FPGA Applications Guide

Telecommunicafions and Networking Edition

Actel Corporation, Sunnyvale, CA 94086

Part Number: 5192609-0 February 1995

Trademarks

Table of Contents

Introduction

Telecommunications and Networking is a vast industry from both a business and technology perspective. This global market is growing at 10-15% annually and hit over $500 billion in 1990s. Equally impressive is the rapid innovations and diversity of the technology used to allow people to communicate with voice, data, image and video across the globe. FPGAs are one of those enabling technologies.

The evolutionary nature of developing telecommunications standards as well as the diminishing product cycle times demand high speed/high capacity devices that can be designed in weeks. These are the attributes that have catapulted FPGAs to the forefront of the telecom engineer's toolbox. Actel offers four distinct device families targeted at different design needs. The ACT 1 family offers low cost integration up to 2,000 gates. ACT 2 and 1200XL families provide best value high capacity up to 8,000 gates and 50 MHz. ACT 3 offers high capacity and high speed allowing telecom applications up to 10,000 gates to operate beyond 75MHz.

This FPGA Applications Guide uses the ACT 3 devices to show three specific Telecommunications and Networking applications. Each design provides detailed information on the application and the design methodology referencing many diagrams. A disk is appended at the end of the Applications Guide which contains all the design files needed to use the files in your next design. Please reference these files to get a clear and concise view of the schematics. The first two designs were captured using the Viewlogic schematic editor. All the library elements used in the design are included on the disk so you do not need to have a complete Actel system to use the designs. The third design was written in Verilog HDL and the complete source file is included on the disk.

The first application describes how to implement the 100Base-X 100 Mbit/s Ethernet Standard in an Actel FPGA. There is an in-depth discussion of the sub-layer functions referencing the design files so that any required changes to meet your needs can be easily accomplished.

The second application presented is a high speed Asynchronous Transfer Mode (ATM) Switch Fabric. ATM is a relatively new innovation and used to optimally process packets at higher rates. The Actel multiplexer based architecture is ideally suited for ATM applications. Again, full design details are included to allow you to modify this design for your needs.

The third application describes the design of a Cyclic Redundancy Check (CRC) for the IEEE standard 802.3 Local Area Network interface. CRC offers a way to detect small changes in blocks of data ensuring integrity during transfers. This design is implemented in Verilog HDL. Schematics were generated to show the modules utilized in the design.

For further technical information, contact Actel using your favorite means of communications:

Technical Hotline	800 262-1060
FAX	408 739-1540
E-mail	tech@actel.com

Using Actel FPGAs to Implement the 100 Mbit/s Ethernet Standard

One of the more recent entrants into the high-speed networking standards battle is 100Base-X—Ethernet operating at 100 Mbit/s. This standard is supported by the Fast Ethernet Alliance and sponsored by several key networking companies such as Intel, National Semiconductor, Sun Microsystems, and 3Com. This proposed standard involves many of the types of digital logic functions facing high-speed network designers and, as will be shown in this application note, can be readily implemented using Actel FPGA devices.

The emerging 100 Mbit Ethernet market is expected to mushroom as network performance requirements continue to grow. Network users are expected to have almost doubled between 1991 and 1994, and networks will need to provide these new users with just as much (if not more) bandwidth. A high-speed Ethernet network could solve the bandwidth problems for many classes of users while maintaining compatibility with current equipment and software.

100Base-X Network Standards

The 100Base-X proposal uses two established networking standards to support the 100 Mbit data rate required to implement the tenfold increase in the 10 Mbit rate of the current Ethernet standard. The 100Base-X standard keeps the Media Access Control (MAC) layer the same as the current Ethernet standard, but it raises the data rate to 100 Mbit/s. Since the MAC layer was defined independently of performance level, this increase can be accomplished relatively easily, and the well-proven behavioral dynamics of the Ethernet MAC can be retained. The only change required is to reduce the physical network span to 1/10 of the 10 Mbit/s distance, resulting in a span of about 250 meters.

This reduced span fits well within current structured wiring methodologies. Building-floor wiring in modern installations of Ethernet, such as 10Base-T, are organized as physical stars with a centralized wiring closet and cable runs of less than 100 meters. For LANs, this results in a hub-station architecture with interconnections of less than 100 meters.

At the physical layer, 100Base-X leverages off the proven FDDI standard for 100 Mbit/s communications using a full-duplex 125 Mbit/s Physical Media-Dependent (PMD) sublayer. This supports fiber optic, shielded twisted-pair (STP) and unshielded twisted-pair (UTP) wiring. Combining the MAC layer of Ethernet to the PMD layer of FDDI requires a convergence sublayer (CS) between them. Using the CS, 100Base-X maps the PMD's constant signaling system to the packet-oriented half-duplex system imposed by the Ethernet MAC.

Convergence Sublayer Interfaces

The MAC transmits data to the convergence sublayer in the form of 4-bit words (Figure1). This data is then encoded into 5-bit groups, serialized, and transmitted by the CS to the PMD sublayer as the transmitPMD signal.

Received data is sent from the PMD to the CS as the receivePMD signal and is synchronized with the 125 MHz clock. Note that the PMD also generates signalDetect when data is detected on the line. The CS decodes the serial data, converting the input 5-bit code groups into 4-bit hex characters and sends it to the MAC as the receiveMAC signal. Note that the PMD extracts the clock from the serial bit stream input. The 125 MHz frequency is recovered from the input data stream by the PMD clock circuits in the CS. In addition, receiveError is generated by the CS to indicate to the MAC that an error has occurred during reception. The carrierSense signal is provided to the MAC to indicate that the line is active. The collisionDetect signal notifies the MAC if a collision has occurred.

This application note will show you how to use Actel FPGAs to develop a complete convergence sublayer. It will subdivide the CS into its functional divisions and will show you how each can be implemented using Actel ACT 3 FPGAs.

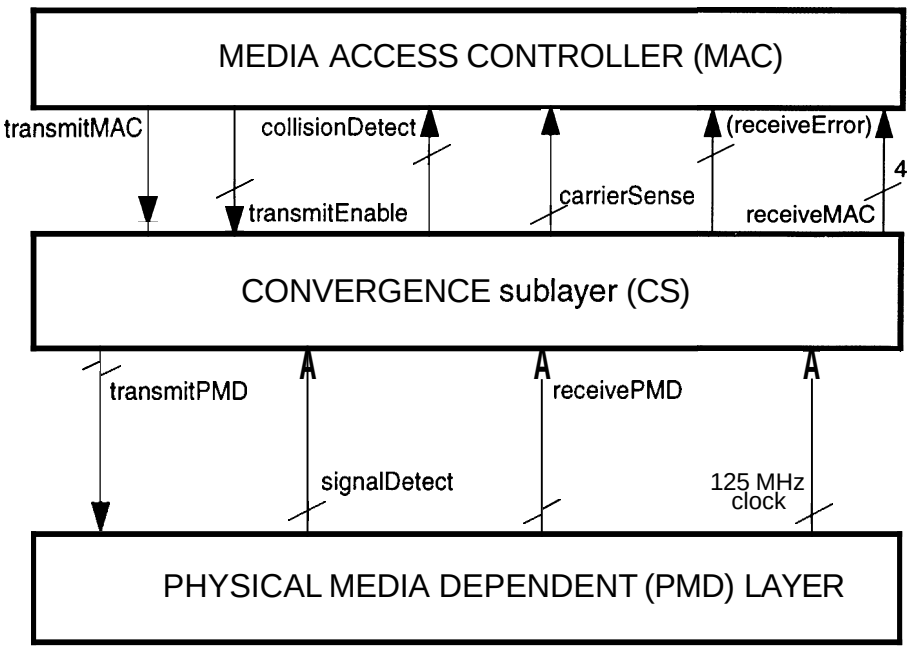


Figure 1. Convergence Sublayer Interfaces

Convergence Sublayer Functions

Figure 2 shows the basic dataflow in the convergence sublayer. The CS receives transmit data from the MAC as 4-bit words designated transmitMAC. These 4-bit words are encoded into 5-bit symbols (designated TxSYM) that are shifted out to the PMD at the 125 MHz clock rate.

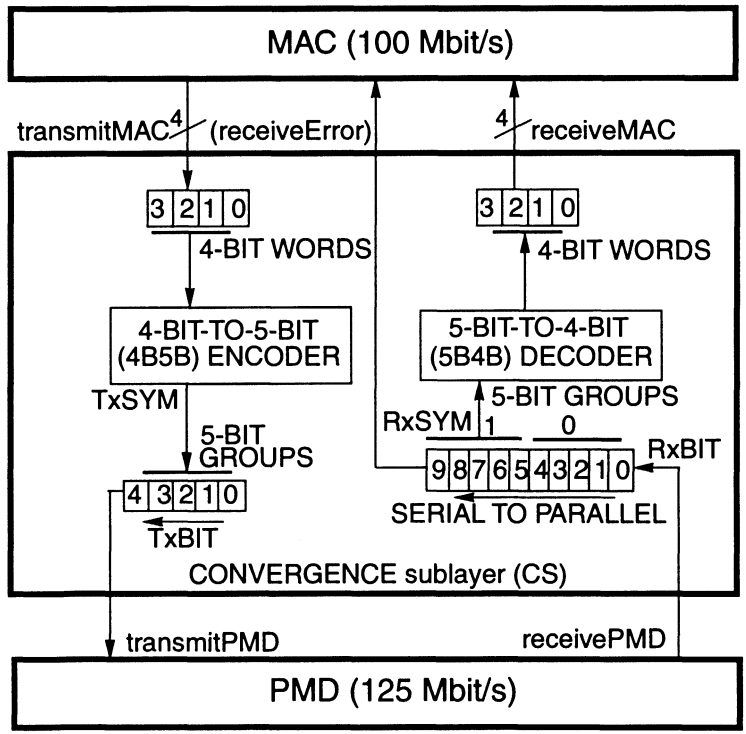


Figure 2. Data Flow in Convergence Sublayer

Received data at a 125 Mbit/s rate is sent from the PMD to the CS as the receivePMD signal. The CS formats input data to produce 5-bit symbol groups. Detection of the two-symbol sequence, J and K, marks the beginning of a packet and starts the synchronization of the input data stream. The 5-bit groups are then decoded by the 5B4B decoder and sent to the MAC as a stream of 4-bit words until the packet's end is detected by the reception of the end-of-packet delimiter characters, T and R.

The 4B5B encoding/decoding method, which is a subset of the standard FDDI 4B5B encoding method, employs 5-bits to encode/decode both 16 data (hex) characters and the signaling symbols required to indicate the start and end of the data packet. In addition to the 16 valid 4-bit-binary code groups shown in the Table 1, there are five special control signals used to indicate start of packet (J followed by K), end of packet (T followed by R), and idle (I). A number of other 5-bit combinations are designated as invalid and represent channel errors or repeater collision artifacts. Thus, the physical line idles until the start of a data packet is indicated by a J symbol followed by a K. Data symbols then follow with the end of data being indicated by a T symbol followed by an R. Idle symbols immediately follow. The job of the convergence sublayer is to extract the control characters or idles from the packet and then send a data-only packet to the MAC. Thus, the MAC never receives idle, JK, or TR symbols. When receiving, the CS reverses the process, encapsulating and encoding the data from the MAC for transmission by the PMD.

Convergence Sublayer Data Flow

The block diagram of the convergence sublayer is shown in Figure 2. The receive state machine generates receiveMAC data and receiveError for the MAC based on the receivePMD data input from the media. The transmit state machine accepts transmitMAC and transmitEnable from the MAC and generates the transmitPMD data to the physical layer. The collisionDetect function is generated by the transmit state machine, based on transmitEnable and the receive state machine's receiving signal.

The Carrier Sense function asserts the carrierSense signal when the convergence sublayer is either transmitting or receiving, based upon the two corresponding internal signals generated by the Transmit and Receive functions. The Link Monitor function generates linkTestFail based on the PMDs signalDetect. LinkTestFail is an internal signal unused by the MAC and can optionally be used by your network management entity.

Transmitted data, shown as the transmitMAC signal in Figure 3, indicates that MAC data is available and is registered in the convergence sublayer logic. Groups of 4 data bits in the transmit bit'stream are converted to 5-bit code groups by 4B5B encoding prior to transmission on the 125 Mbit/s PMD. Note that TxDATA, TxSYM, and TxBIT are all different views of the same data, but at different data rates.

