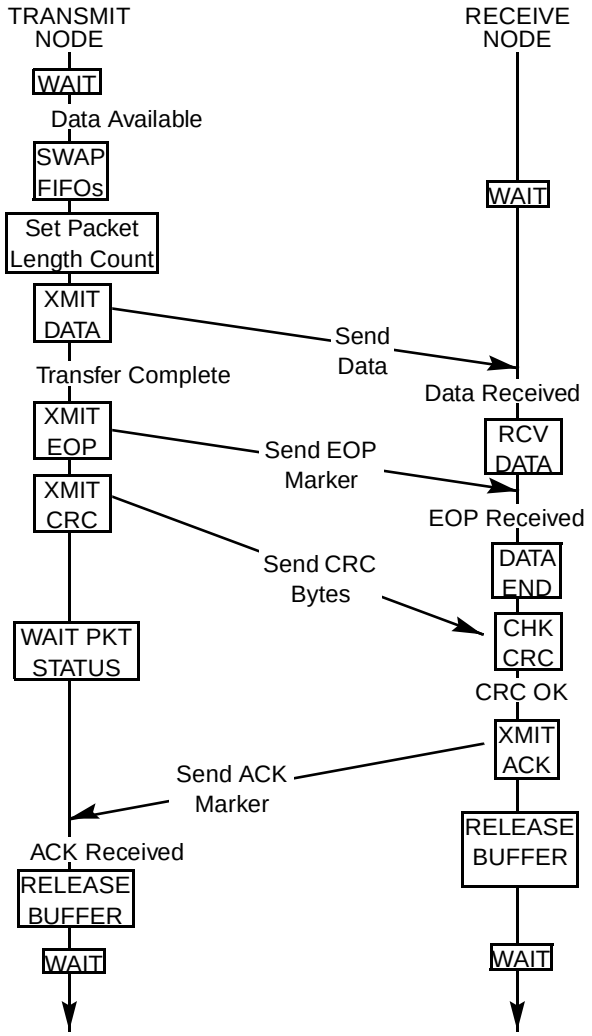



**Figure 3. Default Packet Transmission**

ity continues until the packet length counter in the Control-PLD indicates that all data has been read from the FIFO.

When the transfer counter indicates that the entire data field for the packet has been sent, the transmit node marks the end of the data field by sending an EOP character to the receive node, immediately followed by two bytes of CRC information. Following the end of the CRC information the transmit node waits for an acknowledge for the packet from the receive node.

The receive node is normally in a wait state where it constantly monitors the received character stream for data characters. When the first data character is received, it is clocked into a CRC checker and written into a receive packet FIFO. As subsequent data characters are received they are also clocked into the CRC checker and packet FIFO.

When the EOP character is detected in the data stream, the receive node knows that the next two data characters are not part of the data field, but are the CRC bytes for the packet. As these CRC bytes are received they are clocked into the CRC checker, but are not loaded into the receive packet FIFO.

After the second CRC byte has been received, the receive node checks for a valid CRC. If the CRC is valid, the receive node stuffs an acknowledge (ACK) character into its outbound data stream and releases the packet FIFO, allowing it to empty into the bulk receive data FIFO.

When the ACK character is received by the transmit node, it knows that the packet was delivered without error to the receive node. It then releases the packet FIFO by performing a master reset on it, and then returns to the wait state to see if any more data is ready to be transferred.

Operation in this manner assumes that the transfer went perfectly, and that the CRC register results at the receive node indicated an error-free transfer. For those cases where the received packet was detected in error, the information flow is changed to follow that in *Figure 4*.

Here everything operates with the same flow shown in *Figure 3* until the CRC is checked at the receiver. When the CRC indicates an error in transmission or reception, the receive node stuffs a not-acknowledge (NAK) character into its outbound data stream, and erases the packet buffer.

When the NAK is received by the transmit node, it reloads the packet length counter, and starts transmission of the same packet again. Since the data was written back into the FIFO

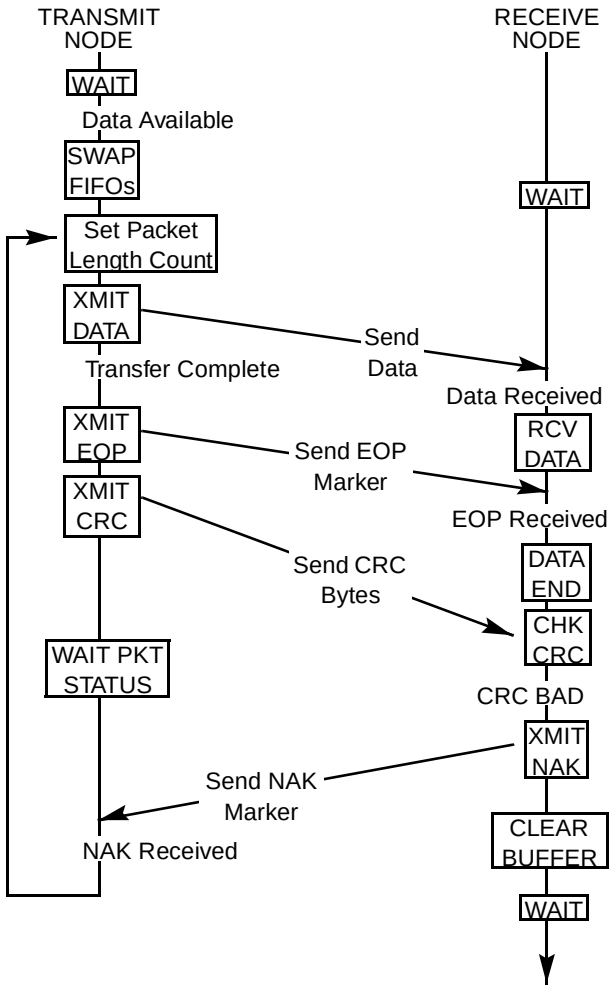



**Figure 4. Packet Transmission With CRC Error**

when it was originally transmitted, the word at the output of the FIFO is the first word of the packet to be retransmitted. Data transmission occurs, as shown in *Figure 3*, until the CRC is checked. If the CRC is now valid, an ACK is generated and the flow follows that in *Figure 3.* If it is still bad, another NAK is generated and the flow follows that shown in *Figure 4*.

*XON/XOFF*

These data transport flow diagrams are followed for standard data transmission. However, these flows may be modified by reception of an XOFF control code. The XOFF control code is sent by the receive node to instruct the transmit node to stop data transmission. This control code is sent by the receive node if there is no available receive packet buffer to accept a new packet from the transmit node. (This XOFF control code can also be sent under control of the host system by setting bit 2 in the transmit control register.)

The determination of when the receive node should send an XOFF control code is made following validation of the packet CRC. If the CRC indicates the packet is bad, the packet handling is the same as that shown in *Figure 4*. However, if the CRC indicates the packet is good but the present packet buffer cannot yet be allowed to empty, the flow in *Figure 5* is followed.

The XOFF is sent to the remote node to inform it that the receive node, while functional, cannot at this time accept any more data. This event is generally caused by one of two events: either the host bus has allowed the bulk receive FIFO to fill such that the alternate (draining) packet FIFO has no place to put its data, or the packet just received was shorter than the previous packet, and the alternate receive packet FIFO has not yet had sufficient time to completely empty.

When the XOFF is received by the transmit node, it sets a bit in its status register that can be monitored by the host system and by the internal state machines in the transmit node. This status allows the host to use different activity timers to check for possibly link hung conditions. In the case of an actual link hang it also allows faults to be isolated between a link failure (no XOFF status present), and a remote host failure (constant XOFF status).

The receive node sends an ACK for the validated packet immediately after transmission of the XOFF control code, and then waits for the alternate (draining) packet FIFO to become empty. When this FIFO is empty, the receive node generates an XON control code, releases the validated packet buffer, and prepares to receive the next packet. The transmit node, upon reception of the XON control code, clears the XOFF status bit and prepares to send any pending data.

## Performance

The definition of performance for a data-mover can take on many different meanings. These definitions are all tied to various forms of “how much,” “how fast,” and “how efficient.” This data-mover is optimized to provide low-latency or “time to first data” for the link. Because of this, it may not be the most efficient or fastest link implementation for all types of data traffic.

The goal in most data-movers is to get the data from the source to the destination as fast as possible and to allow the data to be used as soon as possible. Unfortunately, the link setting to maximize one of these tends to minimize the other. To achieve the highest speeds requires that maximum length packets are sent. To allow data to be used as soon as possible requires sending short packets; both to limit the packet accumulation delay at the source, and to limit the time to validation at the end of the packet. This data-mover attempts to satisfy both objectives by offering a compromise solution capable of moving (and validating) small amounts of data with very short latency, and increasing the packet size as the amount of data to be sent increases.

**Link Start-up**

When data is first presented to the transmit bulk FIFO, it immediately (within a couple clock cycles) is presented to the transmit packet FIFOs. If only a single byte is present, that byte is transferred to one of the transmit packet FIFOs. Once

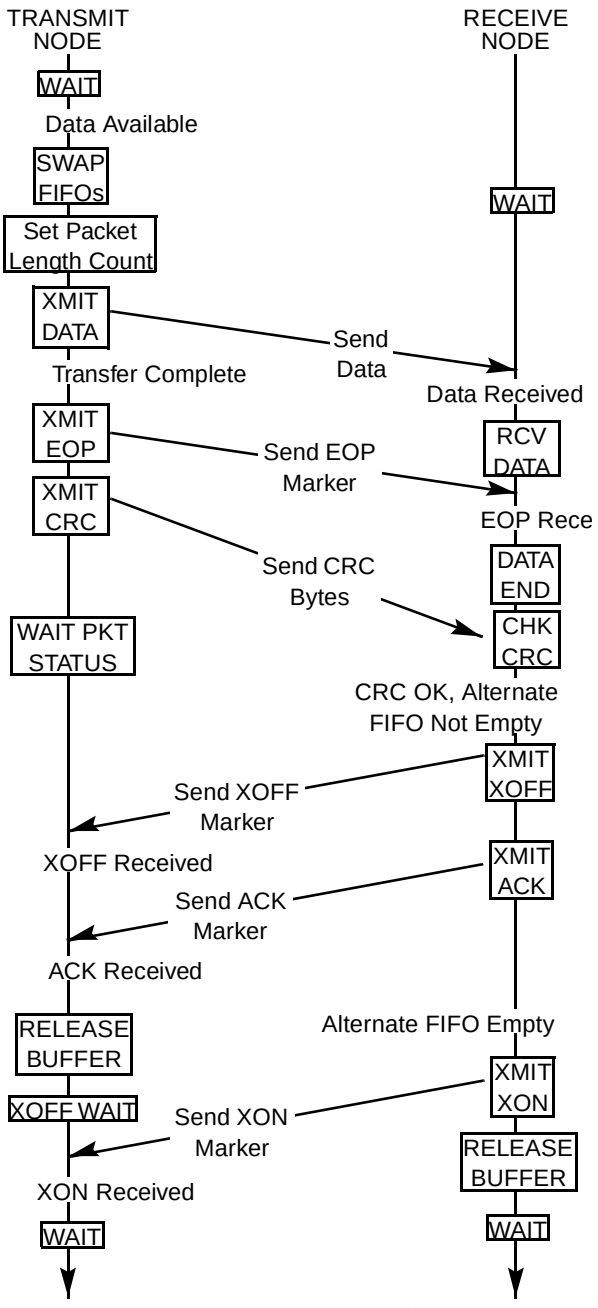



**Figure 5. Packet Transmission With XOFF**

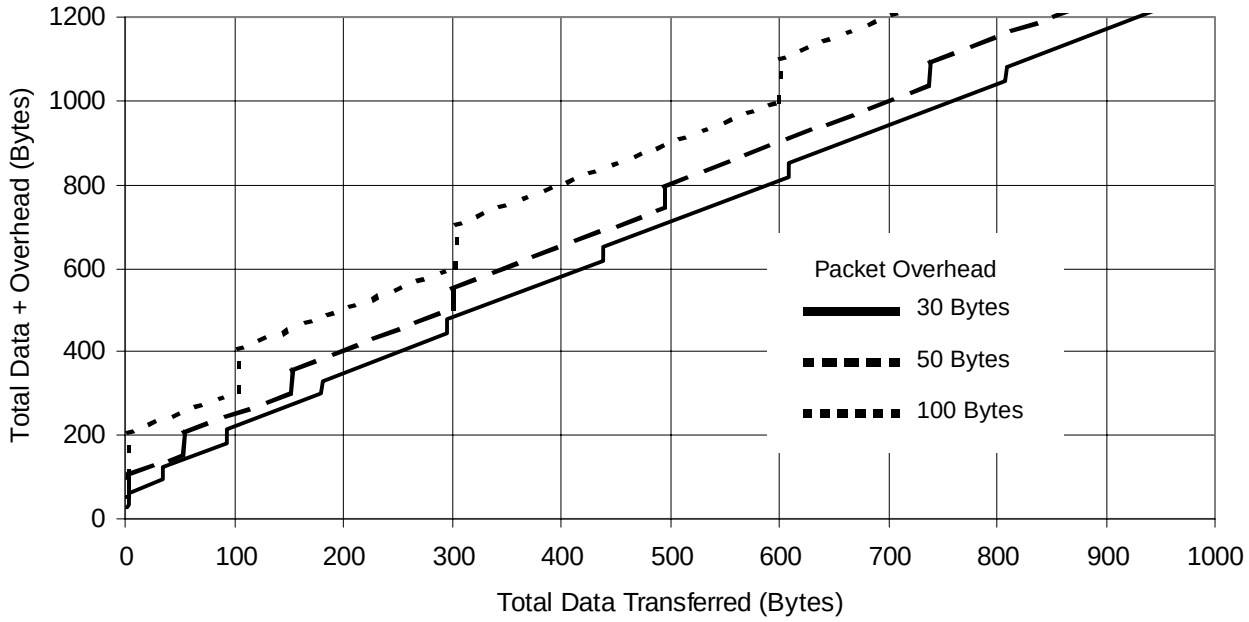



**Figure 6. Data Transfer Profile**

there is any data present in this loading transmit packet FIFO (and none in the other), the loading function is disabled and the data is transmitted across the serial link.

This transmission of even a single byte involves numerous state machines, FIFOs, and pipeline registers, in both the source and destination ends of the link. It also requires numerous overhead characters to mark the end of the packet (EOP), the CRC bytes, and other characters used to maintain the link. Because the link requires an acknowledge for each packet, the round-trip transmission time on the media is also important. These delays and extra characters combine to form a fixed per-packet overhead. For this design the total per-packet round-trip overhead is approximately 30 characters.