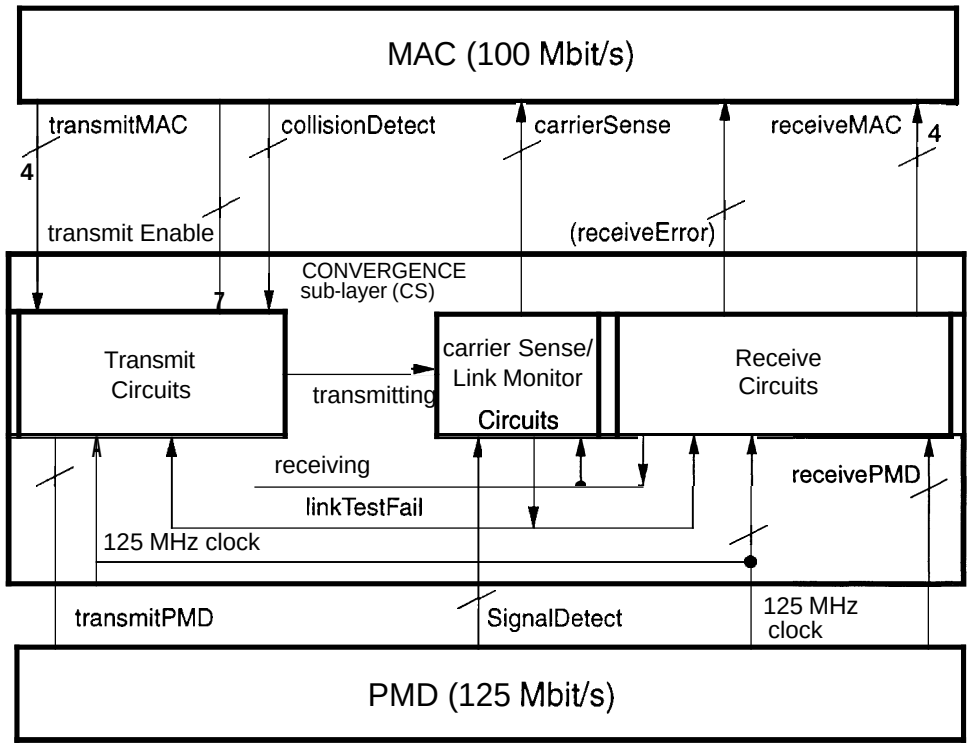


Figure 3. Convergence Sublayer Functional Block Diagram

Using FPGAs to Implement a 100Base-X Convergence Sublayer

As will be seen in the following sections, the 100Base-X convergence sublayer can be implemented as eight functional blocks, each of which forms the subject of a separate application discussion. These are listed under the three main headings: the transmit function, the receive function, and the carrier-sense and link-monitor circuits.

Convergence Sublayer Transmit Function

The design of the transmit function shows some common design techniques used in high-speed FPGA applications. The main function of this block is to provide the requested symbol data to the PMD at the 125 Mbit/s serial rate. This requires the 4-bit MAC data words to be 4B5B encoded and then shifted out using the 125 Mhz clock. In addition, the serial data stream needs to be framed using the leading /J/K/ symbol pair and trailing /T/R/ symbol pair. When data is not transmitting, it is replaced by the constant transmission of idle symbols. The transmit function is divided into two blocks as shown in Figure 4:

- The Transmit State Machine
- The Data Path

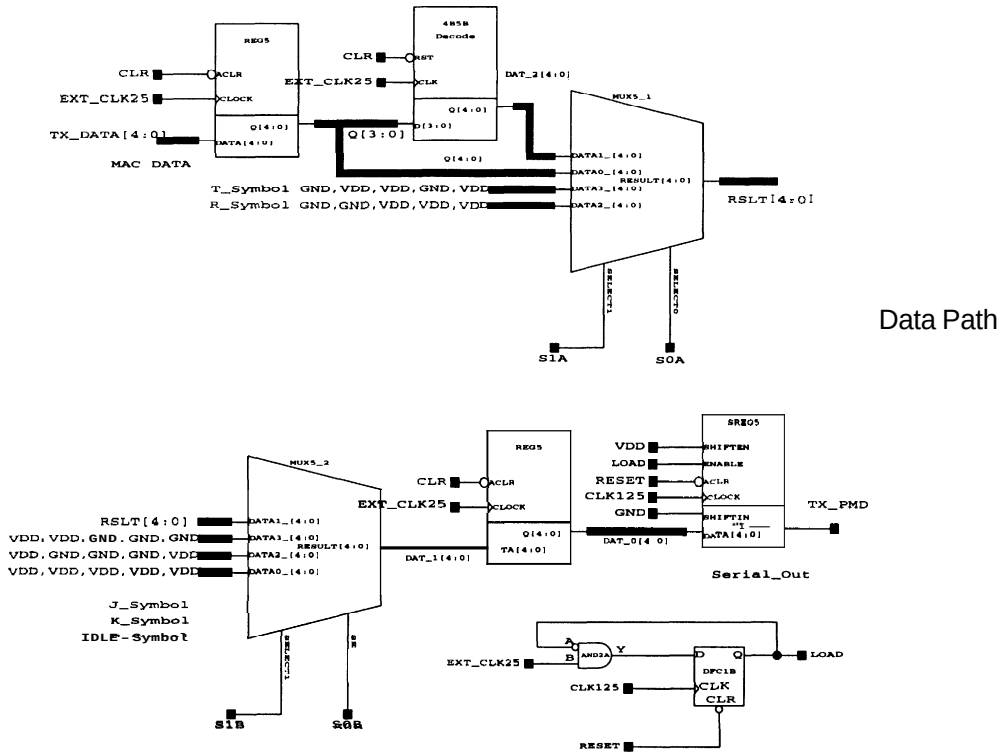


Figure 4. Convergence sublayer Transmit Functions

Data Path

The data path portion of the transmit block is shown in Figure 4. The main flow of data comes from the MAC at 25 MHz with a 4-bit-wide data word and a control signal that enables transmission. When MAC data is not being transmitted, the convergence sublayer sends continuous idle (I) symbols to the PMD. When the TransmitEnable signal becomes active, a /J/K/ symbol pair is transmitted to indicate the beginning of a data packet. MAC data symbols then follow and are encoded into 5-bit symbols using the 4B5B encoding scheme shown in Table 1. The end of MAC data is indicated by the TransmitEnable signal going inactive and a /T/R/ symbol pair is inserted at the end of the data packet. Finally, the CS logic returns to transmitting idle symbols.

Table 1. 4B5B Symbol Coding

Symbol	5-bit Code Group (in Convergence sublayer)	4-Bit-Binary Code Group (in MAC)	Interpretation/Function
0	11110	0000	Data character: 0H
1	01001	0001	Data character: 1H
2	10100	0010	Data character: 2H
3	10101	0011	Data character: 3H
4	01010	0100	Data character: 4H
5	01011	0101	Data character: 5H
6	01110	0110	Data character: 6H
7	01111	0111	Data character: 7H
8	10010	1000	Data character: 8H
9	10011	1001	Data character: 9H
A	10110	1010	Data character: AH
B	10111	1011	Data character: BH
C	11010	1100	Data character: CH

Table 1. 4B5B Symbol Coding (Continued)

Symbol	5-bit Code Group (in Convergence sublayer)	4-Bit-Binary Code Group (in MAC)	Interpretation/Function
D	11011	1101	Data character: DH
E	11100	1110	Data character: EH
F	11101	1111	Data character: FH
I	11111	--	Idle character transmitted between packets
J	11000	--	First control character in start-of-packet delimiter
K	10001	--	Second control character in start-of-packet delimiter
T	01101	--	First control character in end-of-packet delimiter
R	00111	--	Second control character in end-of-packet delimiter
V	00000	--	Invalid character
V	00001	--	Invalid character
V	00010	--	Invalid character
V	00011	--	Invalid character
V	00100	--	Invalid character
V	00101	--	Invalid character
V	00110	--	Invalid character
V	01000	--	Invalid character
V	01100	--	Invalid character
V	10000	--	Invalid character
V	11001	--	Invalid character

The implementation of the described functions involves selecting six different symbol sources for PMD data: the I (idle), J, K, T, and R symbols and the 4B5B encoded MAC data. In addition, a TestData input can be used to provide raw unencoded data to the PMD for use in diagnostics and testing. This selection is accomplished via the multiplexer in front of the output shift register. One multiplexer selects from the I, J and K symbols or from another multiplexer output. The other multiplexer selects from the T and R symbols and encoded the MAC data. Note the additional path around the encoder, which allows raw (unencoded) data to be provided to the PMD. This is used for system test and diagnostics and is the only way to inject known errors into the system, simulating collision remnants and exercising the boundary conditions of the standard.

The Actel logic implements multiplexers directly in a single logic module so, by inspection, the path through the multiplexer tree requires only two module delays and can easily meet the 25 Mhz performance requirement. The 4B5B encode block also requires only two logic levels and can be designed via schematics or via equations. The equations can be automatically compiled using Actel's ACTMap tool and then incorporated into the schematic.

4858 Encoder

Symbol encoding of the 4-bit data words transmitted from the MAC into the 5-bit coded groups required by the convergence sub-layer and the PHY layer employs a modified version of the coding used in FDDI-based systems. The differences from FDDI are that the symbols S, Q, and H are not used and that R is now used as part of the /T/R/ end-of-packet delimiter character group.

Table 1 lists all 32 5-bit data- and special-symbol codes that the PMD can send to the convergence sublayer. The 16 data characters--0 through F (hex)—are shown in Table 1, both as 5-bit code groups and as their 4-bit binary equivalents, as sent by the CS to the MAC. The idle character I and the control characters J, K, T, and R are shown in Table 1 in 5-bit form only, because they are not used in the MAC. The same applies to the remaining 11 possible 5-bit combinations that might be received on the media, all of which have no meaning to the decoder and hence are treated as invalid. For simplicity, each of the 11 invalid symbols is designated as V.

Encoding of 4-bit data words into 5-bit symbols can be accomplished in a few simple logic equations, as shown in the PALASM entry format shown in Figure 5.

The above equations translate each bit in sequence. Bits D0-D3 are the 4-bit data word input to the decoder, and B0-B4 define the 5-bit output symbols from the decoder. Thus, in the first equation, bit D0 is always the same as the bit B0, as can be seen by inspection of Table 1. The decoding equations for the remaining output bits (bits B4 through B1) are derived in a comparable fashion.

```
;Encoder for 4B to 5B
;Used in 100 Mbit Ethernet application
CHIP 4b5b generic
clk rst d3 d2 dl d0 q4 q3 q2 q1 q0

EQUATIONS

q4 := d3 + (/d2 * dl) + (/d2 * /d0)
q3 := d2 + (/d3 * /dl)
q2 := dl + (/d3 * /d2 * /d0)
q1 := (/d1 * /d0) + (d3 :+: d2) + (d2 * /dl)
q0 := d0

q4.clkf = clk
q3.clkf = clk
q2.clkf = clk
q1.clkf = clk
q0.clkf = clk

q4.rstf = /rst
q3.rstf = /rst
q2.rstf = /rst
q1.rstf = /rst
q0.rstf = /rst
```

Figure 5. PALASM2 Description for the 4B5B Encoder

ACTmap VHDL Synthesis and FPGA Optimization

These equations are then processed by ACTmap, a computer-aided design tool for working with the Actel families of FPGAs. It performs three basic functions:

- PALASM 2, VHDL to netlist translation
- Netlist-optimized mapping
- 1/0 insertion

ACTmap reads the PALASM 2, or VHDL source file and translates it into either an EDIF or an ADL (Actel Design Language) output file or Verilog netlist. The output file that it generates is optimized for a specific family of Actel FPGAs (ACT1, ACT 2, 1200XL, or ACT 3).

You can specify whether the design should be optimized for area or speed. The PALASM 2 description for the 4B5B encoder shown in Figure 5 was processed by ACTmap, and the following results where achieved:

- Area = 9 modules
- Estimated worst-case delay = 8.80 ns

These results easily meet the 25MHz requirements of the transmit function and show the speed and capacity capabilities of the ACT3 architecture. The schematic logic implementation of the 4B5B encoder (see Figure 6) shows the compact nature of the final implementation.

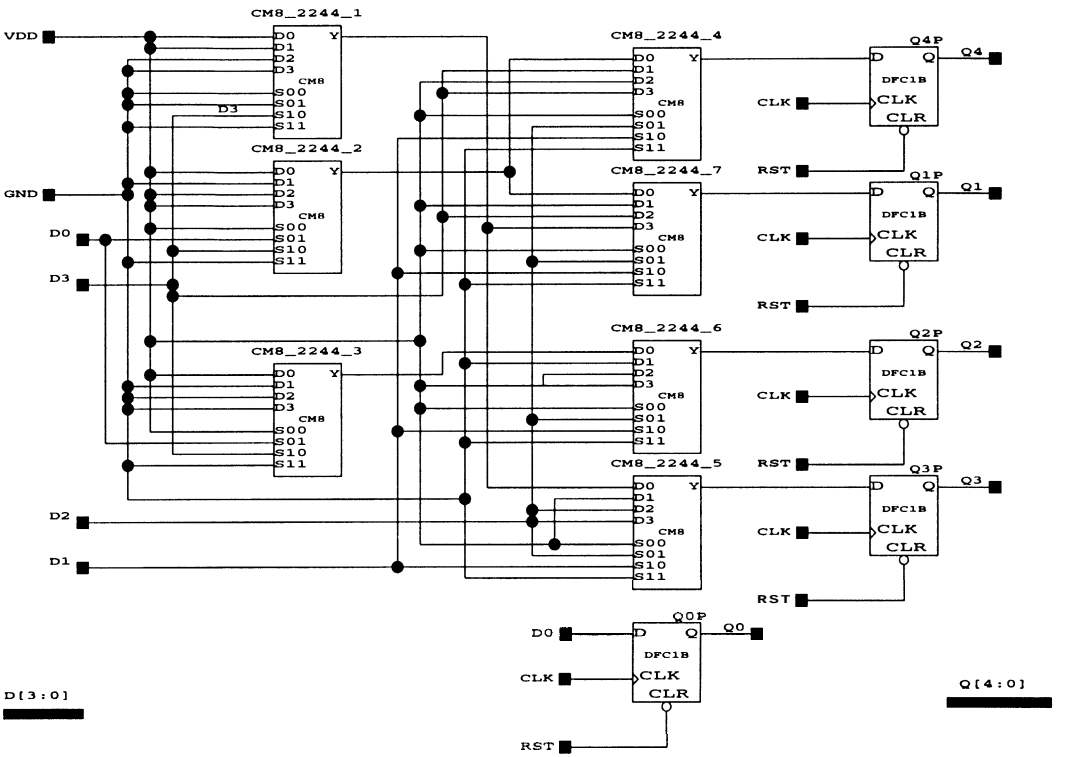


Figure 6. Transmit Path: 4B5B Encoder FPGA Logic

Transmit Operation

Transmit operational states are shown in block diagram form in Figure 7.

The actions shown in Figure 7 are assumed to be instantaneous, although, for simplicity, some time-sequenced events are contained in single states. Unconditional state transitions are unlabeled. Conditional state transitions occur when explicitly shown by the accompanying condition; a state is repeated until some transitional condition is detected. States are atomic in that conditions are evaluated only at the completion of the state's actions. Transitions shown without source states, notably linkTestFail, are evaluated at the completion of every state and take precedence over other transition conditions.

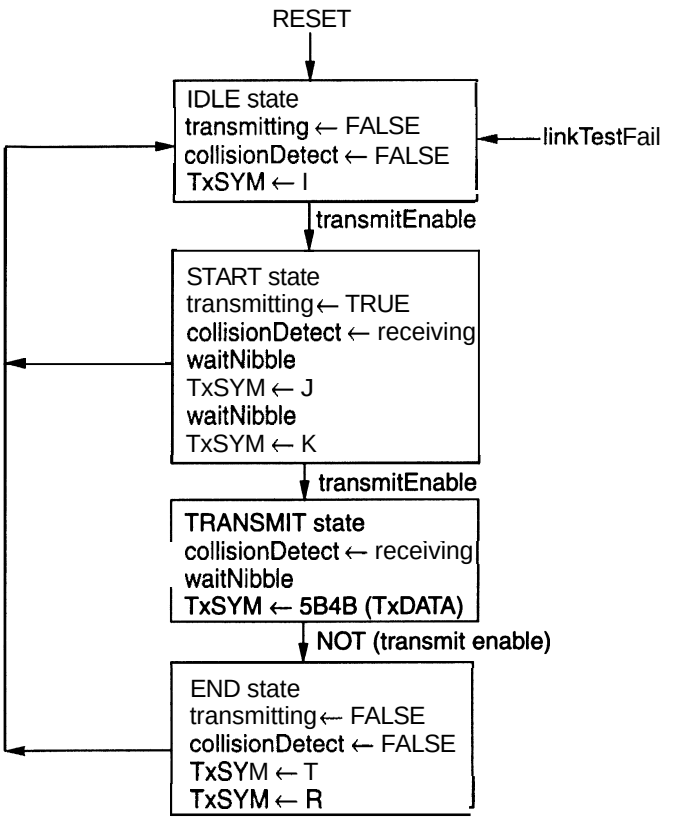


Figure 7. Convergence Sublayer Transmit Operation

Convergence Sublayer Transmit Operation

The transmit state block diagram begins with the IDLE state. The transmitting and collisionDetect signals are initialized as FALSE and the IDLE symbol is continuously supplied to the PMD. Once the MAC has data to transmit, it asserts transmitEnable and the START state is entered. The transmitting signal is asserted (set to TRUE) to indicate to the Carrier Sense function that data is being transmitted. In addition, collisionDetect is set to the level of the receiving signal. The receiving signal comes from the Receive function; if it is also asserted, a collision has occurred. The waitNibble function synchronizes the MAC data with the PMD clock. The first 8-bits of the MAC preamble are replaced with the /J/K/ symbol pair. If transmitEnable becomes FALSE, the machine makes a transition back to IDLE. If transmitEnable stays asserted, the next state becomes TRANSMIT. During TRANSMIT state, collisionDetect is still set to receiving. The MAC data (TxDATA) is encoded using the 5B4B function, and encoding continues until transmitEnable is disabled. Once transmitEnable is deasserted, the machine makes a transition to the END state. In the END state, transmitting and collisionDetect are both FALSE. The /T/R/ symbols are transmitted to indicate the end of data, and the machine moves to the IDLE state. The assertion of linkTestFail (by the Link Monitor function) causes an immediate transition to the IDLE state and takes precedence over any MAC request.

Transmit State Machine

The state-diagram implementation of transmit operation is shown in Figure 8. The state machine starts in the IDLE state and transmits the idle symbol (I) until transmitEnable (TE) is TRUE. As long as TE is TRUE, the machine proceeds through the J and K states, sending first the J symbol and then the K symbol to indicate the start of a data packet. The machine then transmits data until TE goes FALSE (i.e., transmit not enabled (/TE)), after which a T and an R are transmitted, indicating the end of the data packet. The state machine then returns to the IDLE state and waits for the next data packet. The test mode may be entered from the IDLE state by asserting Test mode (TM). In this mode, any 5-bit code symbol may be transmitted, thus allowing known error conditions to be injected onto the network.

The logic implementation of the transmit state machine is shown in Figure 9. Each state is encoded into the transmit state machine flip-flops to allow symbol selection in the transmit multiplexer. (See Note on page 26.) These transitions are controlled by the input logic for each flip-flop and depend only on the TE signal and the current state.

The resulting design employs only 11 logic modules and runs well in excess of the required 25 MHz speed.

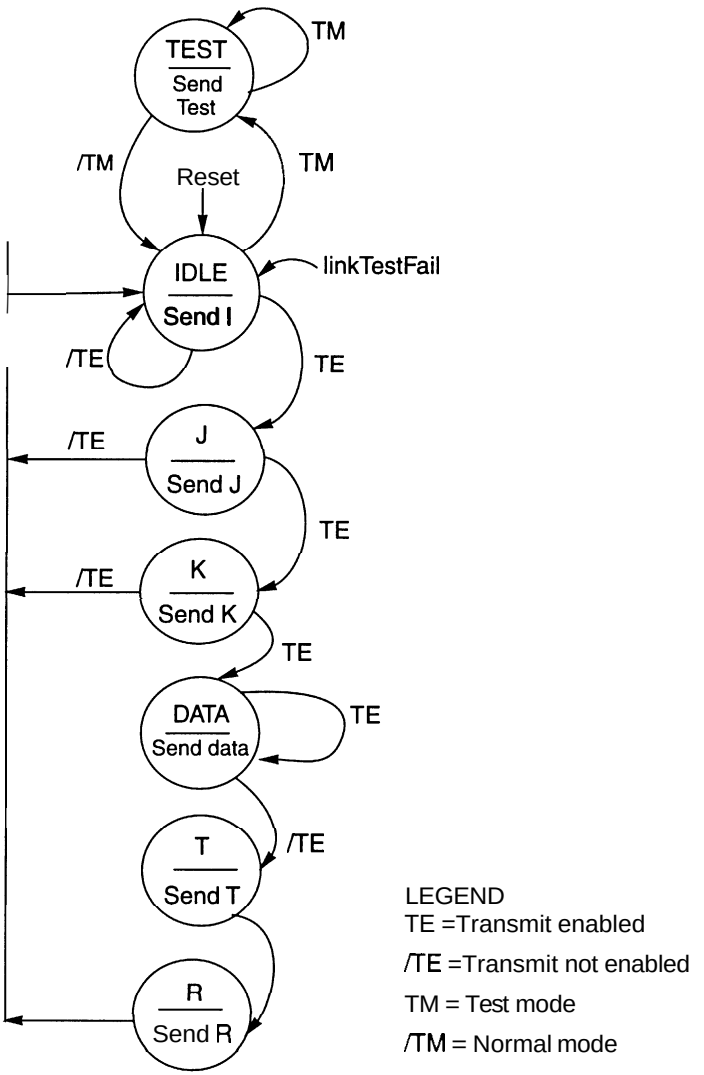


Figure 8. Transmit State Machine Diagram

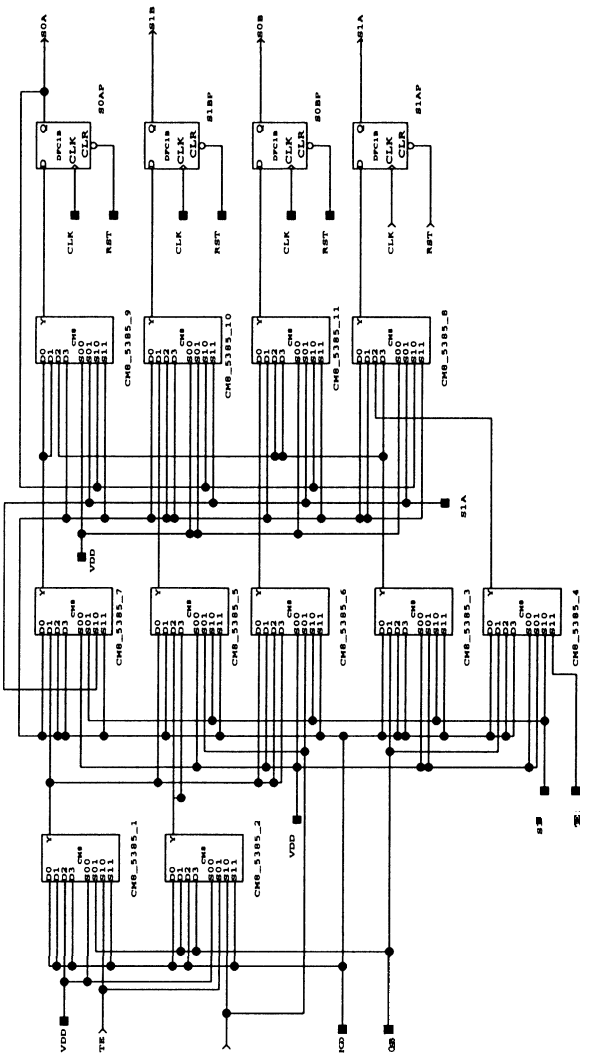


Figure 9. Transmit State Machine Schematic

Note This state-machine implementation differs from the commonly seen one-hot approach in that the states are encoded into four D flip-flops rather than a single flip-flop per state. Also, the state-machine encoding shown in this application is more efficient than the one-hot approach because the state flip-flops can drive the data-path multiplexer directly, eliminating the additional encoding logic that would be required to handle the one-hot state variables and to select desired multiplexer sources.

Convergence Sub-layer Receive Function

The receive functions of the convergence sublayer are shown in the block diagram in Figure 10. These functions are discussed in the following sections.

- Shift register, sync detect, and squelch
- Clock generation
- 5B4B symbol decoder
- Receive state machine

Receive Operation

The sequence of receive states is shown in the receive-operation diagram, Figure 11. The receive state machine tracks the received symbols to ensure that a complete packet has been received and indicates the current line state to the next layer of the protocol. The receive process (see Figure 11) involves two separate sets of states. The constituents in the first set—the IDLE, SCAN, CARRIER, and ALIGN states—are prealigned and operate on the raw input bits using RxBIT. The remaining states are aligned and operate on the input data stream as symbols (RxSYM). Output data, designated RxDATA, is sent directly to the MAC in these states.