This overhead means that from the time a byte is loaded for transmission, until it is available as validated data at the receive FIFO outputs is on the order of 20 characters, and that the transmit end cannot start transmission of the next packet for at least 31 characters (round-trip overhead + data length).

This is not the delay per byte in the system. If the initial data load was two or three bytes, then the total link delays and latency would only increase by the delay of the added characters. A two byte transfer loop time would take 32 characters, with a three byte transfer taking 33 characters.

Beyond three bytes, things change significantly. Since the first packet is launched as soon as possible (to get validated data to the receiver as soon as possible), any data beyond the first three bytes is queued up for loading into the next packet FIFO. This second packet FIFO is enabled to load data from the bulk transmit FIFO while the first packet of three data bytes is being sent. It remains enabled for loading until a packet-valid acknowledge (ACK) is received. This allows up to 33 bytes (30 bytes during link overhead and three bytes during data transfer) to be loaded into the second packet FIFO.

When this second packet is sent, it must be accompanied by a second packet acknowledge and its associated delay. If the amount of data to be transferred exceeds the amount that can be loaded into the second packet buffer in the available time, additional packets are generated. Each packet becomes successively larger than the previous packet until either all the available data has been sent, or the maximum packet size is reached. A plot of data bytes transferred versus total characters to complete the transfer is shown in *Figure 6*.

In *Figure 6*, three different packet overhead amounts are shown. The solid line, at 30 characters, represents the overhead of a minimum length link implementation. With this small value, almost all overhead characters are caused by required protocol characters, pipeline delays, and state machine decision delays. The monotonic increases in total character times that occur at semi-regular intervals are generated when the amount of data to be transferred requires an additional packet, and shows the incremental overhead that occurs with that packet. Note also that the amount of data transferred between each of these packet boundaries increases with each successive packet.

As the media length increases, so too does the packet overhead. When a media delay of 20 characters (10 each direction) is added to the link, the total amount of overhead per packet increases to 50 characters. A media delay of 70 (35 each direction) would have a total overhead of 100 characters. An examination of these curves in *Figure 6* shows that the additional link delay slows the overall transmission of data; i.e., more overhead characters are incurred to transfer the same amount of data. However, this additional delay is spread across fewer but longer packets to limit the impact to total delay and link efficiency.

**Maximum Length Packets**

The data transfer state machines start the first packet of any transfer when only three bytes of data are loaded into the first packet FIFO. For systems sending very low-rate traffic this may be the best solution. Systems that operate with a known

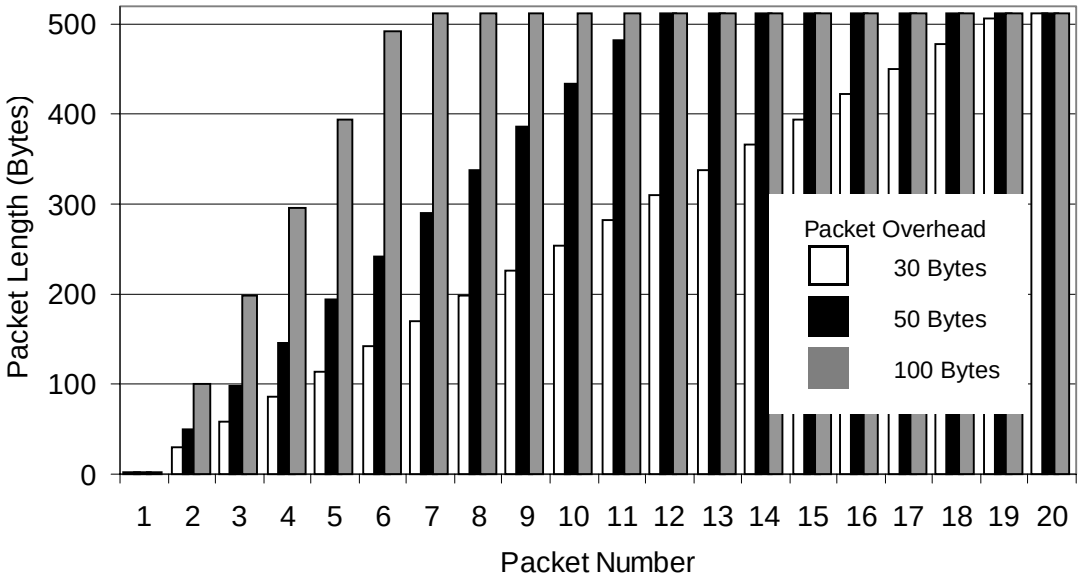



**Figure 7. Data-Mover Packet Startup Profile**

minimum transfer size may wish to modify the loading mechanism to create a larger first packet.

Following the first packet, if there is still data to be transferred, each packet will be larger in size than the previous packet. The rate that this packet size grows is determined primarily by the length of the previous packet, and by the link delay. A plot of packet length vs. packet number for three different link delays is shown in *Figure 7*. Note that the longer the link delay, the fewer the number of packets that must be sent before the maximum packet length (determined by the size of the packet FIFO) is reached. This has a direct relationship to the efficiency of the link.

**Link Efficiency**

Link efficiency is a measure of how much usable data is moved across a link relative to the total number of characters transferred. Links such as this, with a fixed packet overhead but variable packet size, operate at different efficiencies at different times.

Two types of efficiency profiles actually exist. The first is the link efficiency to transfer a specific quantity of data, starting from a no-data-present situation. This is plotted in *Figure 8*. The link efficiency quickly rises as the transfer size grows, reaching to within 10% of the theoretical maximum within the first 2000 characters.

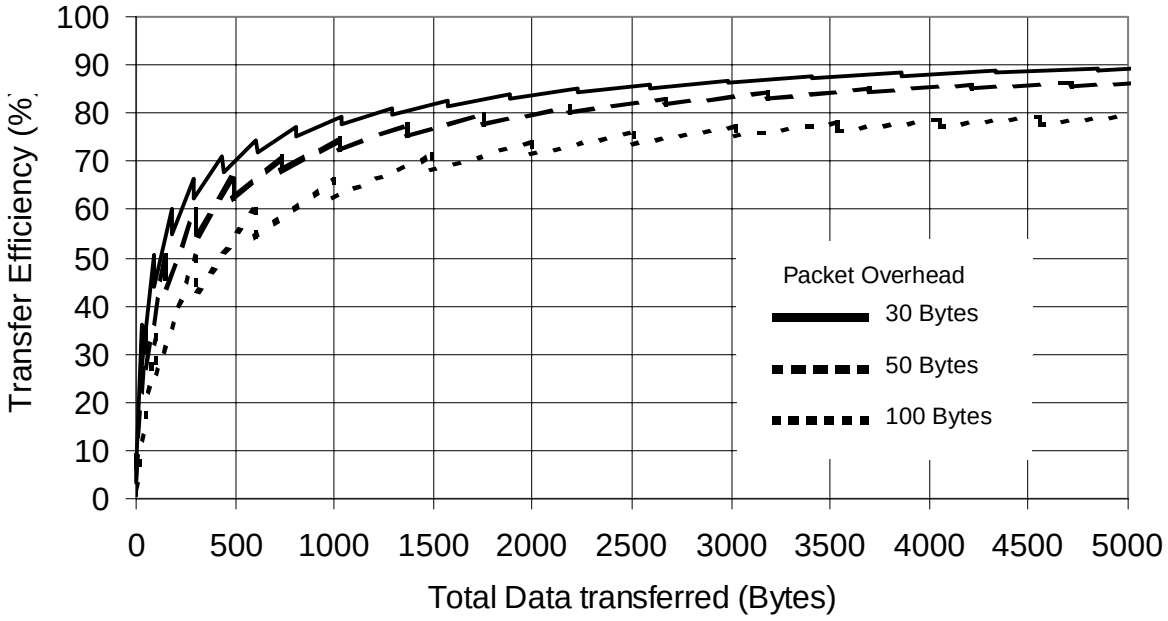



**Figure 8. Link Efficiency**

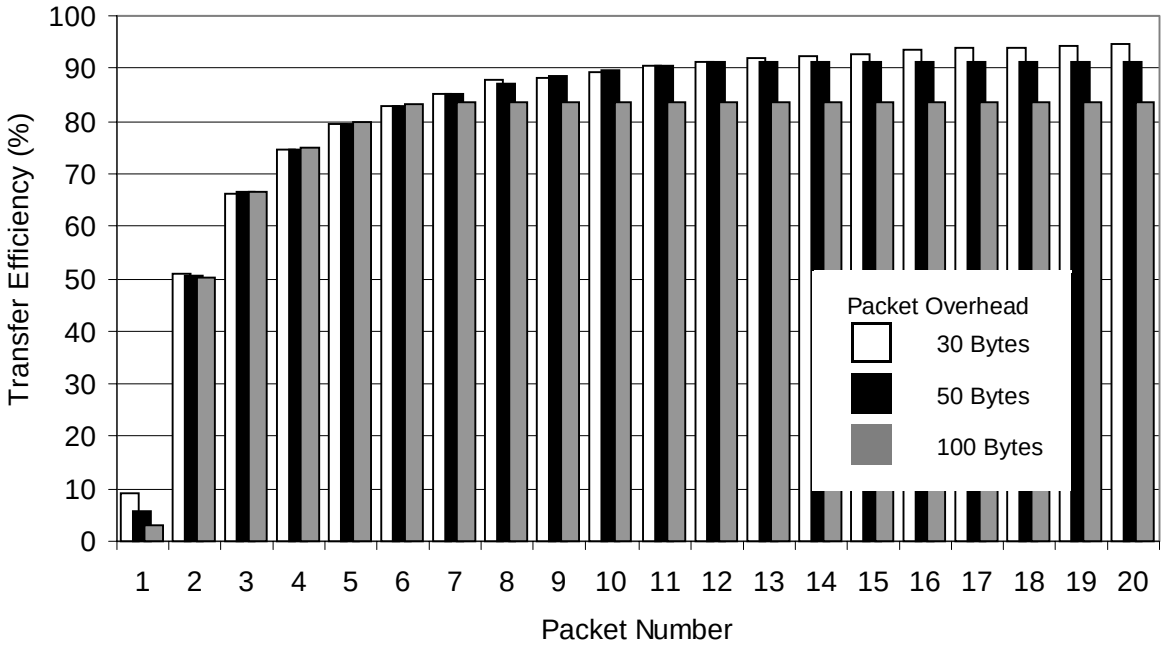



**Figure 9. Data-Mover Packet Efficiency**

The second efficiency profile, actually a derivative of the first, is the instantaneous efficiency of any individual packet in a transfer. This relationship is shown in *Figure 9*. On a per-packet basis, the efficiency remains nearly identical, regardless of the link overhead. However, once the maximum packet length is achieved on a specific link, the efficiency locks in at a constant percentage.

## Control-PLD Design

A block diagram of the Control-PLD is shown in *Figure 10*. The Control-PLD contains three primary functional blocks: control path, transmit data path, and receive data path.

The control path is used to both control and monitor the operation of the transmit and receive data paths. It contains two 8-bit registers (one read-only, one write-only) that are accessed through a bidirectional 8-bit bus. It also contains numerous state machines for diagnostic functions.

The transmit path controls the reading and writing of the transmit FIFOs to send packets of data across the serial link. Framing, CRC, and other protocol overhead characters are merged with the FIFO data stream as it is sent to the HOTLink transmitter for serialization. If a packet is received at the remote end incorrectly, the transmit path automatically resends the packet.

The receive path accepts and interprets a stream of characters from the HOTLink receiver. The data stream is made synchronous to the transmit path clock (CKW) through a small FIFO between the HOTLink receiver and the receive data-path logic.

The receive data-path logic then removes any overhead characters added by the remote transmit path logic, places the data into one of the receive packet FIFOs, and uses the CRC at the end of each packet to validate the received data. If a packet contains an error, it is discarded, and the remote transmitter is instructed to resend the packet.

### Control Path

The control path contains two registers and numerous small state machines that control

- Local/Remote BIST
- Local/Remote reset
- Local/Remote loopback

*Control Register*

The Control-PLD is placed in its different modes of operation through a single writable register, with status monitored through a single readable register. This allows the entire Control-PLD to occupy a single address in the I/O space of the host system.

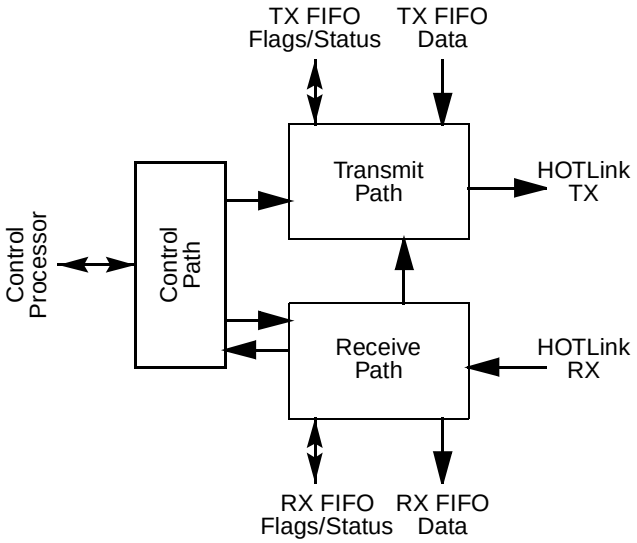



**Figure 10. Control-PLD Block Diagram**

The default mode of operation of the Control-PLD is to move data. Other modes of operation are set through the control register. The bits of this register are shown in *Table 3*.

The control register contains eight settable bits. When the register is written (by an I/O write operation) the bits are all set to match the contents of the I/O bus at the end of the write operation. The individual bits are

- Bit 7—Enable Local BIST. When set, this bit places both the local HOTLink transmitter and receiver into BIST mode, and selects the alternate (B) port on the HOTLink receiver. This bit also drives FOTO to disable external transmission of the serial BIST sequence.
- Bit 6—Enable Remote BIST. When set, this bit places the local receiver into BIST mode and notifies the remote transmitter to enter BIST mode.
- Bit 5—Enable Local Loopback. When set, this bit selects the alternate (B) port on the local receiver. This bit also drives FOTO to disable external transmission of serial data.
- Bit 4—Enable Remote Loopback. When set, the local transmitter informs the remote Control-PLD to route all received command and data characters to its transmitter, and not to its receive packet buffers.
- Bit 3—Send XOFF. When set, causes the local transmitter to send an XOFF command character to the remote receiver. The remote transmitter should then stop sending data immediately. Due to recognition and transport delays (>20 character clocks), the response to this will not be immediate.
- Bit 2—Remote Reset. When set, causes the local transmitter to send a sequence of RRESET command characters to the remote receiver. Upon reception, the remote receiver initiates a local master-reset sequence and enters normal data mode.
- Bit 1—FOTO. When set, this bit activates the FOTO signal to the local transmitter. This disables the external transmission of serial data.
- Bit 0—Master Reset. When set, this bit causes an internal reset of the Control-PLD, identical in operation to driving the external RESET signal LOW. This bit is self clearing as part of the Master reset process in the Control-PLD. Writing a zero to this bit clears the Reset Complete bit (bit 0) in the status register.