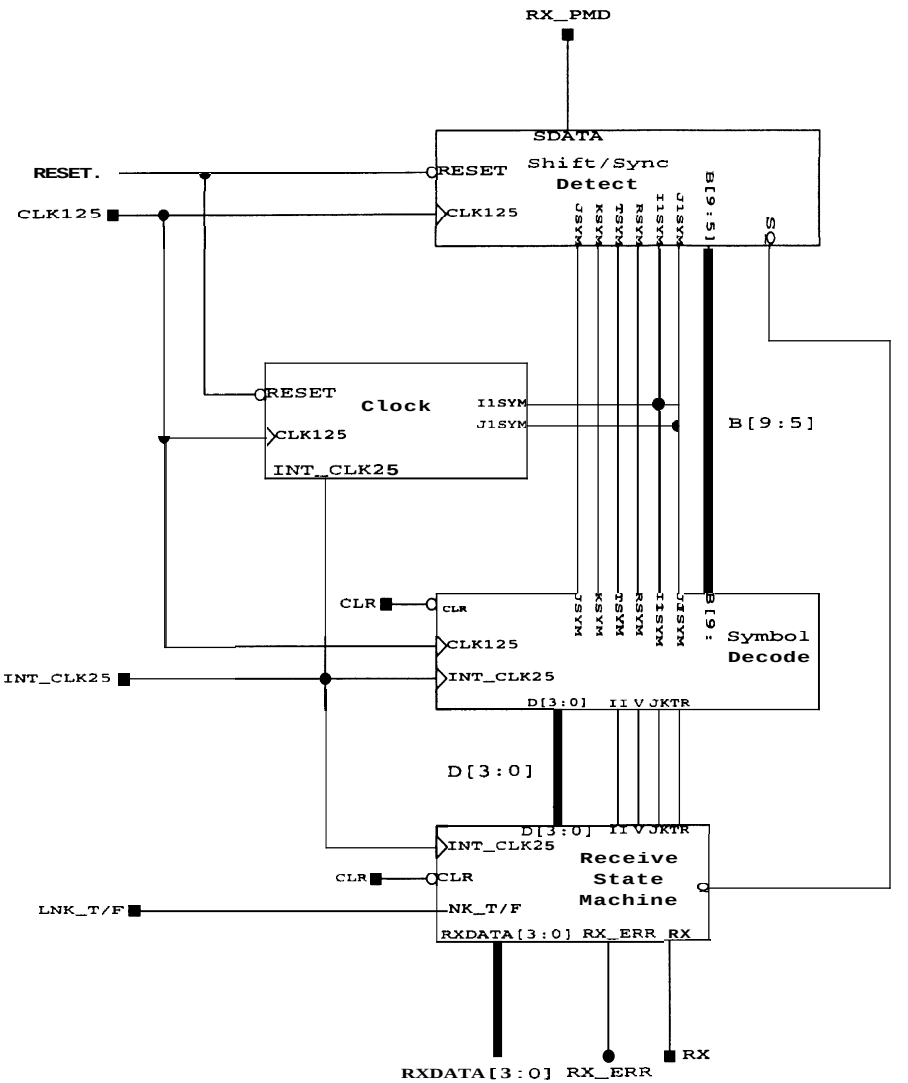


Figure 10. Convergence Sublayer Receive Functions

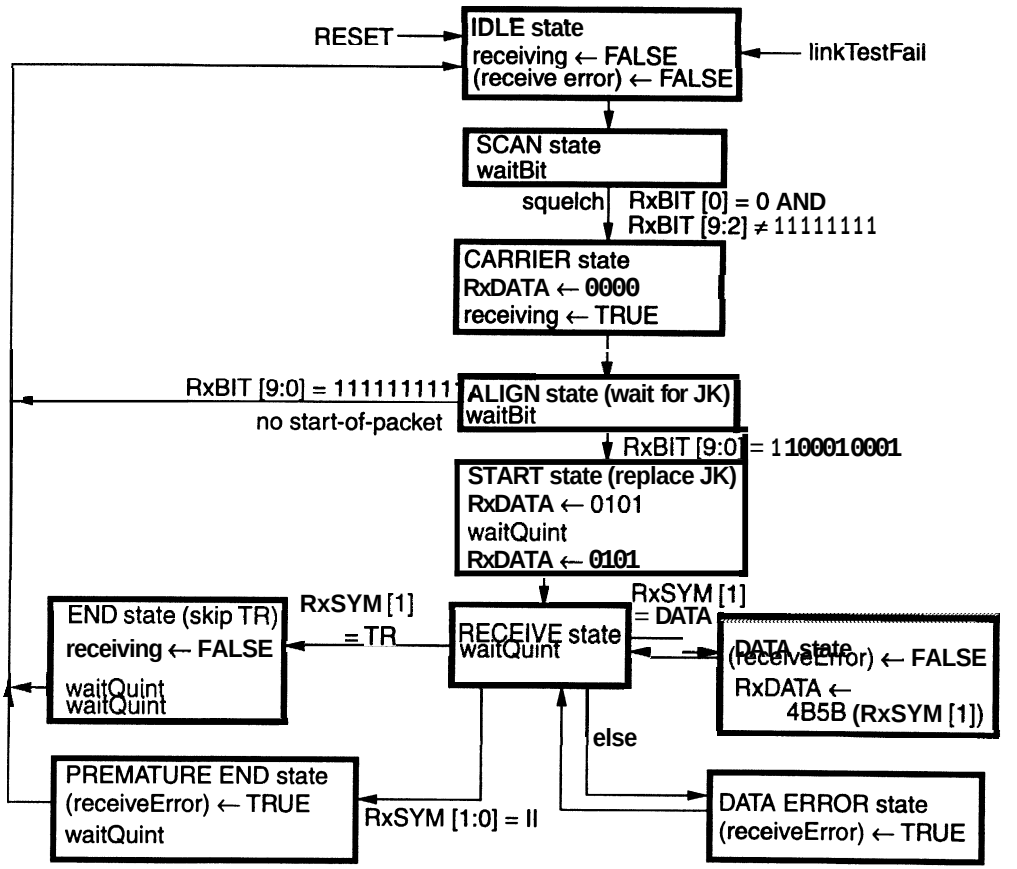


Figure 11. Convergence Sublayer Receive Operation

The RECEIVE state sequence begins with the IDLE state. The receiving signal and the optional receiveError signal are initialized to FALSE. The SCAN state is entered next and the waitBit function synchronizes the machine to the received data stream. At this point, the squelch function filters out noise events by not allowing a transition to the CARRIER state unless two nonconsecutive zeros are detected. Because carrierSense is used by the MAC for deferral purposes, it must be asserted on the detection of any received signal (i.e., received energy, or non-IDLE input) whether or not it's an actual packet. Since carrierSense is also used to detect collisions, it's important to avoid triggering on noise, specifically a single-bit event. If CARRIER is entered, RxDATA is initialized to all zeros (0000) and receiving is set to true.

The system enters the ALIGN state next. In ALIGN, the start of packet symbols /J/K/ is searched for. If at least two idle symbols (1111111111) are found instead, no start of packet has been detected and the machine moves to the IDLE state. If the /J/K/ symbols are successfully found, the START state is entered. In START, the MAC preamble data (55) is substituted for the received /J/K/ symbols. The waitQuint function assures that MAC data is not overwritten. The RECEIVE state is entered next. Usually, in the RECEIVE state, valid data is received and a transition to the DATA state is made. In the DATA state, receiveError is deasserted and 4B5B decoded data is sent to the MAC.

From the DATA state, the machine returns to the RECEIVE state. If, during the RECEIVE state, two idle symbols are received, the PREMATURE END state is entered, receiveError is asserted, and IDLE is reentered. If, in the RECEIVE state, invalid data is received, the DATA ERROR state is entered, receiveError is asserted, and RECEIVE is reentered. Invalid data is not transmitted to the MAC. If, in the RECEIVE state, a /T/R/ symbol pair is detected, the END state is entered, receiving is deasserted, and IDLE state is reentered.

Shift Register, Sync Detect, and Squelch

The shift register, sync detect, and squelch circuits (Figure 12) are responsible for shifting serial data at the 125 Mhz line rate and detecting clock synchronization symbols. Once a sync symbol is detected, the clock generation state machine adjusts the 25 Mhz symbol clock by stretching it the required number of 125 Mhz clocks to align it with an input symbol. Control symbols in the input data stream can then be captured correctly by the 25 Mhz clock and decoded by the 4B5B decode block.

Serial data is clocked into the shift register and sync detect block by using the 125 Mhz clock. Sync symbols are detected as the data shifts. As shown in Figure 12, only a single logic level is required to detect each of the five important sync signals (J,K,T,R and I). The code groups corresponding to each of these symbols are shown in Table 1 on page 13.

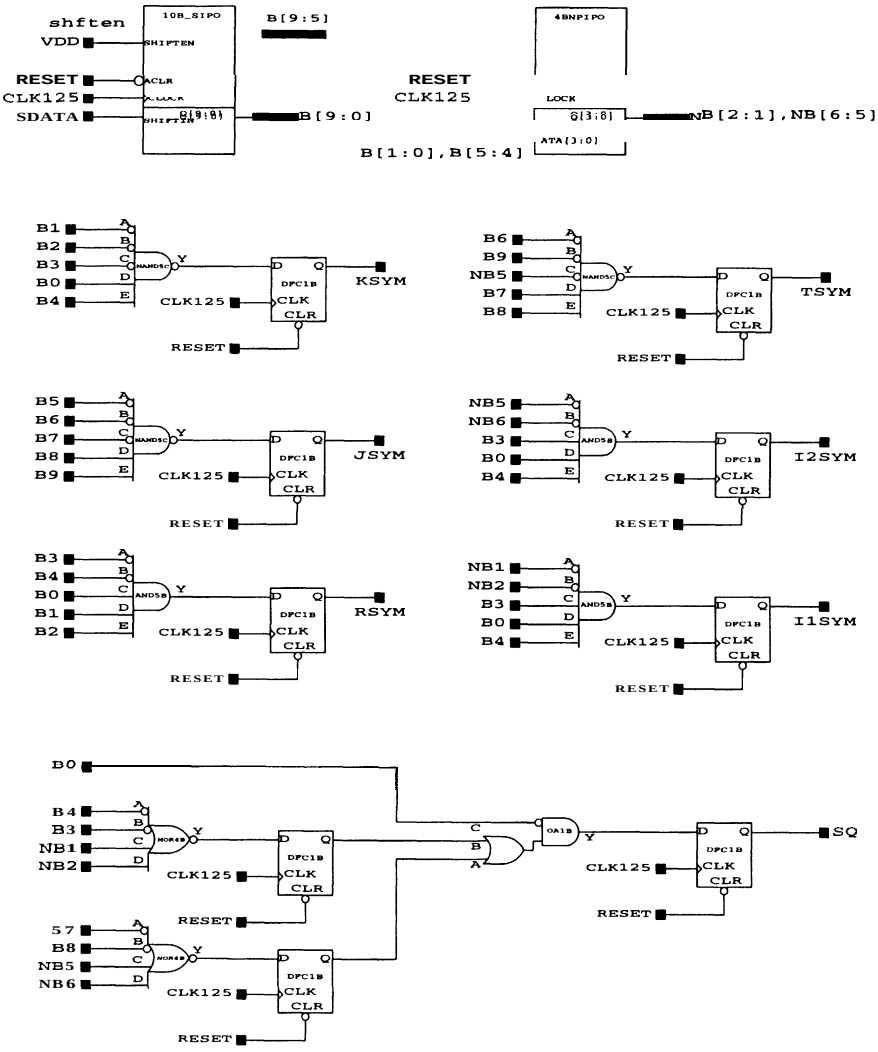


Figure 12. Shift Register, Sync Detect, and Squelch Schematic Diagram

Note The shift register generates both the true and complemented versions of bits B3, B4, B8, and B9. This is required to implement single-level decode for the I symbol because the ACT3 logic module implements five-input AND/NAND gates with at least two inverted inputs. The technique of providing additional registers with inverted outputs is common when implementing logic functions using fine-grained antifuse FPGAs. The additional registers cost little because of the fine-grained logic module, and they can be used where needed to provide additional logic signals. The abundant routing resources available with antifuse FPGAs also supply the additional routing required to create these additional logic signals.

The squelch function filters out noise events from the received data stream. Zeros are ignored unless there are two noncontiguous zeros within the first 10 bits. At first glance, it would appear that the logic to detect two noncontiguous zeros in a 10-bit word should be quite extensive. However, once it is observed that this function is used only on a serial data stream, several simplifying logic reductions can be made. First, check the least significant bit (BO) for a zero, and then check that at least one of the higher-order bits (only B2 through B9, since B1 is contiguous) is also zero. **Any** other combination is simply a shift from BO. However, the resulting logic equation for a squelch state (S),

S:= /B0(/B2+/B3+/B4+/B5+/B6+/B7+/B8+/B9)*

is too large to implement in a single FPGA logic level.