*Status Register*

The status register is used to present status of any active diagnostic function, and the current operating state of the transmit and receive paths of the data-mover. The bits of the status register are shown in *Table 3*.

The status register contains eight readable bits. When this register is read (by an I/O read operation) the current status bits are latched and do not change while the read I/O cycle is active. To get an updated version of the status bits it is necessary to perform a separate read cycle. This is done to make sure that the status bits do not change in mid-cycle, possibly violating the set-up and hold times of the reading processor or interface. The individual status bits are

- Bit 7—BIST Status. This bit is used to track the Valid/Invalid status of the received BIST loop. If the current loop is received without error this bit is cleared (0). If an error is detected at any point during the BIST loop, this bit is set (1). This status bit changes only at the end of each BIST loop. Since the BIST loop is 511 characters in length, this bit should be polled a minimum of once per BIST loop (once every 15 μs at a 33-MHz character clock rate) to avoid missing a possible error indication.
- Bit 6—BIST Loop. This bit is used to track the passage of each BIST loop. It toggles state on each pass through the BIST loop. It can be used to validate that the status information in bit 7 is from a new pass through the BIST loop. This bit should be polled at a minimum of once per BIST loop (once every 15 μs at a 33-MHz character clock rate) to avoid missing the indication of a completed pass through the BIST loop.
- Bit 5—Wait for BIST. This bit is used to indicate the status of a BIST operation that places the local receiver into BIST mode. When HIGH, it indicates that the local receiver has not been able to locate the start of the BIST sequence in the received data stream. Once enabled and presented with a valid BIST data stream, the receiver should lock onto the transmitted BIST stream in one or two passes of the BIST loop (15.5 μs to 31 μs at a 33-MHz character rate). If it takes significantly longer than this (>100 μs), something is wrong with the link hardware and it should be repaired.
- Bit 4—Remote BIST Active. This bit indicates that the RBISTON control code (C8.0) has been received while the HOTLink receiver is operating in normal (non-BIST) mode. Reception of this bit informs the transmit path to stop sending data, and to assert the HOTLink transmitter $\overline{\text{BISTEN}}$ signal. This bit is cleared by reception of an RBISTOFF control code (C9.0).
- Bit 3—Remote Loopback Active. This bit indicates that the RLBON control code (C6.0) has been received while the HOTLink receiver is operating in normal (non-BIST) mode. Reception of this control code informs the receive path to direct all received characters to the transmit path, instead of to the received packet FIFOs. The transmit path then routes all received characters back to the HOTLink transmitter without modification. The receive path is placed into a monitor-only state looking for the RLBOFF and RRESET control codes. This status bit and functionality are cleared by reception of an RLBOFF control code (C7.0).

**Table 3. Control/Status Register Bits**

| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| **Control** | Enable Local BIST | Enable Remote BIST | Enable Local Loopback | Enable Remote Loopback | XOFF | Remote Reset | FOTO | Master Reset |
| **Status** | BIST Status | BIST Loop | Wait For BIST | Remote BIST Requested | Remote Loopback Active | Loss Of Signal | Transmit Parity Error | Reset Complete |

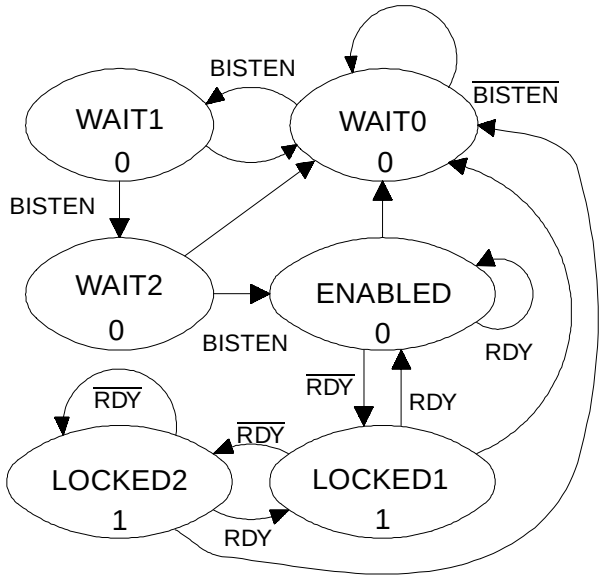



**Figure 11. Local-BIST State Machine**

- Bit 2—Loss Of Signal. This bit indicates the latched status of the carrier-detect signal of the receiver. When HIGH it indicates that the carrier detect circuit detected a loss of carrier at some time since the Status Register was last read. When LOW it indicates that the carrier has remained valid since the Status Register was last read. This status bit is cleared (LOW) when read.
- Bit 1—Parity Error Detected. This bit indicates the status of byte parity for the transmit path. When HIGH, it indicates that a character was read from one of the transmit packet FIFOs that did not contain correct (odd) parity, and that this parity error remained during a subsequent re-read of the packet FIFO contents. This is considered a fatal error, and can only be cleared by a master reset of the Control-PLD.
- Bit 0—Reset Complete. This bit is set following the completion of either a local or a remote master reset of the Control-PLD. This bit is cleared by writing a 0 (LOW) into bit 0 of the control register.

*Local-BIST State Machine*

The local-BIST state machine, shown in *Figure 11,* allows the local HOTLink transmitter and receiver to test themselves and the local serial link between them. When enabled, the local HOTLink transmitter generates its BIST sequence and sends it to the local HOTLink receiver as shown in *Figure 12*.

This state machine is enabled when bit 7 is set in the control register. This bit causes assertion of the $\overline{\text{BISTEN}}$ inputs to the local HOTLink transmitter and receiver, the FOTO pin on the HOTLink transmitter is driven (HIGH) to disable external transmission, and the A/$\overline{\text{B}}$ pin on the HOTLink receiver is driven (LOW) to select local serial data loopback.

This state machine uses input signals ($\overline{\text{RDY}}$ and RVS) that are synchronous to the receiver CKR clock, and asynchronous to most of the Control-PLD logic (clocked by CKW). To minimize the number of metastable prevention circuits (and the possibility of lost state transitions), this machine is constructed to operate from the local CKR clock.

The local-BIST state machine has a total of six states. It controls when bits 5, 6, and 7 of the status register change in response to BIST progress and any detected errors. It is controlled by two input signals: $\overline{\text{BISTEN}}$ and $\overline{\text{RDY}}$. Whenever $\overline{\text{BISTEN}}$ is not active, the machine is returned to the WAIT0 state (while all state transition arrows are shown for these transitions, not all of them are labeled).

The WAIT0 state is the default or reset state of the machine. It remains in this state when bit 7 of the control register is cleared (0). Once $\overline{\text{BISTEN}}$ is activated by bit 7, the machine goes through two secondary wait states (WAIT1 and WAIT2) before entering the ENABLED state where it looks for $\overline{\text{RDY}}$ being active. These wait states are necessary to allow the receiver time to recognize the $\overline{\text{BISTEN}}$ signal and bring $\overline{\text{RDY}}$ HIGH.

When the ENABLED state is reached, the machine remains in this state until $\overline{\text{RDY}}$ goes LOW, indicating that the local HOTLink receiver has detected the start of the BIST loop. Detection of $\overline{\text{RDY}}$ being LOW then causes the machine to move to the first of two LOCKED states. These LOCKED states signify that the HOTLink receiver is now performing matching of the received data bits to its internal pattern generator.

Bit 5 of the status register is active (1) when the local-BIST state machine is in the WAIT1, WAIT2, or ENABLED states. Once the machine moves to either of the LOCKED states, this bit is cleared (0). Should the local receiver ever detect too many errors and have to locate the start of the BIST sequence again, the machine will re-enter the ENABLED state and bit 5 will again be active.

In the two LOCKED states, bits 6 and 7 of the status register are used to track BIST errors and completed BIST loops. The reason two LOCKED states are present is to allow for the single pulse on $\overline{\text{RDY}}$ that indicates the end of a BIST loop. If $\overline{\text{RDY}}$ is ever HIGH for more than one clock, the HOTLink receiver has determined that it is no longer in sync with the transmitter and it starts looking for the start-of-loop character again.

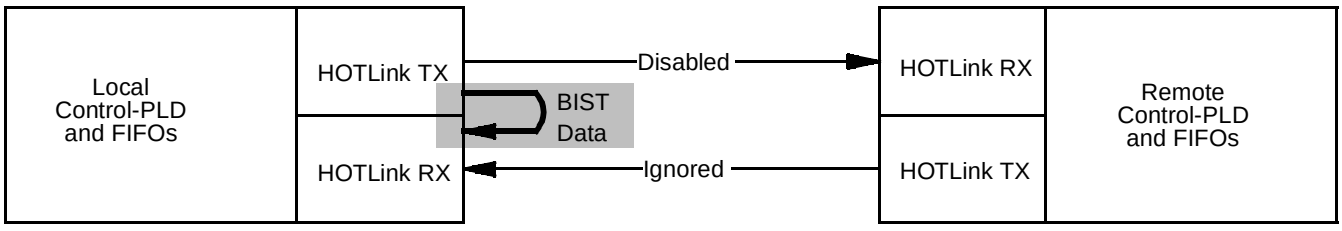



**Figure 12. Local BIST Data Flow**

Bit 7 of the status register is used to track pattern-match errors in the BIST sequence. These errors are reported by the HOTLink receiver by driving its RVS line HIGH during any character where the received character does not exactly match the expected character.

During the 511-character BIST loop, if an error is detected in any character it is latched in a flip-flop. At the end of each BIST loop the state of this flip-flop is captured in bit 7 of the status register. The state of this status bit is maintained until either the local-BIST request is cleared (bit 7 of the control register set to 0) or new error status is passed by the completion of another BIST loop.

Because this BIST error status is updated once per BIST loop, the status of a detected error will be lost if it is not captured by the host system during the following BIST loop period. For this to work correctly, the host system must be intimately involved with initiating and monitoring BIST operations.

For systems not capable of such tightly-coupled testing, the functionality of this bit should be changed to capture any reported errors, and retain that error status until the status register is polled by the host system. However, when configured in this way, it is no longer possible to determine if an error occurred within a specific pass of the BIST loop.

If BIST error-count information is necessary, a small counter could be added to the design to track each error as it occurs. However, the extremely low error rates of HOTLink-based interfaces make the benefit of such a counter questionable.

Bit 6 of the status register is similar to that of bit 7, except that it is configured to toggle state on completion of each BIST loop. By using bits 5, 6, and 7 in combination, it is possible to track both the progress of BIST in the receiver, as well as to count the number of BIST loops completed and those containing errors. This simple BIST interface should be sufficient for most applications.

*Remote BIST*

Remote BIST exists in both master and slave forms. The form is determined by where the request or directive for remote BIST operation originated. If bit 6 is set in the control register, the local interface becomes the master or controlling entity for the remote BIST operation.

In this mode, the local HOTLink receiver is again placed into BIST mode (through assertion of the local receiver's $\overline{\text{BISTEN}}$ input) while the local transmitter stops sending data and instead sends a single RBISTON control code (C8.0) to the remote receiver.

Once the remote receiver detects this C8.0 control code, it enters slave mode for remote BIST. Here it sets bit 4 in its local status register, suspends normal data reception and transmission, and asserts $\overline{\text{BISTEN}}$ to its HOTLink transmitter (the remote transmitter). This causes the remote transmitter to generate its BIST sequence.

The local HOTLink receiver, and status register bits 5, 6, and 7, then operate the same way they do when in local BIST mode. Now, however, the local transmitter is also used to send BIST status and progress information to the remote receiver and status register. This is done by sending an ACK (C1.0) control code to the remote receiver for each completed pass of the BIST loop where no errors were detected, or a NAK at the end of the loop if an error was present in one or more of the characters.

In the remote status register, bits 6 and 7 are used to track these received ACK/NAK control codes, and to present status similar to that received during local BIST operations. This BIST data and ACK/NAK response flow is shown in *Figure 13*.

The local interface returns to normal data transmission and reception when bit 6 is cleared (0) in the local control register. When this transition is detected, the local transmitter generates a single RBISTOFF control code (C9.0). The remote receiver detects this code and also returns to normal data operations.

*Local/Remote Reset*

Resets exist in two forms in the data-mover. They can be generated by the local interface (local reset) or by the remote interface (remote reset). Local resets can be initiated either by an external signal to the Control-PLD, or by setting bit 0 in the control register.