To simplify this approach, note that because data is being serially shifted in, higher-order terms can be precomputed and then combined with the critical B0 signal using a single logic level. The logic that results can be expressed by the following three equations:

S1:= /(/B1+/B2+/B3+/B4)

S2:= /(/B5+/B6+/B7+/B8)

S:= /B0(/S1+/S2)*

These three equations are the ones actually implemented in FPGA form, (see Figure 12.)

Note As shown in Figure 12, bits B1–B4 and bits B5–B8 are used with registers to develop the two intermediate terms S1 and S2. These two are then ORed with bit B1 to develop the final squelch function (S). This form of pipelined operation works well in serial data applications and will almost always result in faster and more area-efficient FPGA designs.

Also, notice that the extra inversions on the S1 and S2 terms (Figure 12) are used because NOR functions with inverted inputs map more easily into in a single ACT3 logic module. Synthesis software like Actel's ACTmap Program figures this out automatically, allowing the designer to focus on architectural and functional issues instead. Thus, what initially looks like a difficult decoding problem can be significantly simplified to only three logic modules that operate easily at the serial data rate.

Clock Generation

The clock generation state diagram and the clock generation schematic diagram are shown in Figures 13 and 14, respectively. The clock generation logic divides the 125 Mhz serial clock by 5 to generate the 25 Mhz symbol clock, Clock25. The Clock25 signal (Figure 13) is stretched when a sync symbol is detected, to align it with the 5-bit symbols. This is accomplished by a transition to the Q0 state when the JK signal (start of packet) is active. Entry into Q0 synchronizes the 25 Mhz clock (Clock25, the output from states Q2 and Q3) and the load signal. The load signal is active every 5 clocks after synchronization, which captures the 5-bit symbol from the aligned data stream. The symbol can then be safely captured by the 25 Mhz clock, Clock25 in the aligned symbol register (ASR). The schematic implementation for this process is shown in Figure 14.

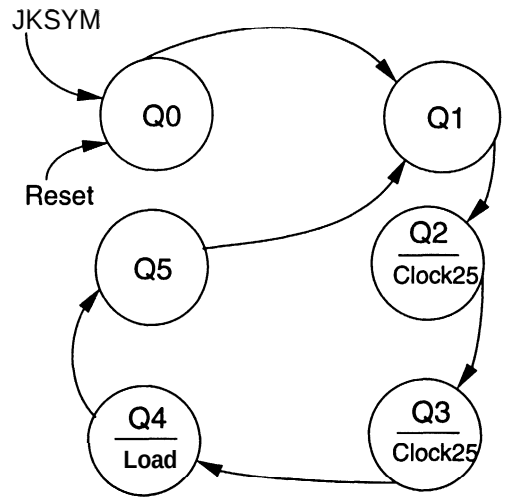


Figure 13. Clock Generation State Diagram

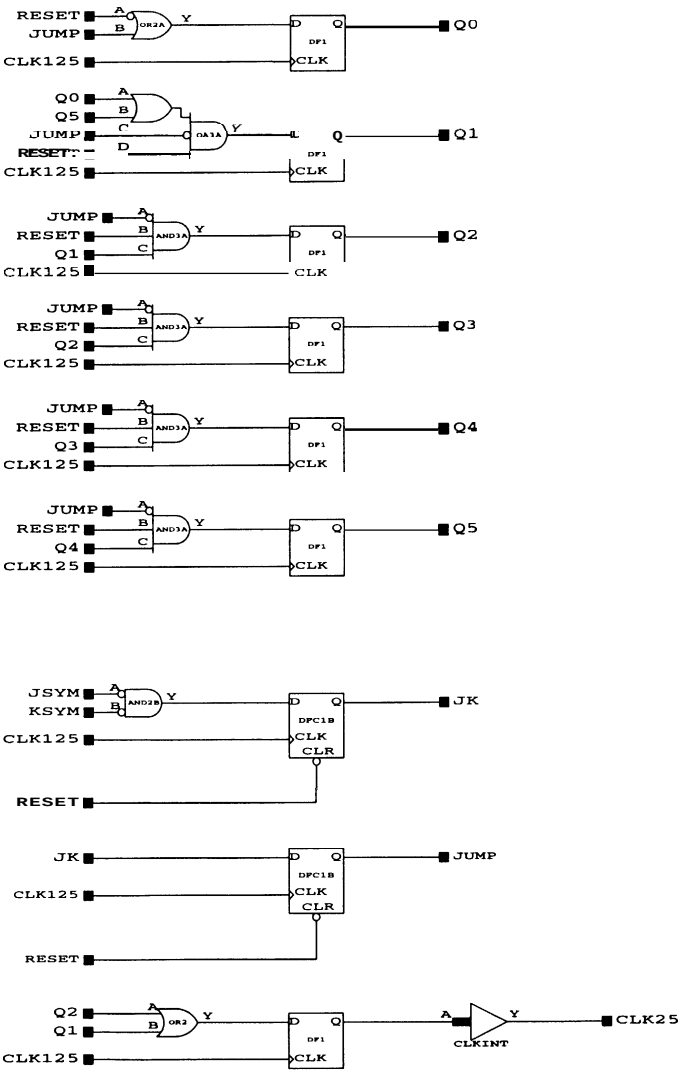


Figure 14. Clock Generation State Machine Schematic

Note Each state in the machine usesa single register. This one-hot (i.e., one register at a time) type of state machine design uses the register-intensivenature of fine-grained, antifuse-based FPGAs to reduce the logic complexity required to determine next-state transitions. In traditional encoded designs every state bit is needed to determine which state the machine is in. This can make for large transition terms in complex state machines. FPGAs, on the other hand, can use the additional register available to reduce the logic complexity, because only a single register output is required to determine the state of the machine. Thus, the FPGA's narrow, high-speed logic module can be used to generate the transition terms efficiently. In fact, on closer examination, the implementation of the state machine maps very closely to the state diagram. Transitions from one state to another result in a connection from the starting-states register to the entered-states register. Logic complexity can easily be estimated directly from the state diagram. Because only a single logic module is required to implement even the most complex transition, the entire machine runs easily at the 125 Mhz clock rate.

5848 Symbol Decoder

Once a symbol has been aligned, the data must be extracted by converting the 5-bit input from the PHY into a 4-bit data word that is sent to the MAC. The logic diagram for this decoder is shown in Figure 15. Symbol conversion is done in accordance with the 4B5B decode table, Table 1. Implementation of the decoder in the ACT3 family is automatically generated from the logic equations developed from the encoding table by using the ACTmap tool. As shown in Figure 15, the full decode requires only 24 modules and only two levels of logic, easily meeting the speed required for the 25 Mhz clock.

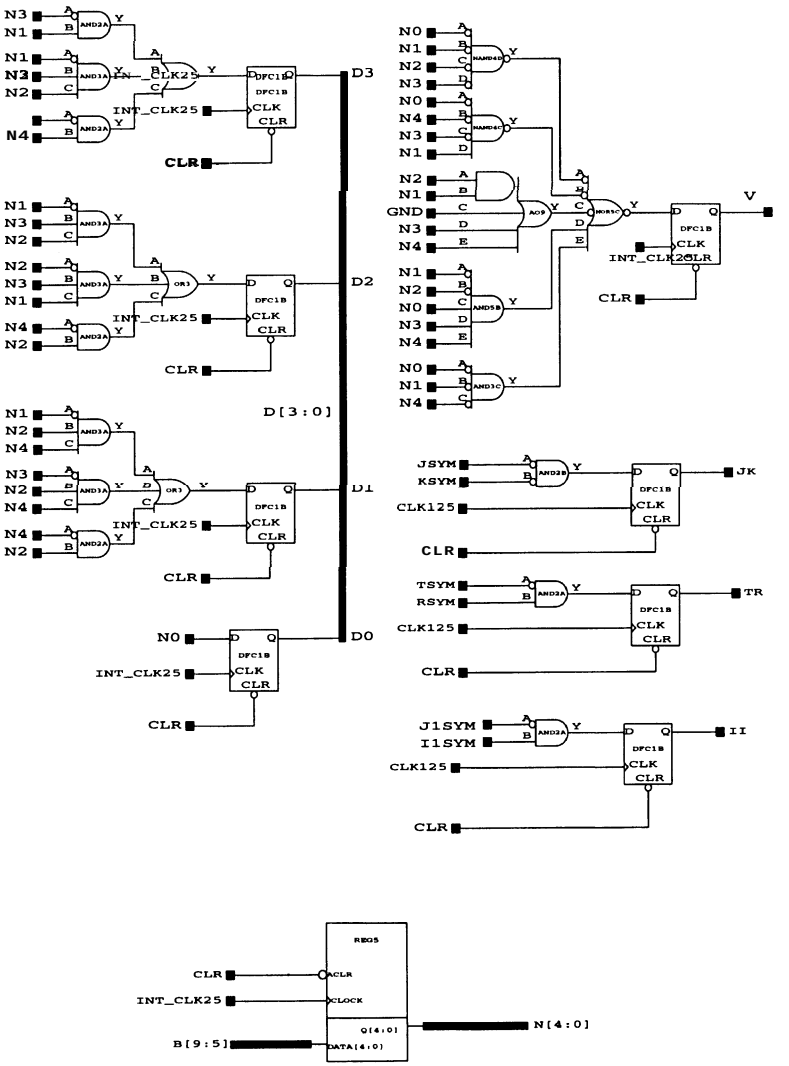


Figure 15. 5B4B Decoder Schematic Diagram

Receive State Machine Diagram

The RECEIVE state diagram is shown in Figure 16. The machine begins in the START state and waits for the reception of a JK symbol pair. The RECEIVE state is entered upon the reception of this pair and exited only under one or more of the following conditions:

- Reception of a TR symbol pair (end of data packet)
- Reception of an idle (I) symbol (premature packet end)
- Reception of an invalid symbol (error condition)

Note that if an invalid symbol is received, the ERROR state is entered to capture the event. This state is cleared only by resetting the state machine.

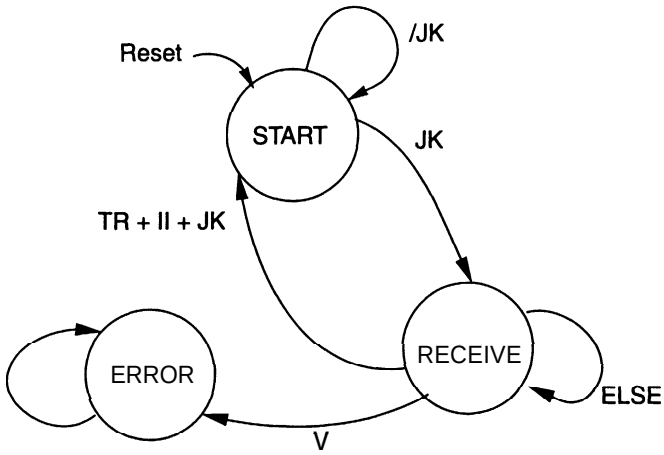


Figure 16. Receive State Machine Diagram

Receive State Machine Logic

As shown in Figure 17, the schematic implementation of the RECEIVE state machine requires only 4 logic modules and two levels of logic. It easily meets the 25 Mhz clock rate required for this portion of the design.

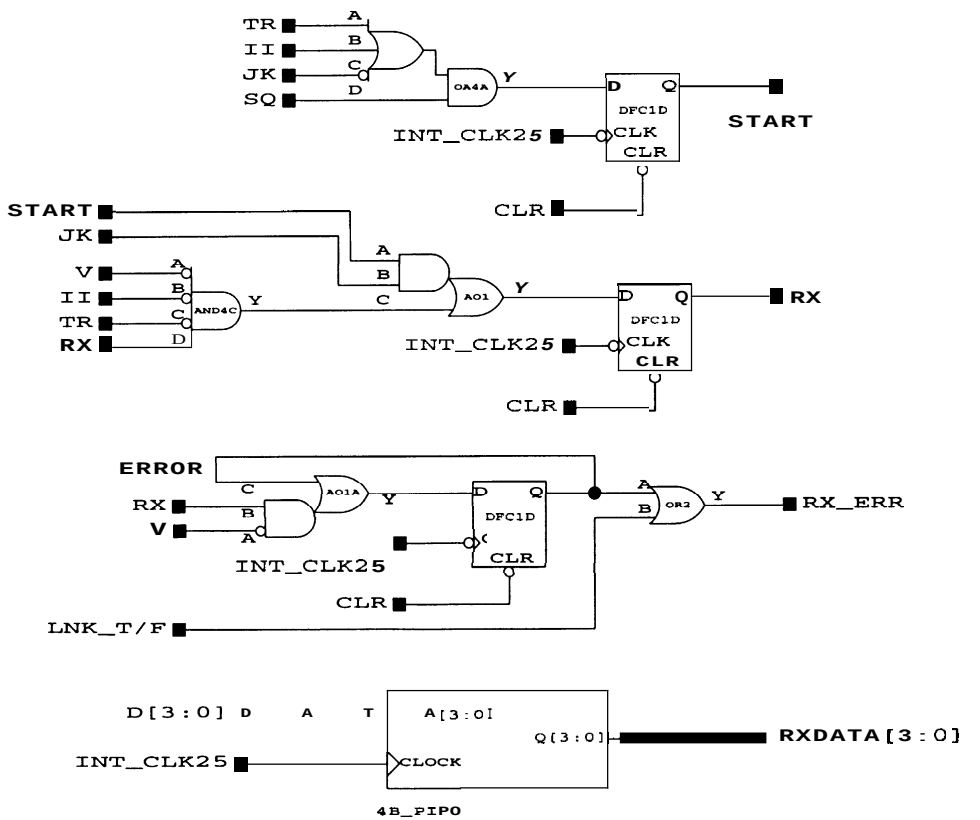


Figure 17. Receive State Machine Schematic Diagram

Carrier Sense and Link Monitor Circuits

The carrier sense and link monitor circuits combine outputs from the transmit, receive, and PMD blocks to develop the receiveError, carrierSense, and collisionDetect signals. The logic for implementing this process is shown in Figure 18.

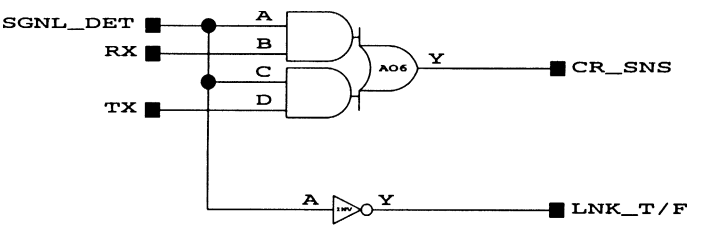


Figure 18. Carrier Sense and Link Monitor FPGA Logic Schematic

Conclusion

This application note has described the complete design of a 100Base-X convergence sublayer. Each building block has been fully tested and documented and is available on disk to those contemplating similar or related applications.

Designing High-Speed ATM Switch Fabrics by Using Actel FPGAs

The recent upsurge of interest in Asynchronous Transfer Mode (ATM) is based on the recognition that it represents a new level of both speed and simplification in telecommunication networks. The most significant characteristic of ATM is that it requires minimum cell processing in network nodes and in links such as repeaters, bridges, and routers. This means that ATM allows systems to operate at rates much higher than current packet-switching systems allow. This improved performance is due to higher media quality and to ATM operation in a connection-oriented mode that guarantees minimum packet loss. This low packet loss is the result of not granting entrance to the network until completion of a setup phase that allocates all necessary network resources.

To reduce the size of the internal buffers in switching nodes, and thus to reduce the queuing delays in these buffers, the information field length in ATM packets is kept relatively small. As a result, as packet size goes down, the speed requirement for each switching node on the network goes up. In general, to keep packet loss to a minimum, the throughput of ATM switching nodes must be in the 1-gigabit-per-second range.

This application note describes how to design typical high-speed switch fabrics that route ATM packets on broadband networks. *Switch fabric* is a term used to denote a large group of basic switching building blocks connected in a specific topology. The design, analysis, and implementation of these building blocks will be described.

ATM Switching Applications

One of the main tasks of an ATM switching node is to transport ATM cells at high speed from its input ports to its destination output ports. This task is performed by the switch fabric. The switch fabric establishes a connection between an arbitrary pair of input and output ports. Switch fabrics usually consist of identical basic units called *switching elements*. The switching elements are interconnected in a specific topology to create the switch fabric.

The Actel ACT 3 family of FPGAs, with their high-speed multiplexer-based architectures, are, as will be seen in this application note, an excellent fit for applications, such as ATM switching, that stress the heavy use of multiplexing. This application note will describe in detail the designs of two typical high-speed ATM switches:

- A pipelined 16:16 switch fabric
- A 16:16 multipath interconnect (MIN) switch fabric

Piplined 16:16 FPGA Switch Fabric

One of the simpler FPGA approaches to ATM switch fabric design is shown by the straightforward 16:16 multiplexing scheme in Figure 19. This switch fabric design is known as a single-path network, because the same path is always used from any given input to a given output. In this example, the switch fabric has 16 input ports and 16 output ports. To achieve connectability, it employs 16 16:1 multiplexers called FFMX16's (Figure 20). Each of the 16 FFMX16s uses five Actel DFM6A basic 4:1 multiplexed flip-flops connected in two pipelined stages.

Each DFM6A (Figure 21) is a 4:1 multiplexer driving a flip-flop and occupies a single Actel ACT 3 sequential logic module. This multiplexed flip-flop introduces only one level of logic delay in the design. In this implementation, once each of the two stages of the 16:1 multiplexer is full, the pipeline outputs data on every subsequent clock cycle. Thus, these 16:1 multiplexers are effectively implemented in one logic level, providing improved throughput.

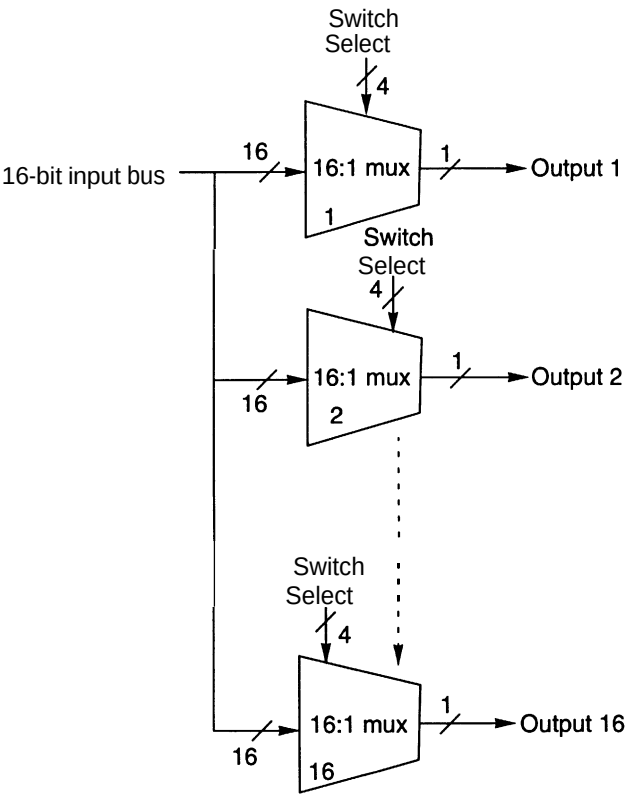


Figure 19. 16:16 Basic Multiplexer Switch Fabric

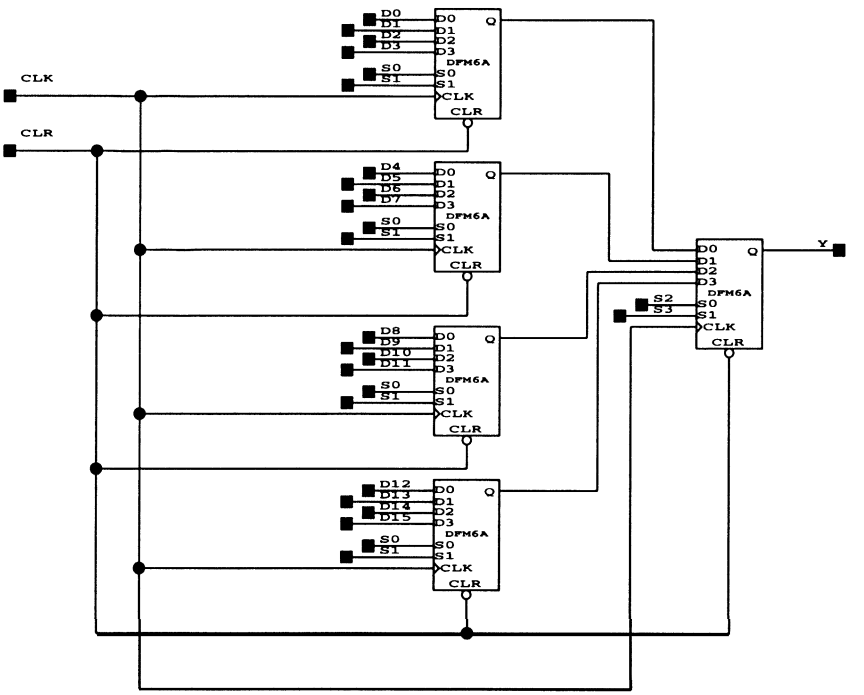


Figure 20. Two-Stage Pipelined 16:1 Multiplexer (FFMX16)

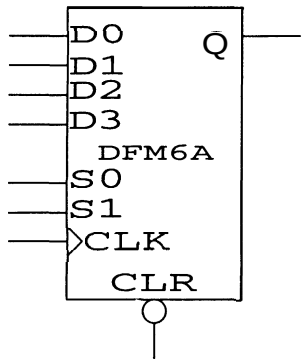


Figure 21. Multiplexed Flip-Flops DFM6A

Switch Fabric Driving Circuit

The driving circuit for each of the FFMX16s in the 16:16 switch fabric is shown in Figure 22. As shown in the figure, selecting data for the output of each FFMX16 requires 64 signals. This number is based on the need for four switch-select inputs (S0, S1, S2, and S3) for each of the FFMX16s. Switch-select signals S0 and S1 operate with the first stage of each multiplexer, and select signals S2 and S3 operate with the second stages. A drive signal is received as a serial bit stream (SWSEL) that is converted to parallel form by a 64-bit serial-to-parallel shift register (SIPO64). Input data to the switch is received on the lines D[15:0]. Notice that the FFMX16 shown in Figure 22 represents 16 FFMX16s, so the inputs are 16 bits wide.

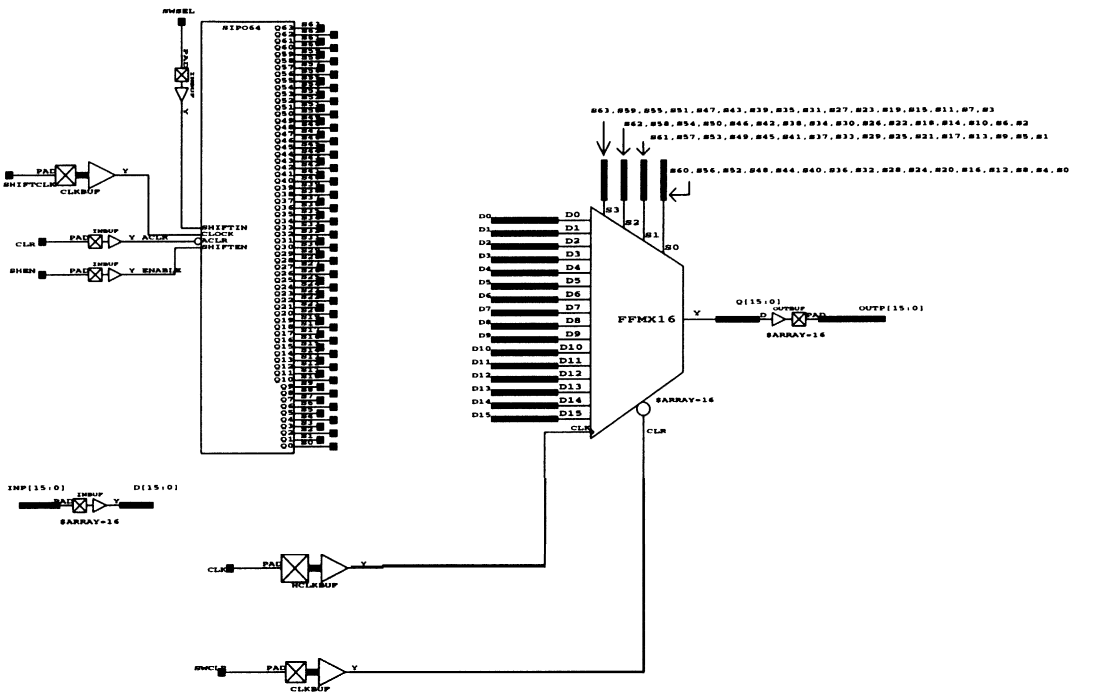


Figure 22. Top Level View of the 16:16 Switch Fabric Design

It takes two clock cycles to fill the FFMX16 pipeline, after which data is present on the OUTP[15:0] bus at each clock cycle. The 64-bit data selection word S[63:0] from the shift register is divided into four groups of 16 bits each, which are used to select the appropriate routing through the switch fabric. Note that there are separate clock lines for both the shift register and for the switch fabric so that data can be clocked to the output bus at a rate different from that of the shifted control bits.