Remote resets are initiated by reception of the RRESET (C10.0) control code. To prevent an accidental reset operation from being started, the reset control code must be received for three consecutive characters to be viewed as an actual reset. Any character received that is other than a C10.0 resets the count. This is handled by the state machine shown in

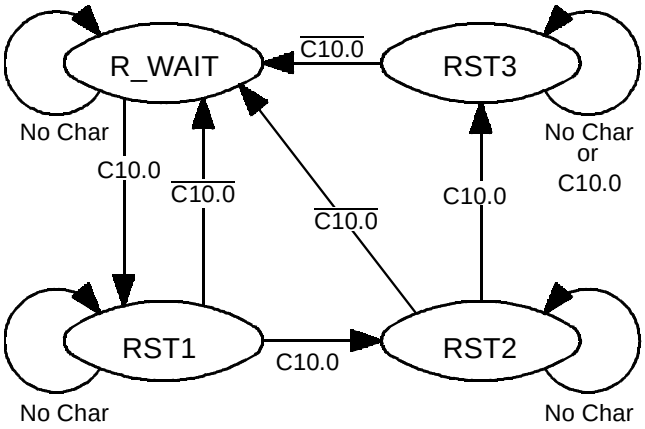



**Figure 14. Remote Reset State Machine**

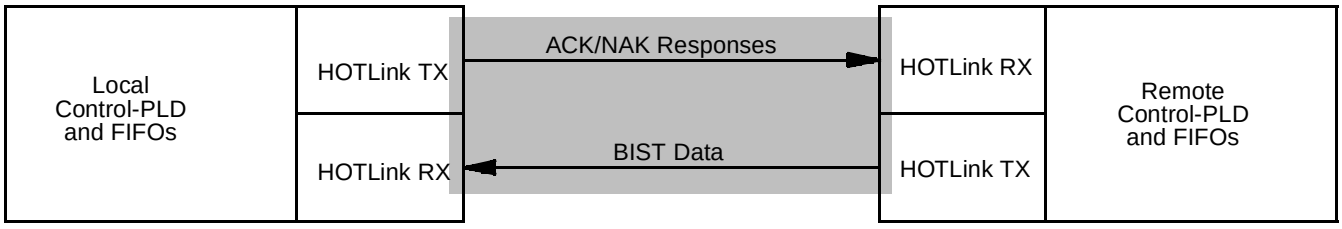



**Figure 13. Remote BIST Data Flow**

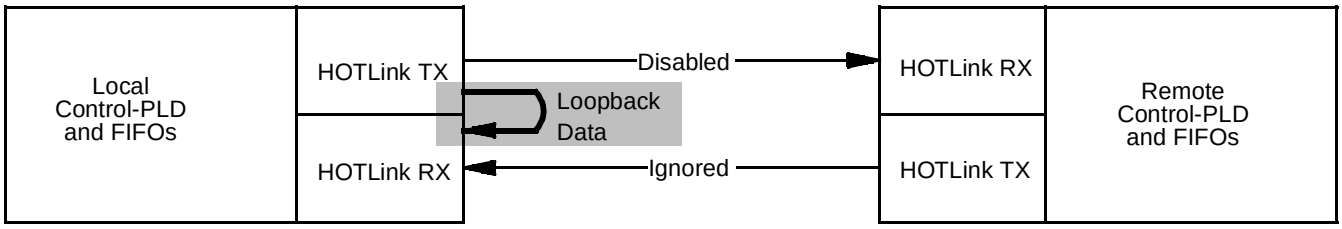



**Figure 15. Local Loopback Data Flow**

*Figure 14*. This multi-character sequence is necessary to prevent accidental resets due to characters being aliased to a false RRESET control code.

A remote reset is initiated by setting bit 2 in the local control register. This bit is polled by the transmit path when it is between packets or not transferring information. When this bit is detected, the transmit path generates a four-character sequence of RRESET (C10.0) characters. When detected by the Remote Reset state machine at the remote end of the link, a reset operation is started in the remote machine.

Both local and remote resets perform the same function, and at least one of them is necessary following power-up. The result of this reset is to initialize all of the state machines in the Control-PLD. Once the reset is removed (automatic if done through the control register) the state machines in the Control-PLD generate controlled master resets to the FIFOs used for data buffering. This is necessary to clear any unwanted or stale data from the FIFOs and to initialize their internal array pointers.

Following completion of the reset operation, a single RCOMP (C11.0) control code is sent by the transmitter to inform the other end of the link that a reset has taken place. When detected by the remote receiver, it sets bit 0 in its local status register.

*Local Loopback*

Local loopback diagnostics route the serial data from the local HOTLink transmitter directly to the local HOTLink receiver as shown in *Figure 15*. This allows testing of the entire packet transmission and reception capability of the data-mover. It is enabled by setting bit 5 in the control register. This bit directly drives the local HOTLink receiver A/$\overline{B}$ input to select the local (B) connection, while disabling external transmission of serial data by asserting FOTO to the local HOTLink transmitter.

*Remote Loopback*

Remote loopback exists in both master and slave forms. The form is determined by where the request or directive for remote loopback operation originated. If bit 4 is set in the control register, the local interface becomes the master or controlling entity for the remote loopback operation.

In this mode the local transmit and receive paths route and handle data normally. When the local transmit path first determines that remote loopback has been requested it sends a single RLBON control code (C6.0) to the remote receiver.

Once the remote Control-PLD detects this C6.0 control code, it enters slave mode for remote loopback. The remote Control-PLD sets bit 3 in its status register, suspends normal data reception and transmission, and routes all received data characters directly to the transmitter for retransmission. This data routing occurs inside the remote Control-PLD as shown in *Figure 16*.

Remote loopback operations are suspended when bit 4 is cleared in the local control register. When this is sensed by the transmit path, it transmits a single RLBOFF control code (C7.0) to the remote receiver. Upon recognition of this code by the remote Control-PLD, it clears bit 3 in its status register, and returns to normal packet processing.

**Receive Path**

A block diagram of the receive path portion of this data-mover design is shown in *Figure 17*. The shaded portions of this figure are all internal to the Control-PLD. The data paths are marked with arrows, while the control signals have been left off for clarity.

*Operation*

Serial data enters the HOTLink receiver where it is framed to a character-rate clock and decoded into valid control codes and data characters. The parallel outputs of the HOTLink receiver (C/$\overline{D}$ and $Q_{0-7}$) are routed to a CY7C451 clocked FIFO. This FIFO is used for speed-matching between the HOTLink CKR recovered clock and the CKW clock used for transmit.

The $\overline{RDY}$ signal from the HOTLink receiver is used to filter out excess Sync/Fill control codes (C5.0) from the data stream. This is necessary to ensure that the input FIFO can never overfill and lose data. This $\overline{RDY}$ signal is actually routed through the Control-PLD to allow all FIFO writes to be disabled during FIFO master reset cycles.

The input FIFO data-outputs connect directly to the Control-PLD. The FIFO read enable is normally active (except during the FIFO master reset cycle), such that FIFO reads are enabled for every clock cycle. The FIFO outputs, along with

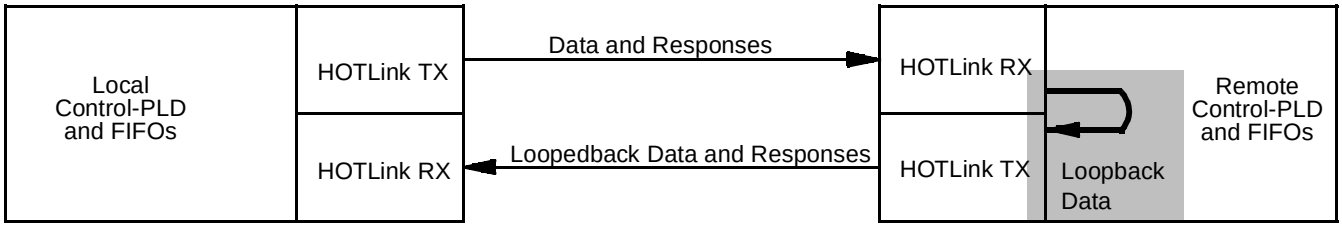



**Figure 16. Remote Loopback Data Flow**

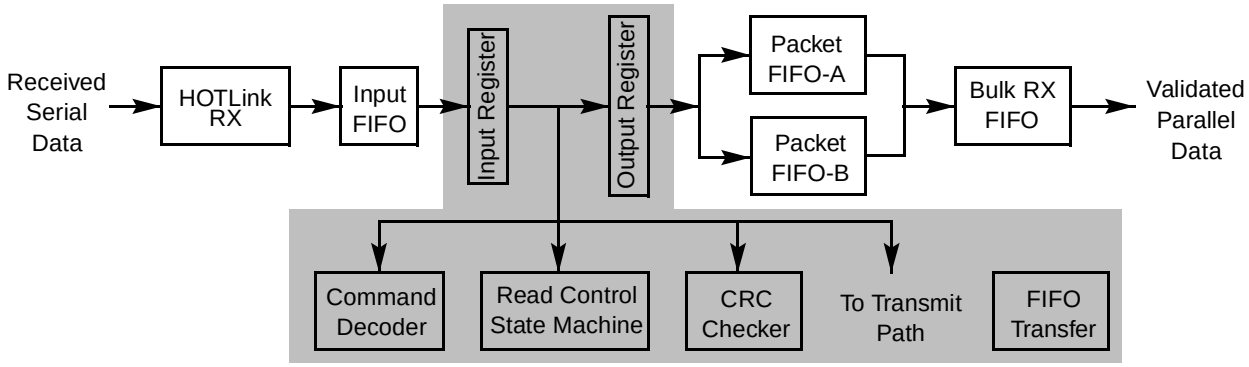



**Figure 17. Receive Path Block Diagram**

its status flags, are captured into the Control-PLD input register on every clock cycle. The FIFO flags are used to determine if a valid character is present in the input register, while the FIFOed SC/$\overline{D}$ signal is used to determine if the character in the input register is a data character or a special code.

*Read-Control State Machine*

The read-control state machine processes the incoming data stream on a character-by-character basis. It makes use of separate flip-flops to reduce the size of the state machine to ten states. These separate flip-flops keep track of the currently loading packet FIFO, and if any errors were detected in the middle of the current packet (received a C0.7 or C4.7, or an attempted write to a full packet FIFO). A state diagram of the read-control state machine is shown in *Figure 18*.

As with all state machines in this data-mover design, the read-control machine remains in a reset state until the reset condition no longer exists. It also enters this state whenever the local receiver is in BIST. Upon exit from reset, the machine performs a master reset to both packet FIFOs to clear any stale data. A single wait state is then forced to guarantee recovery from the master reset cycle.

At this point the machine is ready to process streams of read data from the input FIFO. As each character is received it is routed to the output register, a CRC checker, and a command decoder. For remote loopback functions, the data is also routed to the transmit path for retransmission.

The command decoder interprets all received characters that have the SC/$\overline{D}$ bit HIGH, indicating a control code. These decodes take place no matter what mode of operation the receive path is in. This allows the diagnostic and reset modes to be invoked even when the read-control state machine is waiting for a specific condition to complete in a packet transfer. This also allows any received ACK or NAK responses from the remote transmitter to be passed to the local transmit path, to allow pending packets to be sent or retransmitted.

One of these decodes, EOP (C0.0) is used by the read-control state machine to indicate that the last data character of the packet has been received. The following two data characters are not part of the packet content, but are instead the CRC bytes used to validate the packet. As each is received it is clocked into the CRC checker.

After the second CRC character has been received, the read-control state machine proceeds to the check packet state where the CRC register is examined. If the CRC register contains anything other than a 1D0F (hexadecimal), the packet is bad. During this same state any other pending errors (FIFO overflow, bad character, etc.) are also checked. If any of these errors or a CRC error are present the packet is rejected.

Rejection of a packet causes a NAK interrupt to be sent to the transmit path, as well as starting a master reset cycle on the receive packet FIFO containing the bad packet. The transmit-

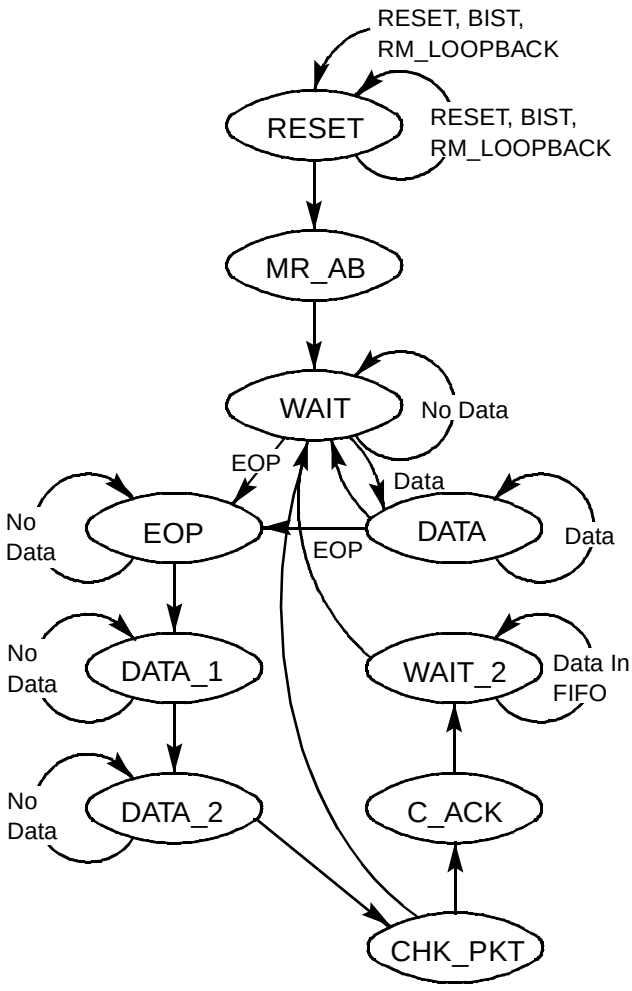



**Figure 18. Read-Control State Machine**

ter then sends a NAK (C2.0) control code to the other end of the link to initiate a retransmission of the bad packet.

If the CRC for the packet is valid (and there are no other packet errors), an ACK interrupt is sent to the transmit path instead. However, this interrupt may not be sent immediately. The read-control state machine must first make sure that the alternate (unloading) packet FIFO does not contain any data. If data is present, an XOFF interrupt is sent to the transmit path. This informs the remote transmitter to not start another packet. The state machine then advances to the C_ACK state, generating an ACK interrupt to the transmit path for the valid packet. This allows the remote transmitter to release its packet buffer. Following this the state machine advances to the Wait_2 state where it remains until the alternate packet FIFO is empty.

Once the alternate FIFO is empty, an XON interrupt is sent to the transmit path. At this same time the FIFO containing the newly validated packet, and the now empty alternate packet FIFO, are swapped. This allows the FIFO containing validated data to drain into the bulk receive FIFO, while allowing a new packet to be received into the now empty FIFO.

*CRC Checker*

The data-mover makes use of a 16-bit Cyclic Redundancy Check to validate each received packet. The CRC code used is the that defined by the ITU V.41 standard. The polynomial for this CRC is listed in Equation 1.

$$x^{16} + x^{12} + x^5 + 1 \qquad \text{Eq. 1}$$

This CRC checker is implemented in parallel to allow an entire byte to be accumulated in a single clock cycle. Because of its 16-bit span, this code can be used to protect packets of nearly 8 KBytes in length. The current CY7C451 maximum packet size of 512 bytes falls well within its capabilities. Following a reset or the end of a packet, the 16-bit CRC register is preset to an all-ones condition. This preset occurs during the CHK_PKT state of the Read-Control state machine.

With many CRC implementations, a packet is determined to be valid if the CRC register returns to an all-zero state at the end of the transfer. The CRC implemented here operates a bit differently. The CRC bytes (generated at the transmit end of the link) that are added to end of the packet are inverted (ones complement) prior to serialization and transmission. This causes the resulting syndrome at the end of the packet to be a 1D0F (hexadecimal) instead of 0000.