Using ACTgen Macro Builder

The 64-bit serial-to-parallel shift register (SIPO64) is generated by the Actel macro generator, ACTgen included with Desinger Series software packages. With ACTgen's graphical user interface, you can build structured macros (counters, adders, etc.) by simply clicking on a few menu choices. The ACTgen Macro Builder then creates functions that effectively use the Actel architecture. Each macro is developed with the goal of limiting module count, maximizing performance, and restricting loading to acceptable levels.

In this design, the SIP064 is generated by simply choosing the desired parameters from ACTgen's graphical user interface (64-bits, serial-to-parallel, active-low clear, active-high shift enable, and positive-edge triggered clock). The created shift register is then instantiated in the design with no need for simulation. The ACTgen macros are already tested to guarantee correct functionality.

Note In the N:N multiplexed structure shown here, any given input may be broadcast to all outputs simultaneously. Also, an advantage of this approach is that after only two clock cycles, the pipeline is full and ready to output data. A disadvantage of this scheme is that it allows only one possible path, and no alternative paths, between each input and output. To implement a multiple-path capability, multiples of this switch fabric can be cascaded together. However a better solution is the MIN switch fabric.

16:16 Multipath Interconnect (Min) Switch Fabric

The primary advantage of a multipath interconnect network (MIN) is that it permits the creation of alternative paths between a given input and a given output in order to avoid possible packet collisions. One implementation of a MIN switch is the two-section Banyan network shown in Figure 23. This network consists of four stages that drive a second group of four. The second group is made up of the first four stages with a reversed topology—it is the mirror image of the first. Adding this second half produces a complete MIN switch fabric in a minimum number of stages.

The first four stages (see Figure 23) enable any output to be reached from any input via one specific path. This is the standard Banyan configuration. The second four stages use a reversed Banyan topology. Together, the two sections provide the multiplicity of paths required for a MIN switch fabric. That is, in an N: N MIN switch fabric, N internal paths are available to reach any output from an arbitrary input.

Each basic switching element used in this switch fabric is a 2:2 switch. (See Figure 24.) Depending on the value of switch line SW, the data will be either passed or crossed between the input and output lines. Figure 25 shows the implementation of the basic switching element using Actel DFME1A ACT 3 multiplexed flip-flops. As shown in the figure, two multiplexed flip-flops are required to implement the switching element. The truth table for the basic switching element shown is given in Table 2.

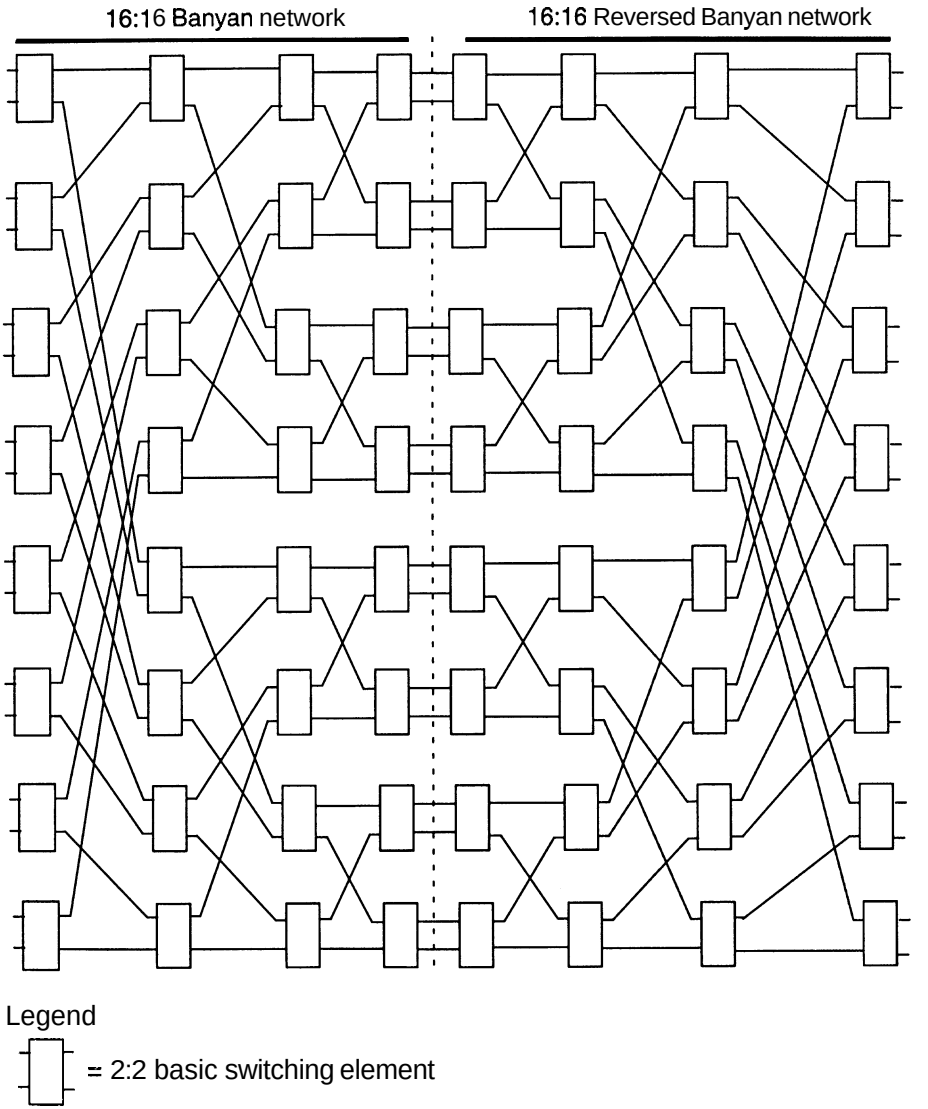


Figure 23. 16 X 16 Mulltipath Interconnect Network (MIN) Switch Fabric

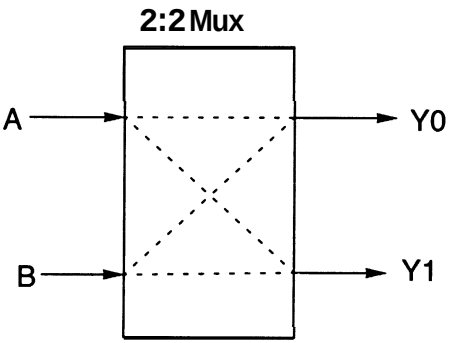


Figure 24. Possible Signal Paths in a 2:2 Basic Switching Element

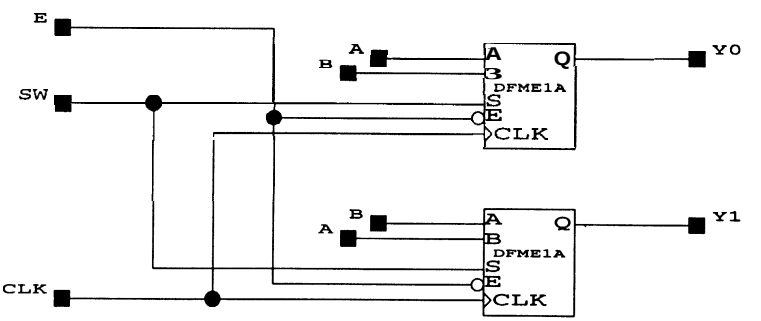


Figure 25. **2:2** *Basic Switching Element SWCELL*

Clock CLK	Enable E	Switch SW	YO	Y1
X	1	X	As in previous state	As in previous state
↑	0	0	A	B
↑	0	1	B	A

Notes: ↑ =Triggered on positive edge of clock
X = Don't care

Table 2. Truth Table for 2:2 Basic Switching Element SWCELL

Assuming an N: N MIN switch fabric, the number of stages in the network is $2\log_2 N$. If N = 16 (as in the present case), the MIN switch fabric is implemented in eight stages. Thus, a 16:16 MIN can be constructed of eight stages, with each stage consisting of eight 2:2 basic switching elements.

Switch Fabric Driving Circuit

The driving circuit for the MIN network is similar to the one used for the multiplexer-based switch fabric described in the previous section. As shown in Figure 26, the switching elements (SWCELL) are connected to each other to implement the topology shown in Figure 23. The SW lines of the switching elements are driven by the outputs (S[63:0]) of a 64-bit shift register. The S[63:0] signals are received as a serial bit stream (SWSEL) and are converted to parallel form by a 64-bit serial-to-parallel shift register (SIPO64). Input data to the switch is received on the lines IN[15:0] and is clocked to the outputs once the SWCELLs are enabled.

It takes eight clock cycles to fill the switch fabric pipeline, after which data is present on the OUTP[15:0] bus at each clock cycle. The 64 bits of data selection word (S63:S0) from the shift register are used to select the appropriate routing through the switch fabric. Note that there are separate clock lines for the shift register and for the switch fabric so that data can be clocked to the output bus at a rate different from that of the shifted control bits S[63:0].

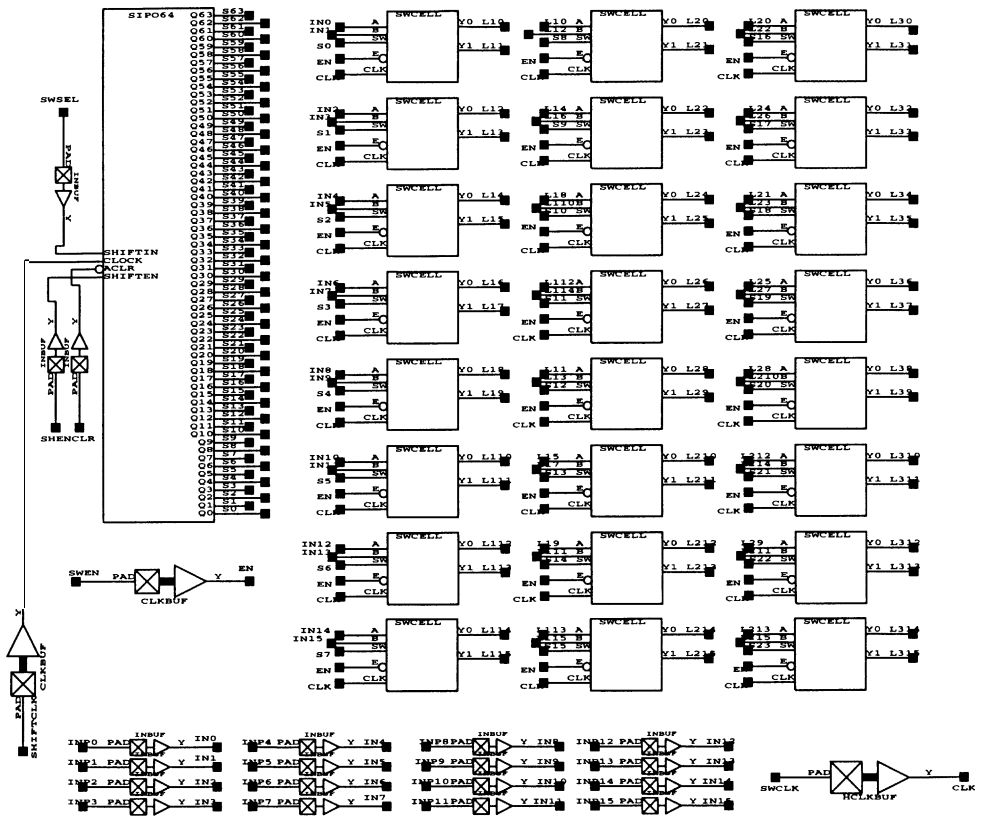
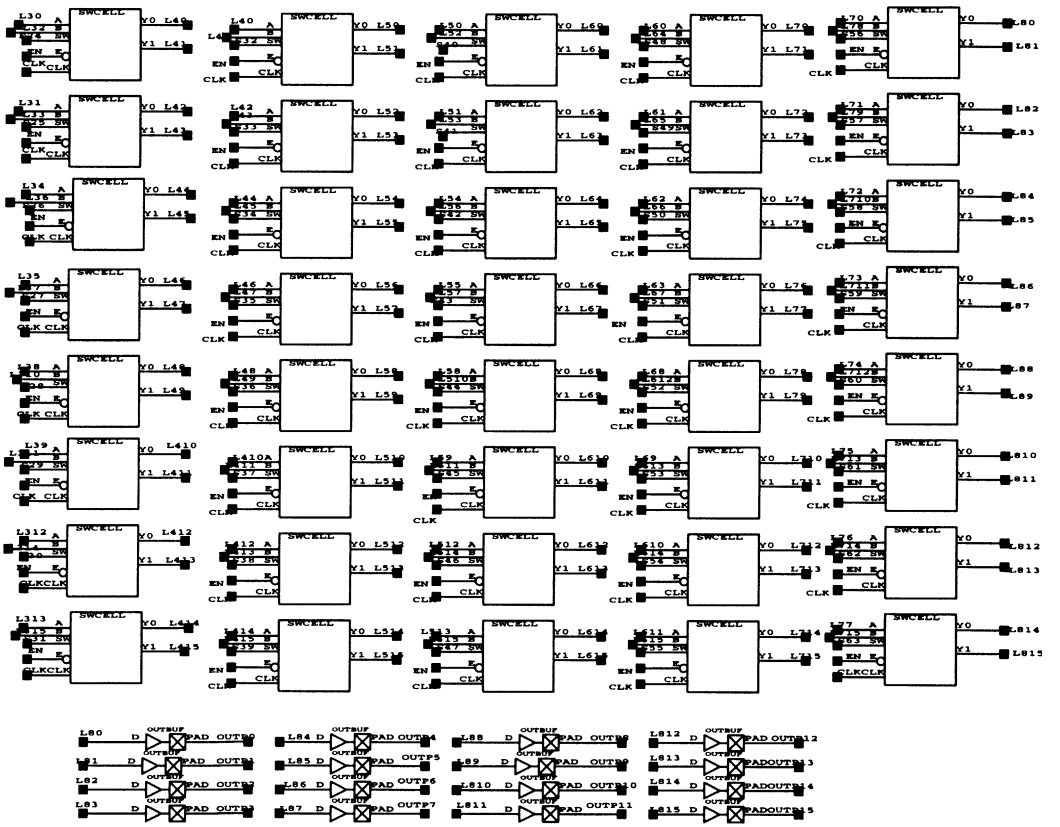