This additional inversion in the data stream is intended to improve the CRC detection of lost characters when the data stream contains trailing zero bytes, and the previous data bytes have caused the CRC register to revert to an all zeros condition. This also matches up with the usage of this same CRC code on the ESCON/SBCON (Enterprise System Connection/Single Byte command Connection) standard computer interfaces from IBM and ANSI. See the Cypress application note “Drive ESCON with HOTLink” for additional information on this CRC and interface.

*Receive FIFO Transfer State Machine*

The packet FIFO draining operation is performed by a FIFO Transfer state machine shown in *Figure 19*. Unlike the Read-Control state machine, data movement at this level exists completely outside the Control-PLD. This state machine operates by directing data, using the read enable and three-state output controls on the CY7C451 packet FIFOs, and by monitoring the FIFO status flags of the FIFO being emptied and the bulk read FIFO.

This machine also remains in the reset state as long as a reset event or BIST operation is active. Upon exit from reset, the bulk receive FIFO is reset to clear any state data and initialize its data pointers. Following this the machine moves to the WAIT_0 state.

In the WAIT_0 state the machine continuously monitors for an ACK interrupt from the read-control state machine. This identifies that the data in the packet FIFO is good, and that it can now be transferred to the bulk receive FIFO. A WAIT_1 state immediately follows this to allow the packet selector flip-flop to swap the incoming and output FIFOs.

Before advancing to the READ_XFR state, it is necessary to make sure that there is room in the bulk receive FIFO for data. If the bulk FIFO status shows it to be at or above an almost-full condition, the FIFO transfer state machine remains in the WAIT_1 state.

The READ_XFR state is where the work gets done. In this state a read enable is presented to the FIFO containing the validated packet, starting a read cycle. A single flip-flop is used to pipeline this read enable for a single clock cycle, where it becomes a write enable for the bulk receive FIFO. This single cycle delay is necessary because of the single cycle latency that exists on clocked FIFO read operations.

If at any time during the data transfer the bulk receive FIFO reaches an almost-full condition, the machine reverts to the WAIT_1 state. When all the data in the packet has been trans-

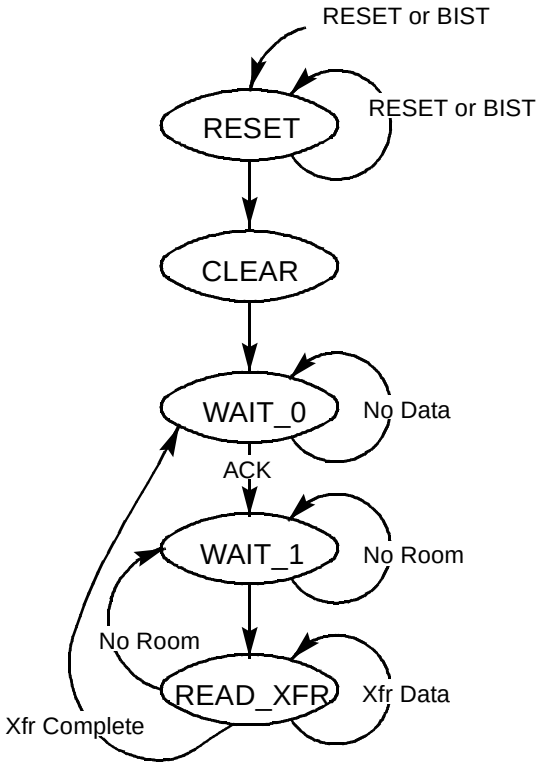



**Figure 19. Receive FIFO Transfer State Machine**

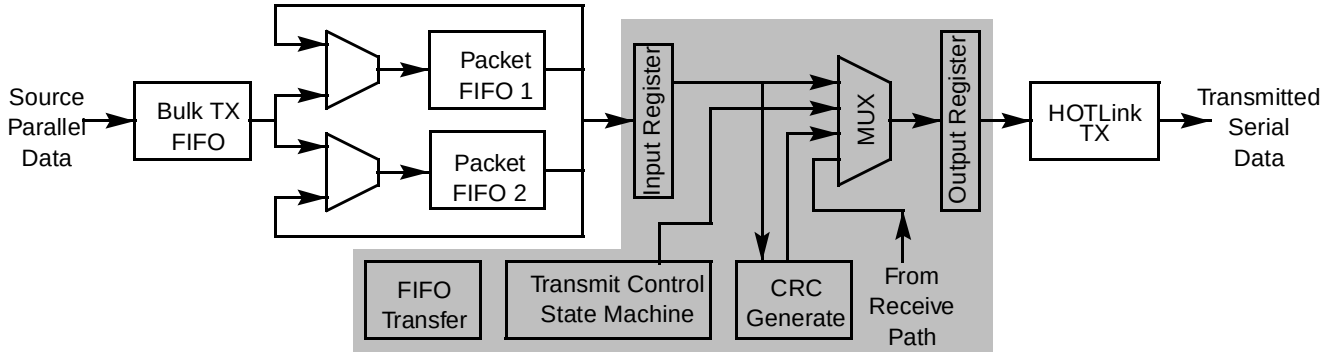



**Figure 20. Transmit Path Block Diagram**

ferred, the machine moves to the WAIT_0 state to look for the availability of the next validated packet.

**Transmit Path**

A block diagram of the transmit-path portion of this data-mover design is shown in *Figure 20*. The shaded portions of this figure are all internal to the Control-PLD. The data paths are marked with arrows, while the control signals have been left off for clarity.

*Operation*

Data transfers are initiated by writing bytes of data into the bulk transmit FIFO. This FIFO is sized to hold multiple packets. The data path here is actually nine bits wide. This ninth bit carries an odd-parity bit added to each byte by the host system. This odd parity is checked in the input register of the Control-PLD. Those implementations not using parity could remove this capability from the design to free up resources in the Control-PLD for other features. However, most of the XOR gates that comprise much of the structure are shared with the CRC generator that operates on the same data.

*Transmit FIFO Transfer State Machine*

When data is present in the bulk transmit FIFO, the transmit path version of the FIFO Transfer state machine becomes active. Its operation is similar to that of the receive path transfer machine, but with significant differences. The transmit FIFO transfer machine reads data from the bulk transmit FIFO and loads it into one of two transmit packet FIFOs. These packet FIFOs are the same CY7C451-type of clocked FIFO used for the receive path packets. The state diagram for this state machine is shown in *Figure 21*.

This state machine remains in the RESET state as long as a reset event is active. Upon exit from RESET, the bulk transmit FIFO is reset to clear any stale data and initialize its data pointers. Following this, the machine moves to the WAIT state.

In the WAIT state the machine continuously monitors for three conditions, all of which must be met to allow it to start a transfer. First the machine must be enabled. The enable signal is generated by the transmit-control state machine, which is in charge of the actual packet transfers to the HOTLink transmitter. This enable is necessary to make sure that FIFO transfers are not active when the packet FIFOs are being swapped. The remaining conditions are that there must be data in the bulk transmit FIFO, and room to put that data in the loading packet FIFO.

Once all these conditions are met the machine advances to the LOAD state. Here the read enable is driven active to the bulk transmit FIFO. This same read enable is delayed by a clock cycle and routed to the write enable of the loading packet FIFO. Following this read/write cycle, if any of the conditions necessary to enter the LOAD state are no longer true, the machine reverts back to the WAIT state.

*Transmit Packet FIFOs*

The transmit path also has the requirement of being able to retransmit a failing packet. Two methods are generally used to perform the retransmit function at a hardware level. The simplest is to use a FIFO with internal support for retransmission of the FIFO contents, like the CY7C42X5(V). However, not all FIFOs contain this capability.

The second method, implemented here, is to operate the FIFOs in a recirculate mode. As data is read out from one of the packet FIFOs it is written back into the same packet FIFO on the next clock cycle. This method can be used on any type of FIFO, but it requires external multiplexers to route the output data back to the FIFO inputs. A total of four CYBUS3384 bus-switch parts were used to perform this multiplexing function. These CYBUS3384 parts are controlled totally by the

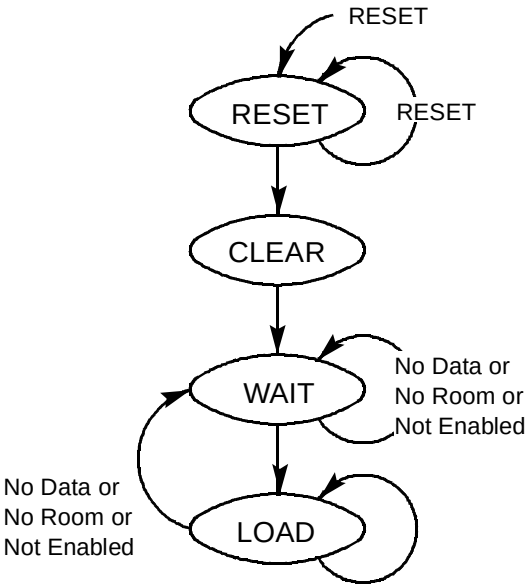



**Figure 21. Transmit FIFO Transfer State Machine**

present state of the packet selector flip-flop in the Control-PLD, and are independent of the transmit FIFO Transfer state machine.

*Packet Counters*

When loading data into a packet FIFO, the status flags are used to limit how much data is written into them. These same flags cannot be used when reading data out of the packet FIFO. The recirculating data keeps the number of bytes in the FIFO at a constant level. Counters located in the Control-PLD are used instead of status flags to track the beginning and end of the packet. The structure of these counters is shown in *Figure 22*.

Counter1 is an up-counter used to track the amount of data loaded into the packet buffer. It is enabled to count by the same signal used to enable the read from the bulk transmit FIFO. This counter is sized to nine bits to match the maximum depth of the CY7C451 packet FIFOs.

When the function of the transmit packet FIFO is swapped from load to transmit, the data count in Counter1 is captured in a holding register. The contents of this holding register are then loaded into Counter2.

Counter2 is also an up-counter, but it is used as a down-counter. It tracks the data read from the packet FIFO as it is sent out over the serial link. To operate as a down-counter, it is loaded with the ones complement of the count captured in the holding register. When the counter reaches its terminal count (CO = 1), all the data loaded into the packet FIFO has been transmitted. If a retransmit is necessary, the count is reloaded from the holding register.

*Transmit Control State Machine*

The Transmit-Control state machine is the most complex portion of the data-mover design. This state machine must

- Monitor the control register for all diagnostic modes
- Generate all control codes for all diagnostic modes
- Generate SYNC codes when no data are present
- Control the packet selector mux
- Control the load/recirculate counters
- Control the packet FIFO resets
- Route data during slave remote loopback
- Generate ACK/NAK responses for received data and remote BIST
- Generate XOFF/XON responses to receive-packet-buffer-full conditions

In addition to all these supervisory functions, the transmit-control state machine must generate properly formatted packets of data. While this last function sounds simple, proper operation required creation of a state machine with a non-conventional structure, shown in *Figure 23*.

This different structure was necessitated by the need to insert packet status responses, as generated by the receive path, into the transmit path data stream at any time. This effects all states of the machine (seventeen states *without* any ACK/NAK/XON/XOFF states) and, if not implemented as shown, would increase the number of states in the machine by a factor of five. While a machine of that size (85 states) is possible, the state encoder would be both very large and inefficient.

The transmit-control state machine is implemented in two sections. The first section determines the next-state of the machine, regardless of the presence of any packet response send requests. The second section interprets only the packet response requests.

The first half of the state machine is fairly conventional in nature. The one exception is the mux located just before the state register. This mux is effectively part of the next state encoder, but is shown here separately so that its function is more visible. This mux prevents the state machine from advancing to the next state if a request is present from the receive path to send an ACK, NAK, XOFF, or XON status response. The presence of this request is used in conjunction with the current state to inhibit actions normally enabled by the state machine when in a specific state.

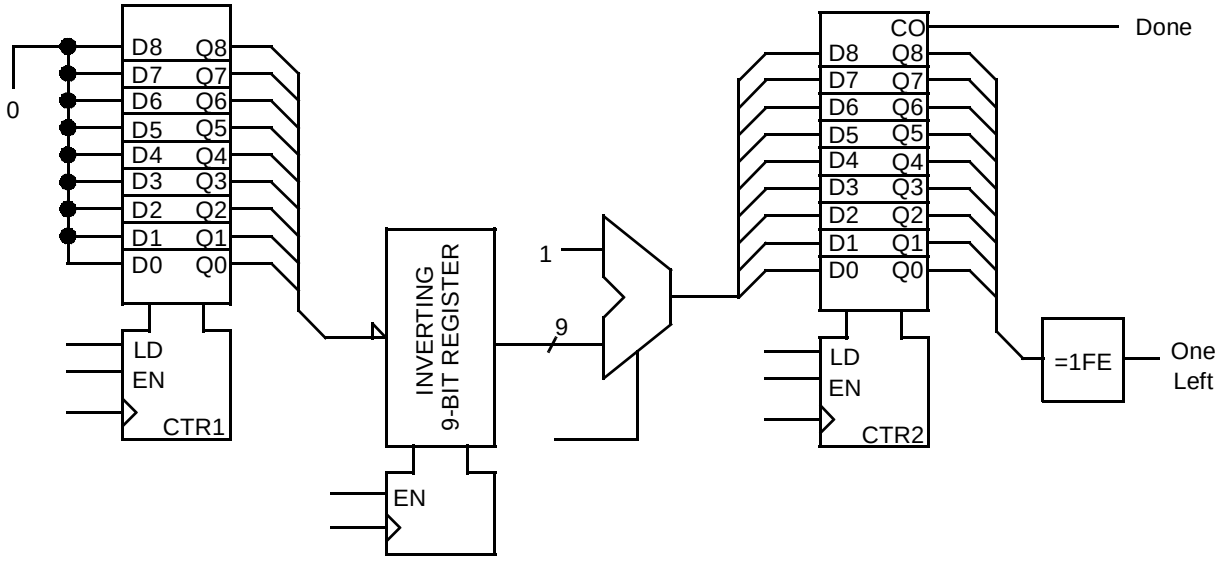



**Figure 22. Transmit FIFO Load/Recirculate Counters**

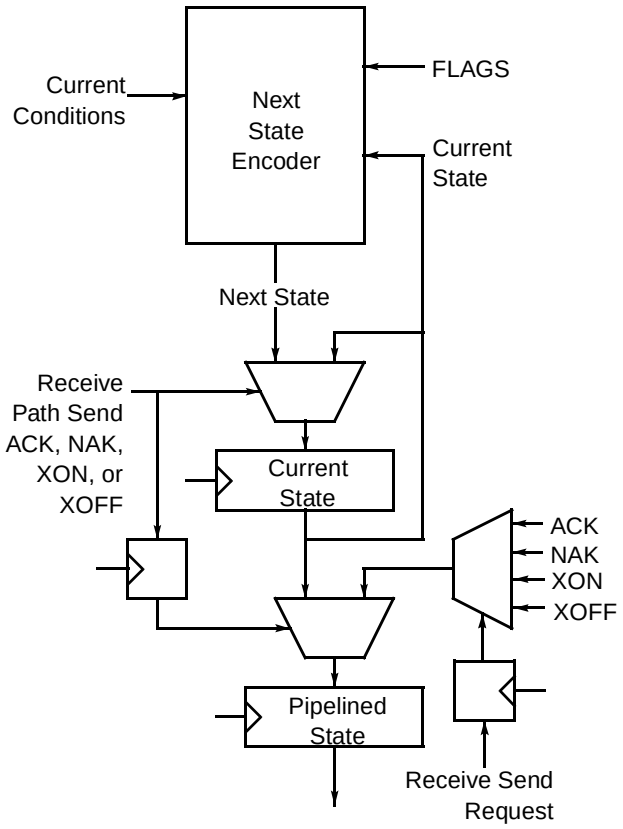



**Figure 23. Transmit-Control State Machine Block Diagram**

The second half of the state machine is effectively a pipeline stage added to the current state output. This pipeline is modified (if a response request is pending) by stuffing either the appropriate ACK, NAK, XOFF, or XON state into the pipeline register.

A state diagram of the output state machine is shown in *Figure 24*. While reduced to a total of seventeen states, the machine could be viewed as being much larger than this. Various state flags and counters are controlled by the state machine to allow existing states (like the four data transfer states) to be used multiple times. Both the XON/XOFF and RLOOP_ON/RLOOP_OFF pairs of states use state flags, set and cleared by the state machine, to make sure that only a single control code is transmitted when the enabling bit is set in the control register. The single T_RRESET state activates an external two-bit counter to remain in the state for four clock cycles.

Three states are used when a remote BIST operation is enabled by bit 4 of the control register. The first state (T_RBIST) transmits the RBIST_ON control code to the remote receiver, enabling the remote transmitter to generate BIST patterns. In the T_RBIST1 state the transmitter outputs a continuous string of SYNC codes, interrupted only by ACK or NAK responses at the end of each BIST loop. When bit 4 is cleared to end the remote BIST operation, the T_RBIST_ND state is entered, which transmits the RBISTOFF control code to the remote receiver.

The normal mode of data transmission uses the four data path states of T_DATA, T_EOP, T_CRC_HI, and T_CRC_LOW. This loop has the lowest priority, and all the diagnostic functions are at a higher priority level. If no diagnostic operations are pending, and data is available (as indicated by a non-zero DONE output from packet counter CNTR2), the machine enters the T_DATA state. It remains in this state until CNTR2 indicates only a single byte remaining. Because a FIFO read is still enabled when only one byte is left, the state machine leaves the T_DATA state after the last byte has been read from the FIFO.

At this point the T_EOP state is entered. In this state the last byte read from the FIFO is now in the input register and is being accumulated in the CRC generator. This same state directs the output mux to output the EOP control code.

Following the T_EOP state, the two CRC bytes are output. This occurs during the T_CRC_HI and T_CRC_LOW states. During these states the CRC generator is disabled, and the upper and lower bytes of the 16-bit CRC register are inverted and passed to the HOTLink transmitter. As the last CRC byte its output, the CRC register is preset to the all ones state for the next packet.

*State Encoding*

The pipelined state register contents are used to determine what data gets sent to the output register. It controls the output character-wide 4-to-1 multiplexer, as well as providing the proper bits for any necessary control codes. Rather than allow a random bit assignment of the states in the state machine, they were selected to minimize the translation necessary

**Table 4. State vs. Command Encoding**

| State | | Control Code | |
|---|---|---|---|
| **Name** | **Encoding** | **Name** | **Encoding** |
| t_reset | 00000 | | |
| t_rcomp | 10111 | RCOMP | 00001011 |
| t_wait | 10000 | | |
| t_data | 01000 | | |
| t_eop | 00001 | EOP | 00000000 |
| t_crc_hi | 00100 | | |
| t_crc_low | 10100 | | |
| t_xon | 00111 | XON | 00000011 |
| t_xoff | 01001 | XOFF | 00000100 |
| t_ack | 00011 | ACK | 00000001 |
| t_nak | 00101 | NAK | 00000010 |
| t_exloop | 00010 | | |
| t_exloop1 | 10010 | | |
| t_rloop_on | 01101 | RLBON | 00000110 |
| t_rloop_off | 01111 | RLBOFF | 00000111 |
| t_rbist | 10001 | RBISTON | 00001000 |
| t_rbist1 | 00110 | | |
| t_rbist_nd | 10011 | RBISTOFF | 00001001 |
| t_rreset | 10101 | RRESET | 00001010 |
| | | SYNC | 00000101 |

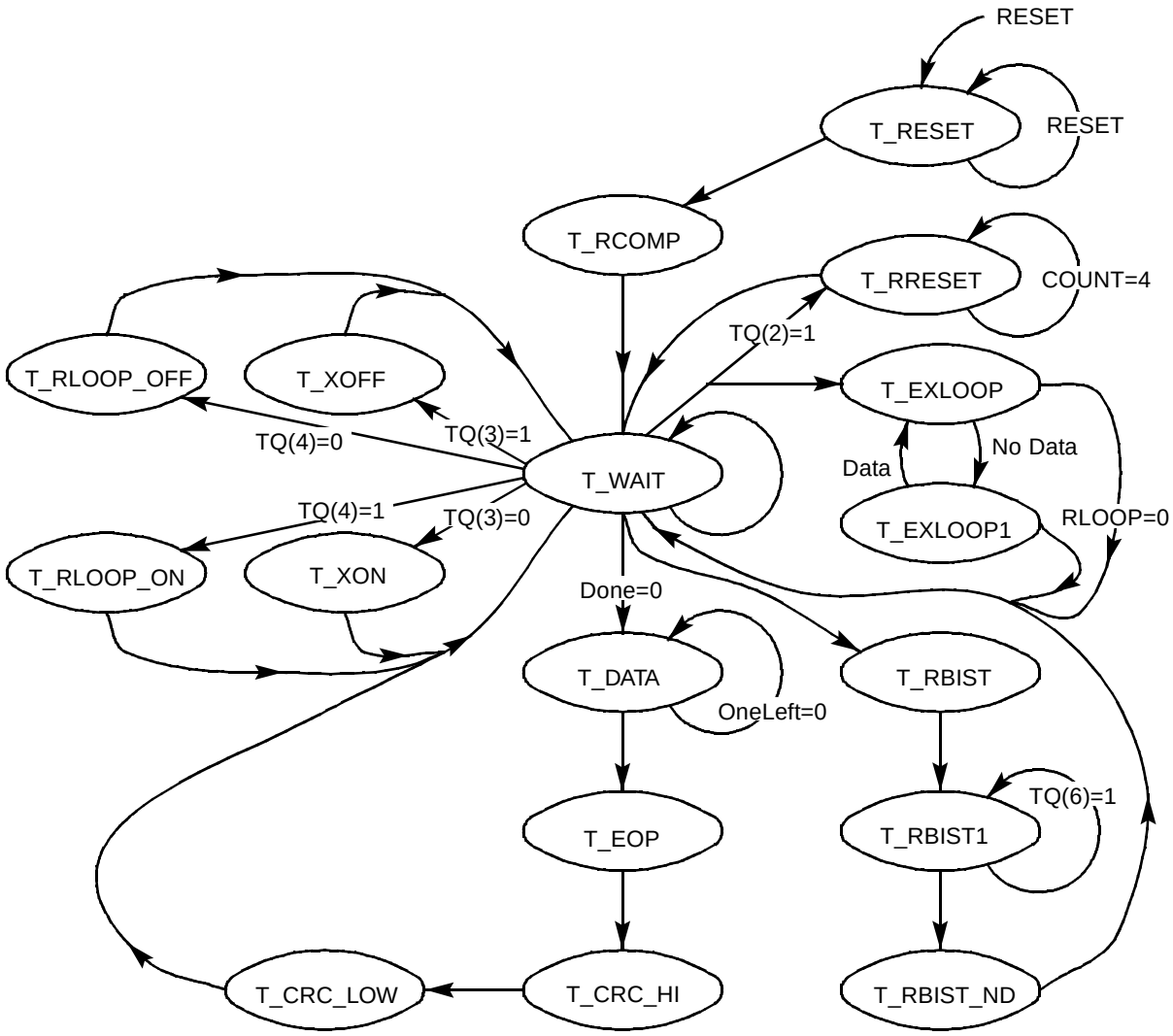



**Figure 24. Transmit-Control State Machine State Diagram**

between the states and any generated commands. This mapping is shown in *Table 4*. Those states that cause command codes to be transmitted have the resulting control code (and its encoding) listed in the adjacent columns.

Close examination of the state encodings shows that the right-most bit in the field (LSB) is only active (1) during those states where a control code is generated. This allows this single bit to be used to control part of the output data multiplexer. Comparing the remaining bits in the state encoding field with the four low order bits in the encoded control codes also shows an exact match. This allows control codes to be output from the output multiplexer without the addition of a separate logic block to convert from the state encodings to the control code encodings.

*CRC Generator*

The CRC generator is very similar to the CRC checker used in the receive path logic. The XOR terms used to feed the 16-bit CRC register are identical to those in the receive path. The primary difference comes in the output side of the CRC register.

In the receive path, the register feeds a 16-bit equality comparator that checks for a valid remainder. In the transmit path, the output feeds a 2-to-1 byte-wide multiplexer. The high-order byte of the register is sequenced out first, followed by the low-order byte.

*Output Multiplexer*

The output multiplexer has four sources of data:

- Control codes from the transmit-control state machine
- Packet data from the FIFOs and input register
- CRC bytes
- Receive path loopback data

Three of these sources are directly selected by states in the transmit-control machine. The T_EXLOOP state selects remote loopback data from the receive path. The T_DATA state

selects transmit data from the input register. CRC bytes are selected in the T_CRC_HI and T_CRC_LOW states.

The remaining condition is the default select for the multiplexer. Here it always outputs control codes. But one control code, SYNC, is not directly associated with a state. This code is output under all other conditions. This is implemented by adding a second multiplexer (not shown in *Figure 23*) to the lower four bits of the control code input (the upper four bits of all control codes are zeros). This multiplexer uses the LSB of the encoded state field. In the default state (LSB = 0) a value of 0101 is routed to the low order four bits of the control code to generate a SYNC (C5.0) control code. The alternate state (LSB = 1) directs the remaining bits of the state vector to the same bits in the control code.

## Design Implementation

### Control-PLD

The Control-PLD for this data-mover design was implemented in a pASIC FPGA. It was designed and simulated in its entirety using Cypress's *Warp3*® VHDL compiler.

The design is structured as a hierarchy of VHDL source files, each containing one or more components. This design hierarchy is shown in *Appendix A*. The label attached to each box is the name of the associated VHDL source file. The top bar of each box lists the package contained in the file. The remaining names under this are the components within each package.

The top-level of the design hierarchy is the MVR_PINS.VHD file. This file contains the pin number assignments, and instantiates the logic for the receive, control, and transmit paths.

Once compiled and programmed into the pASIC IC, the Control-PLD is integrated with the associated CY7C451 and CY7C453 FIFOs, CYBUS3384 bus switches, and a CY9266 HOTLink evaluation board to form one end of a data-mover.

### Data-Mover Schematic

The complete schematic for the data-mover is shown in Appendices B through F.

Appendix B contains the power distribution and handling for the design, including the bulk power-supply filtering and bypass capacitors, power input connector, and sacrificial Zener diode.

Appendix C contains the Control-PLD and speed matching FIFO for the receive data path. A connector is also present to accept a CY9266 HOTLink evaluation board. The routability of the pASIC FPGA allows the connections between the Control-PLD and the connector J3 (the CY9266 connector) to be routed without crossovers or vias.

Appendix D contains the bulk and packet FIFOs and routing logic for both the transmit and receive data paths.

Appendix E shows the stimulus generator for testing the data-mover. It contains a CY7C374 Flash programmable CPLD, configured to exercise the data mover. Switches and indicators are also present to allow the control and status registers in the Control-PLD to be configured and monitored.

Appendix F is the top level of the schematic and shows the interconnections to four sheets. These sheets are detailed on the remaining pages of the schematic.

A complete parts list is provided in Appendix G.

## Design Enhancements

The data-mover implemented in this design was not intended to meet the performance, functionality, and reliability needs of all users. However, in its present form, the design presents a basic framework of VHDL modules that can be enhanced with minimal effort to expand, contract, or otherwise modify the present design to meet specific requirements.

The pASIC FPGA used in this design is relatively full, with over 90% of available gate and register resources used. With the design done completely in VHDL, it is a relatively simple activity to target other Cypress CPLDs or FPGAs. Some of the areas that might see changes are

- Host bus interface width
- Additional status/control bits
- Additional types of resets
- Conversion from variable to fixed packet length
- Change from 16- to 32-bit CRC
- Remote read of status and configuration registers
- Additional packet buffers and out of order response and packet handling

### Host Bus Interface

The present host bus width is set to only a single 8-bit path. This may not match up well with newer advanced processors. While a 32-bit control bus appears to be overkill, a 16-bit bus is quite reasonable.

### Status/Control Bits

Widening the host bus to a 16-bit path would also double the number of control and status bits available. These additional bits could be used to allow more type or diagnostic and reset operations, and well as provide additional status information as to how the link is operating.

### Resets

The present design only supports a single level of reset for either end of the link. This reset is quite catastrophic in nature as it clears all data staged in all packet FIFOs. Enhanced reset types that could be added to limit the effect of the reset to either the transmit or receive path of either end of the link, or clear specific packet buffers without disturbing the remainder of the data.

### Fixed Packet Length

Many standard protocols are based on fixed packet lengths rather than the variable packets used in this design. Fixed packet lengths have the advantage of operating with the greatest link efficiency, though often with greater latency due to the time to accumulate the entire packet prior to starting packet transmission.

The primary changes to the existing design to accommodate fixed packets would be in the transmit FIFO transfer state machine. The Counter1 that is used to track the data being loaded would also add an equality comparator to check for a specific packet size. If desired, the value compared to could be made loadable through a write operation from the host bus interface.

The status presented by Counter1 to the transmit control state machine would also need to be changed. It presently indicates if there is any data present in the loading packet,

and it would need to change to show that the full packet is available.

**32-Bit CRC**

The 16-bit CRC used in this design, while a common polynomial, may not meet the detection requirements of some users. Alternate CRCs may be substituted by replacing both the transmit and receiver CRC modules with equivalent modules based on a different polynomial. If the polynomial remains at 16 bits, then no other changes are necessary.

Changing to a 32-bit CRC would require replacement of the transmit and receive CRC modules and changes to the transmit and receive state machines. These state machines presently allocate only two time slots for sequencing out and in of the CRC bytes. The most common CRC polynomial for 32-bit applications is listed in Equation 2.

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \quad \text{Eq. 2}$$

This is the same CRC specified for use in FDDI (Fiber Distributed Data Interfaces) and Fibre Channel. Because of the larger register count and associated increase in XOR terms, usage of this CRC will require moving to a larger programmable device to accommodate the extra gates.

**Remote Status Access**

The ability to read the status registers at the remote end of the link would also be beneficial for link diagnostic functions. However, the changes necessary to allow this are quite significant and would affect nearly every module in the design.

Since all valid control codes are presently allocated, the present command structure would need to be altered to allow use of multi-character sequences to represent additional commands. These new sequences of control codes would allow creation of a new command that could be used to instruct the remote node to transmit its status register contents. This would involve substantial changes to the transmit and receive state machines, as well as the command decoder.

Since the status information would effectively be data characters, a method must be created to differentiate these characters from the normal packet data transfers, and storage registers must be provided internal to the Control-PLD to contain them. A new fixed packet structure of two or more characters in length would also have to be defined. This structure would start with a special control code (or codes), and the immediately following data characters would contain the status information. This would allow the receive path state machines to route these status characters to the internal registers and not into the receive packet FIFOs.

**Additional Packet Buffers**

Of all the previously described possible enhancements to this data-mover design, the addition of a third packet FIFO to both the transmit and receive paths would be the only change to have a significant impact on link performance. This third FIFO would allow the transmit end of the link to start transmission of a second packet, prior to reception of an ACK/NAK response for the packet just sent.

To handle this correctly, including the support for retransmission of failed packets, both the packet format and link protocols would require significant overhauls:

- A preamble or header would need to be added to each frame to allow the receiver state machine to know if a packet was lost or received out of order
- Packet responses would need to be changed to allow identification of which packet is accepted or which should be retransmitted
- Error handling would be more complex because packet responses would now need to be checked for out-of-sequence reception and transmission
- Additional status bits would be necessary to track or report these sequence errors

This change would provide a significant boost to link efficiency and latency by removing much of the link transmission delay overhead from each packet transfer. As the link length increases, minimizing this transmission overhead becomes increasingly more important to maintaining link performance.

## Conclusion

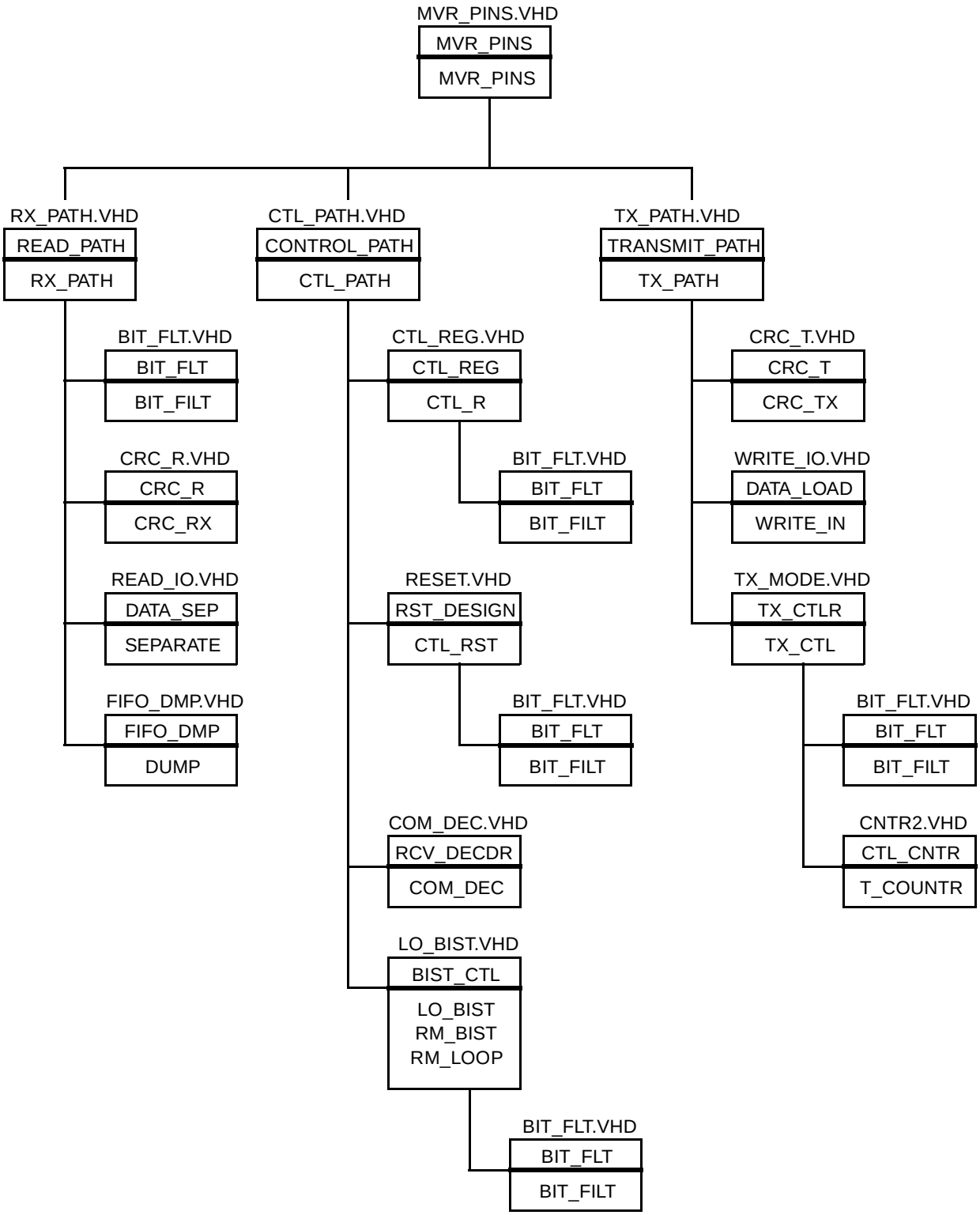
This data-mover design is not targeted to any specific application. The design embodied in this application note is intended to show the design process necessary to build a robust and reliable link. Not all capabilities implemented in this design are necessary for all applications.

All VHDL source files for the Control-PLD are available from the Cypress web site (www.cypress.com). These files, with minor modifications, can be used to implement other data-movers targeted to specific applications.

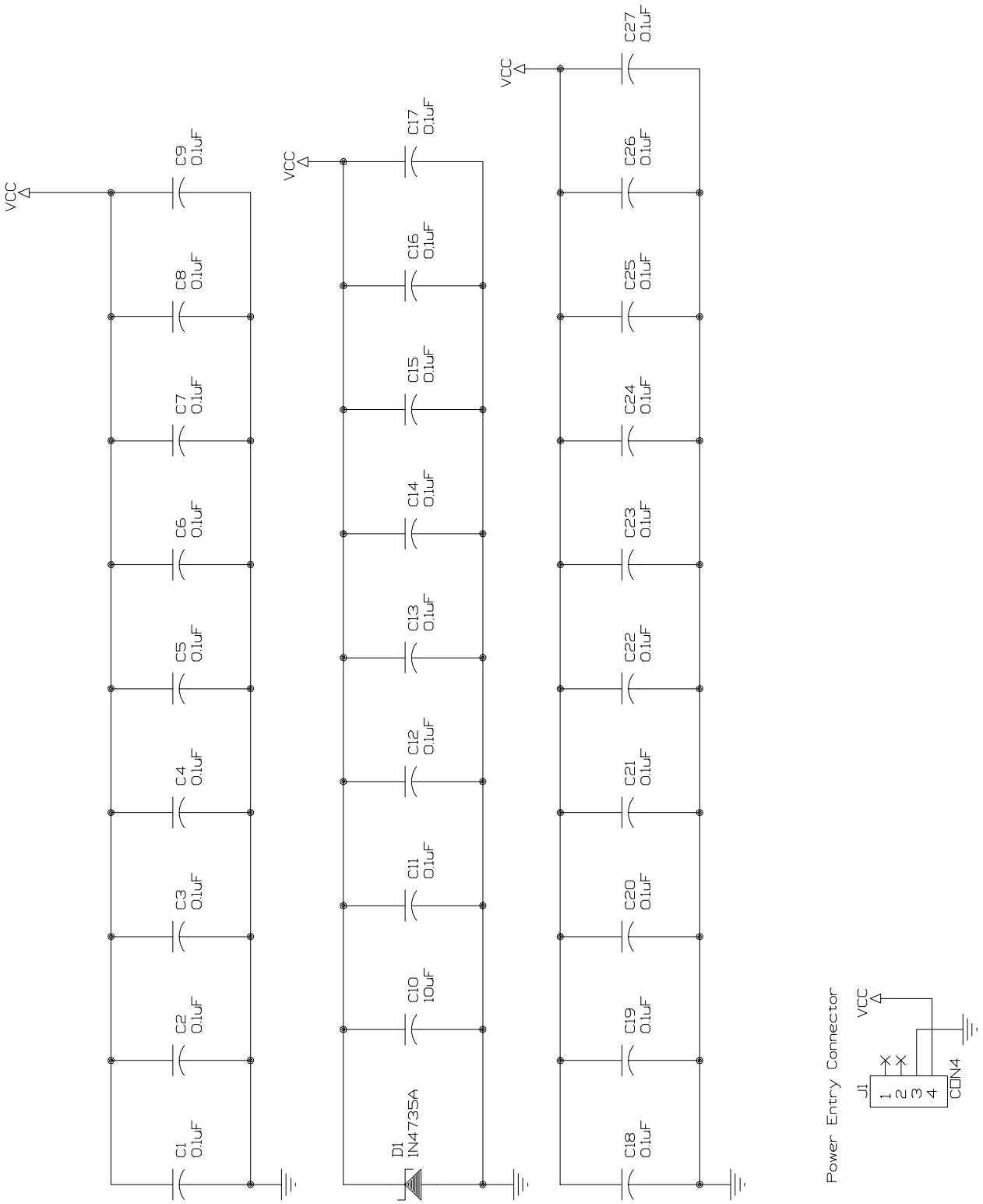
## References

1. *Cypress HOTLink User's Guide*, Cypress Semiconductor, April 1999
2. ESCON I/O Interface, IBM Corporation, 1990, 1991
3. SO/IEC 9314-2:1989, Fiber Distributed Data Interface - Media Access Controller (FDDI-MAC)

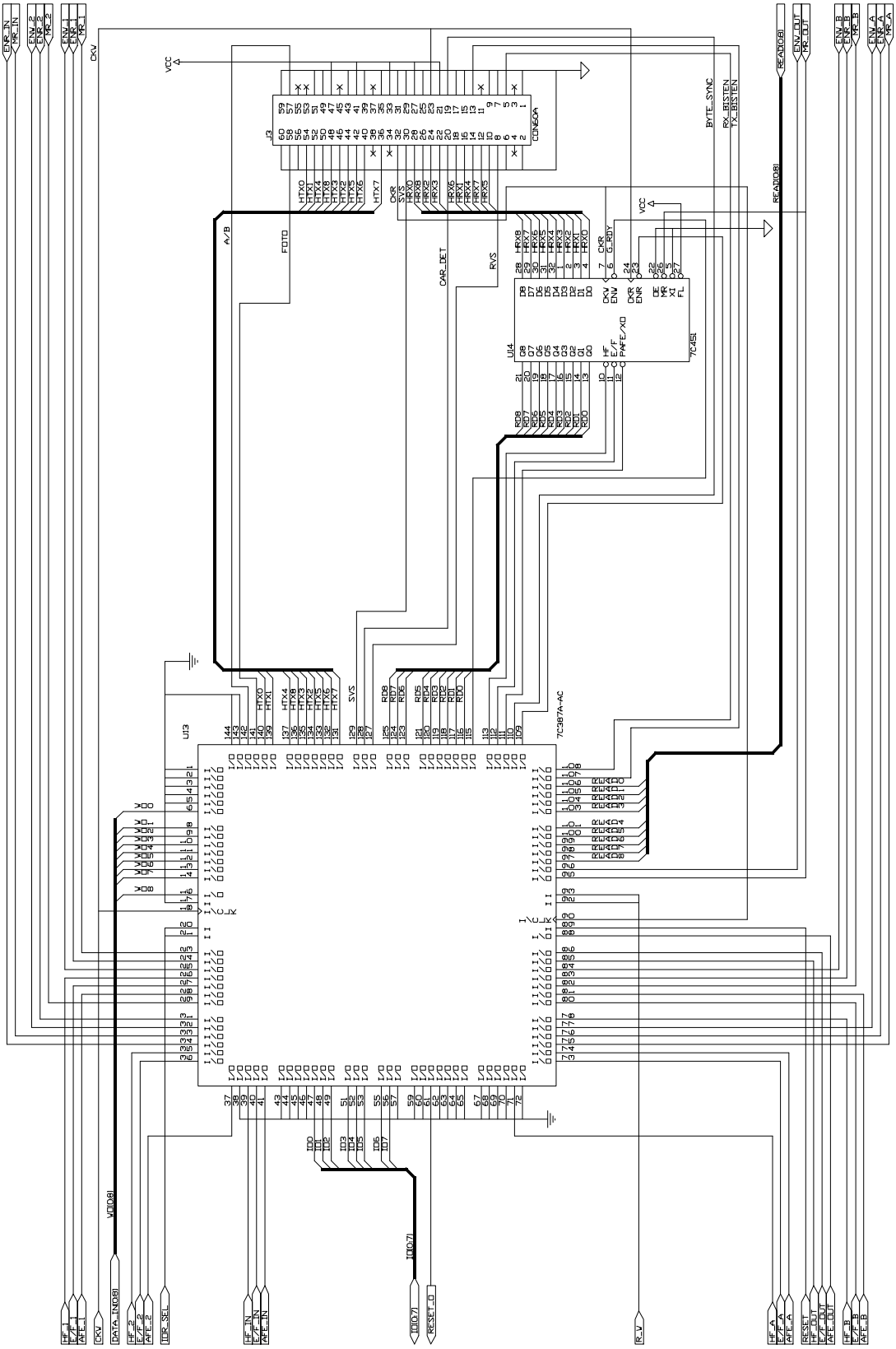
## Appendix A. Control-PLD Design Hierarchy



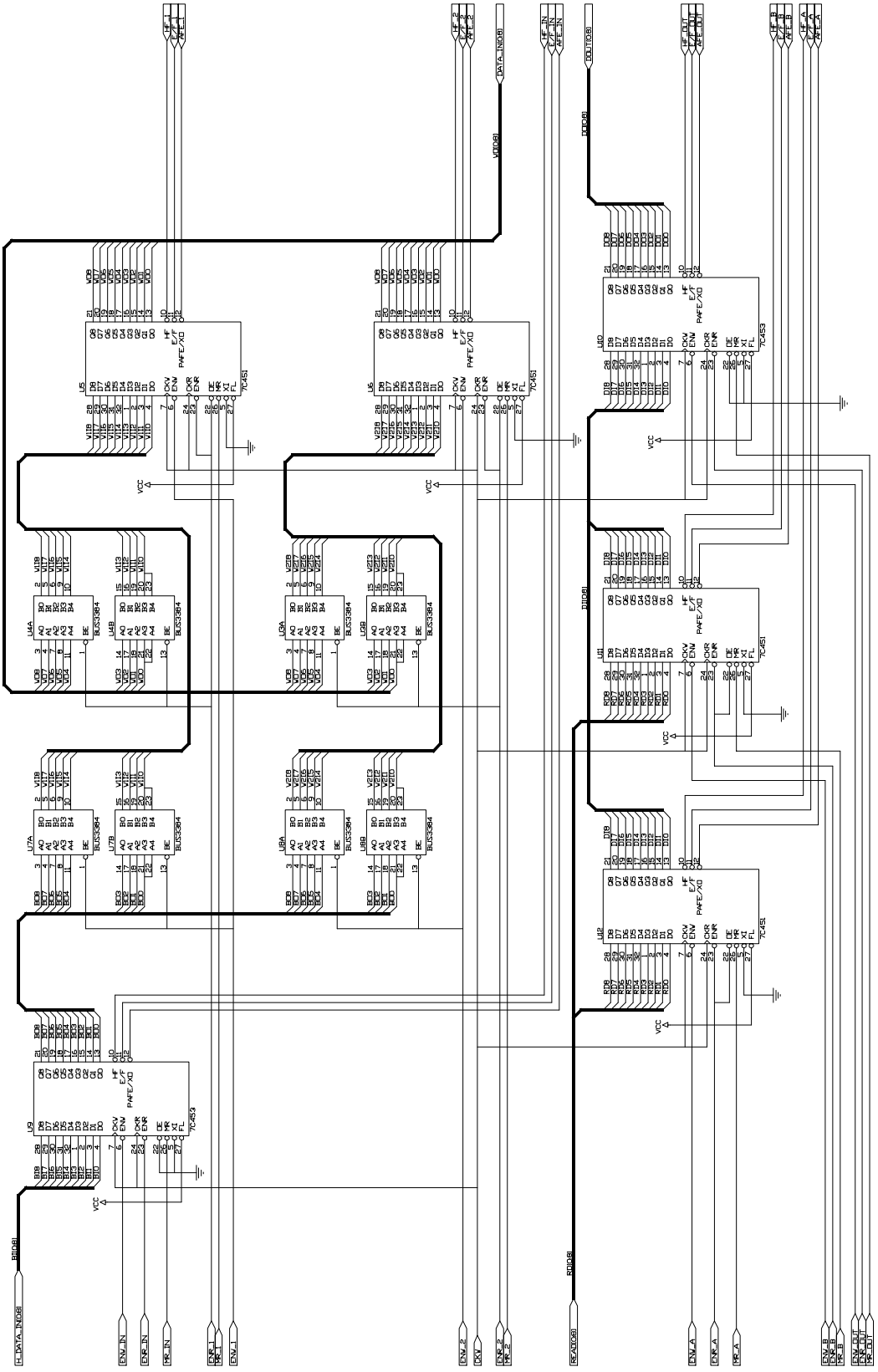
## Appendix B. Data-Mover Power and Power Filtering



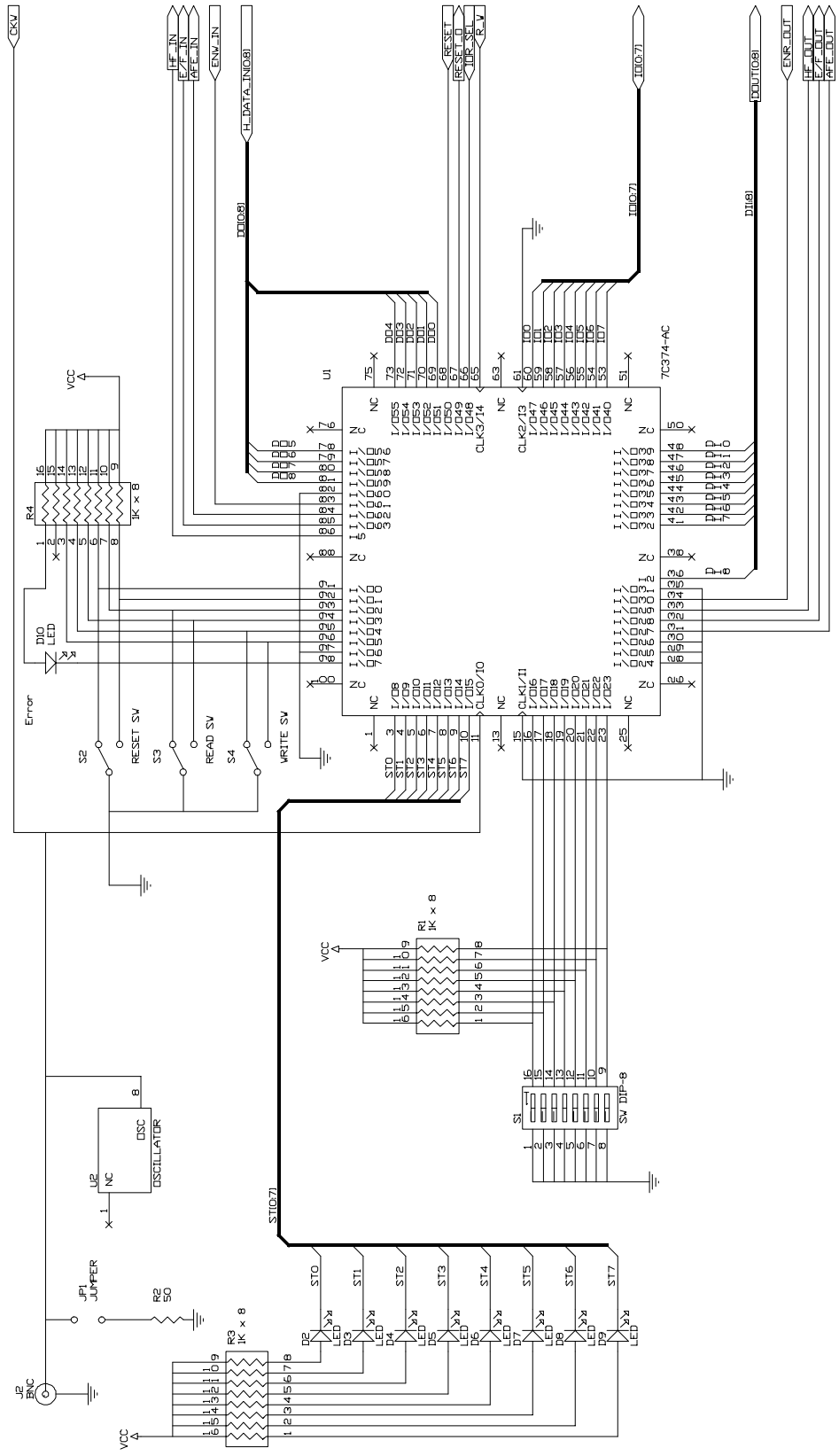
## Appendix C. Data-Mover Control-PLD, Receive Input FIFO, and CY9266 Connector



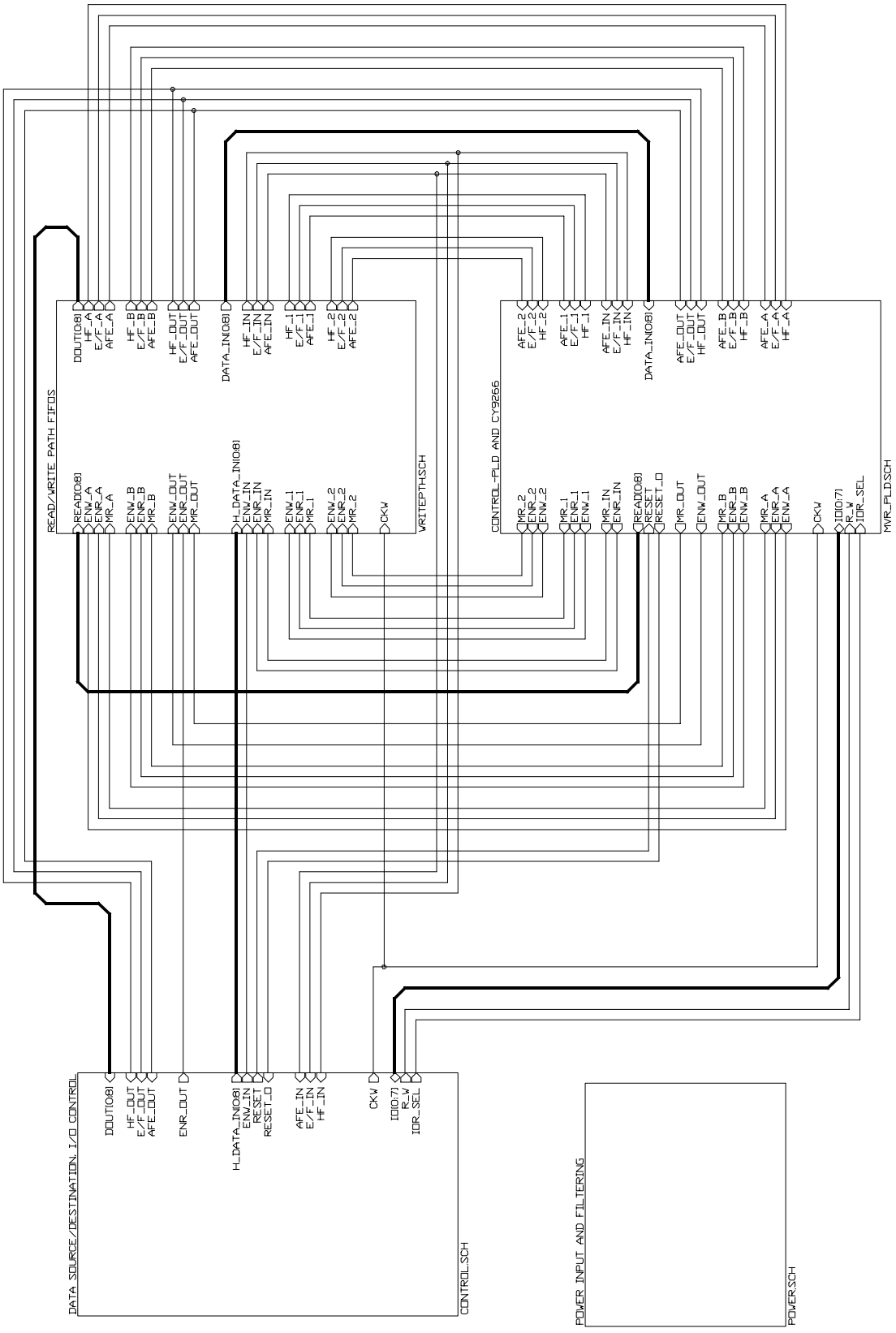
## Appendix D. Data-Mover Receive and Transmit Path Packet and Bulk FIFOs



## Appendix E. Data-Mover Stimulus Generator and Test Fixture



## Appendix F. Data-Mover and Exerciser Top-Level Interconnect



## Appendix G. Data-Mover Parts List

| Instance | Part Number | Description |
|---|---|---|
| U1 | Cypress CY7C374 | UltraLogic 128-Macrocell Flash CPLD |
| U2 | CTS CTX126 or Equivalent | 25-MHz TTL Clock Oscillator |
| U3, U4, U7, U8 | Cypress CYBUS3384QC | Dual 5-Bit Bus Switches |
| U5, U6, U11, U12, U14 | Cypress CY7C451-20JC | 512 x 9 Cascadable Clocked FIFO with Programmable Flags |
| U9, U10 | Cypress CY7C453-20JC | 2K x 9 Cascadable Clocked FIFO with Programmable Flags |
| U13 | QuickLogic QL16x24B | Very High Speed 4K (12K) Gate CMOS FPGA |
| D1 | 1N4735A | 1W, 6.2V Zener Diode |
| D2, D3, D4, D5, D6, D7, D8, D9, D10 | LITEON LT1034 | T1 Red LED |
| S1 | AMP 3-435640-9 or Equivalent | 8-position DIP Switch |
| S2, S3, S4 | C&K 8125SD3V3BE or Equivalent | SPDT Momentary Switch |
| JP1 | Sullins PZC1DAAN or Equivalent | 2 x 1 Position 0.25″ Sq. Pin-Header |
| J1 | AMP 641737-1 | 4-Pin Power Connector |
| J2 | AMP 227161-3 or Equivalent | RA Female BNC Connector |
| J3 | Sullins EZC30DRXH | 2 x 30 Edgeboard Connector |
| J4 | Sullins PZC1DAAN or Equivalent | 2 x 20 Position 0.25″ Sq. Pin-Header |
| C1, C2, C3, C4, C5. C6, C7, C8, C9, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23, C24. C25, C26, C27 | 0.1MLC X7R | 1206 Chip Cap |
| C10 | 10 μF 16V Tantalum | Electrolytic Cap |
| R1, R3, R4 | CTS 766-163-R1K or Equivalent | 1-kΩ R-Pack-8 SO16 |
| R2 | 50Ω 1/8W | 1206 Chip Resistor |
| | 3M 929955-06 or Equivalent | 1 - 0.1″ Centerline Shorting Jumper |
| | Cypress CY9266-F, -T, or -C | HOTLink Evaluation Board |