Figure 26. Top-Level View of the MIN Switch Fabric Design



Note The complete multipath interconnect switch fabric is implemented by using 128 (8 x 8 x 2) multiplexed flip-flops of the type DFME1A. The straight multiplexed switch structure discussed in the previous section requires 80 (5 x 16) DFM6A multiplexed flip-flops. However, this size differential does not translate for larger values of N (where N is the number of input and output ports). As N gets larger, the number of modules required to implement the MIN network does not increase as rapidly as it does for the simple mux-structured network. Also, notice that the multiplexed flip-flop used in the MIN network is a 2:1 type, whereas the straight mux switch fabric requires a 4:1 type. The 2:1 multiplexed flip-flop offers the advantage that, because of its lower fanin, it is easier to route on the Actel software.

Timing Analysis

The MIN switch fabric discussed here can be implemented in most Actel ACT 3 devices, such as the A1425A, the A1440A, A1460, and the A14100A. The timing analysis given in this section was obtained from the A1440A-2.

The MIN switch fabric can be operated as fast as the slowest switching element can switch its data from input to output. The basic switching element has a 5.7 ns clock-to q (input-to-output) delay, along with 0.7 ns of setup time. Hence, the maximum frequency of the switch fabric clock is 156 MHz. In a 16:16 switch, this provides a maximum throughput of 2.5 gigabits per second. Note that it takes eight clock cycles for data to move from the switch fabric input to its output (about 51.2 ns). However, as in all such pipelines, once all stages of the network are filled up, data is output at every subsequent clock cycle.

Conclusion

The basic concept behind switch fabrics is multiplexing data from input ports to outputs. The multiplexer-based architecture of Actel FPGAs fits this requirement. High-speed switching networks of almost any topology can be implemented efficiently using the multiplexed flip-flops in the Actel library. Each of these flip-flops is mapped to only one sequential module within the FPGA to take maximum advantage of the die area within the chip.

Generating/Checking CRC for IEEE 802.3 (LAN interface)

The Carrier Sense Multiple Access with Collision Detection (CSMA/CD) media access method is the means by which two or more stations share a common bus transmission medium. IEEE 802.3 is a standard for local area networks (LAN) employing CSMA/CD as the access method.

IEEE 802.3 consists of submodules such as Receiver, Transmitter, Clock Synchronization, and CRC, as shown in Figure 27.

The focus of this application note is the design of a Cyclic Redundancy Check (CRC) submodule compatible with the IEEE 802.3 protocol. This includes generating the CRC code before transmission and checking the coherency of the data at receiver time.

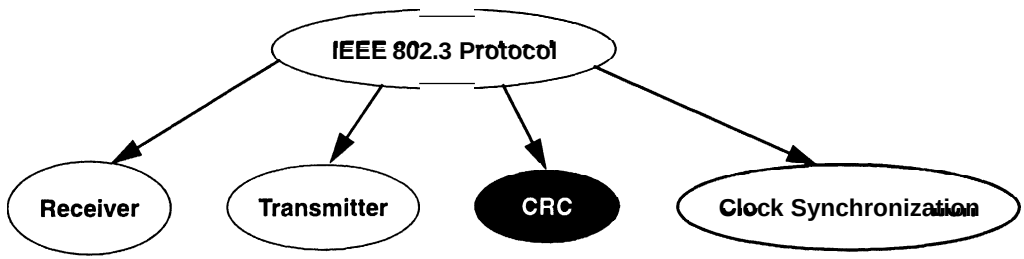


Figure 27. IEEE 802.3 Protocol Submodules

Cyclic Redundancy Check

CRC is a way to detect small changes in blocks of data. Although a few errors in a text file may be acceptable, when transmitting a computer program, an error of even 1 bit is sufficient to make a program faulty. An error-correcting protocol triggered by a CRC error detector can provide protection. The CRC code calculation usually is different from one protocol to another. This application note focuses on the LAN 802.3 protocol algorithm and includes a design implementation and the issues involved in this implementation.

IEEE 802.3 Frame Structure

A Media Access Control (MAC) frame packet is partitioned into six major sections: Preamble, Destination, Source Address, Byte Count, Data Field, and Frame Check Sequence (FCS). Table 3 demonstrates the LAN frame structure.

*Table 3. **LAN Frame** Structure*

Preamble	8 Bytes
Destination Address	6 Bytes
Source Address	6 Bytes
Byte Count	2 Bytes
Data Field	46 to 1500 Bytes
Frame Check Sequence (FCS)	**4** Bytes

The Preamble field is used for synchronization between the receiver and transmitter clock. This field has seven sequences of 101010 followed by one byte of 10101011. This is the last byte indicating the start of the frame—start Frame Delimiter (SFD) byte.

Each MAC frame has two address fields: the destination address field and the source address field. The destination field specifies the station or stations for which the frame was intended. The source address field indicates the station sending the frame.

The Byte Count field is a 2-byte field; its value indicates the number of logical link control (LLC) data bytes in the data field.

The data field contains a sequence of n bytes that may arbitrarily appear in the data field. The maximum size of this field is (2 x (address size) + 48)/8 bytes, and the minimum frame data is ((8 x n) + (2 x address size) + 48) bits.

The FCS field contains CRC code. This field is 4 bytes long and is used to check the integrity of the transmitted data on the LAN protocol.

CRC Module Design

The first task in designing a CRC module is to find and understand the linear algebra that represents the CRC code calculation. The CRC algorithm operates on a block of data transmitted serially as a unit. This block of data can be looked at as a large numerical value. The CRC algorithm divides this large number by a magic number—the CRC polynomial. This operation will leave the remainder with a unique CRC code. After CRC code calculations, the resulting CRC number is usually stored along with the data. The following describes this linear division for a 32-bit 802.3 LAN controller:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

When data is received from storage, the unique CRC code is received along with the data. The CRC design algorithm can be repeated, and the remaining results should be the magic CRC number. This number for LAN 802.3 protocol is DEBB20E3 hex, and since the number is inverted before transmission on the line, it is C704DD7B hex. When checking for validity of data, any other remainder in the CRC register is an indication of error.

The CRC design interface for IEEE 802.3 is completely synchronous and register intensive. This makes it ideal for Actel architecture. CRC design consists of many add and shift operations in every clock cycle. This makes it an ideal choice for behavioral design entry methodology. Using a schematic design tool is not advisable, since design is error prone and schematics are hard to change.

Design Implementation

The following is the `crc_design` file implemented in ACTmap VHDL with some predefined sequential procedures that can be used to define this design. The behavioral code is then processed by Actel's VHDL synthesis and optimization tool, ACTmap. ACTmap is a computer-aided design tool for working with the Actel families of FPGAs. It performs three basic functions:

- PALASM2, VHDL to netlist translation
- Netlist Optimized mapping
- 1 / 0 insertion

ACTmap reads the PALASM2, or VHDL source file and translated it into either an EDIF or an ADL (Actel Design Language) output file or Verilog netlist. The output file that it generates is optimized for a specific family of Actel FPGAs (ACT 1, ACT 2, 1200XL, or ACT 3).

LAN 802.3 also includes the Receiver and Transmitter submodules. Both Receiver and Transmitter submodules are counter intensive. The Actel ACTgen macro generator can create structural macros such as counters that are optimal for Actel devices. These structural blocks can easily be instantiated through Actel VHDL. Even though receiver and transmitter designs are not the focus of this application note, it is informative to know that the ACTgen module generator can easily be linked to the Actel VHDL entry tool.

This application note is focused on generating the correct CRC code at the transmission time and checking for validation of frame data at receiving time. A functional block diagram of the CRC design is shown in Figure 28.

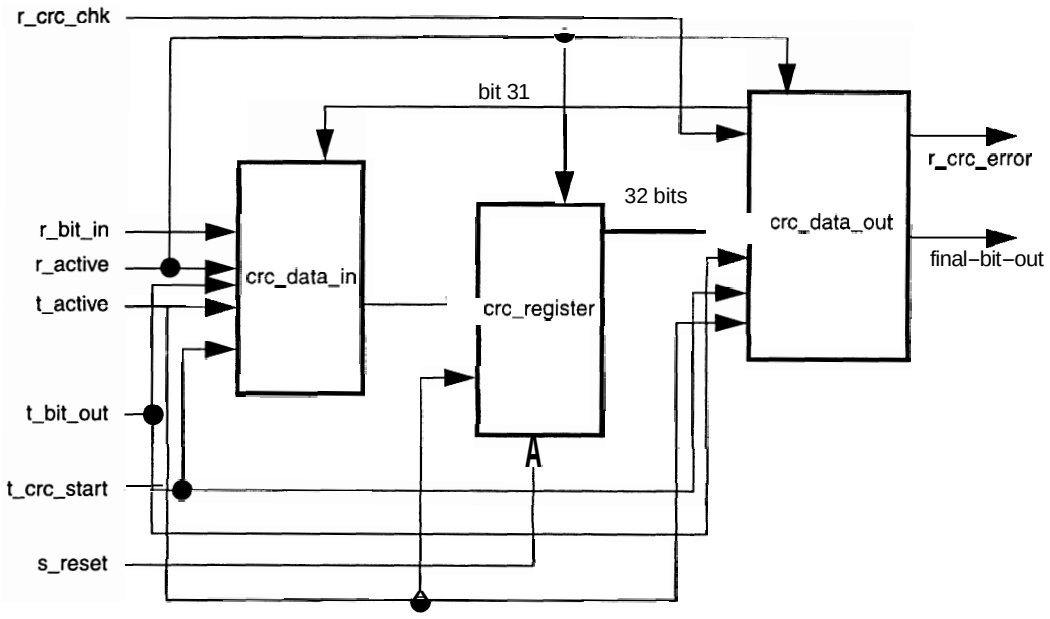


Figure 28. Functional Block Diagram

VHDL Code Description

The following is the ACTmap VHDL code representing this design:

```
--LIBRARY ieee; used for simulation
--USE ieee.std_logic_1164.ALL; used for VHDL simulators

entity crc-data-in is

--decides which data should be
-- sent to crc-register entity.

     port (rcv_active,trm_active: in bit;
     rcv_bit_in,trm_bit_out,crc_reg_b, trm_crc_start: in bit;
     crc-data-in: out bit);
end crc-data-in;

architecture archi of crc-data-in is
signal mux-data-out: bit;
begin

   mux_data_out<= trm_bit_out when (trm_active = '1') else
            rcv_bit_in when (rcv_active = '1') else
            '0';
--This will feed bit 31 of crc-register back in, with the new data.
--It also disables the crc calculation when transmitting crc code.

crc-data-in <= not(trm_crc_start) and (crc_reg_b xor mux-data-out );

end archi;

entity crc-register is
--This is the main module that applies the crc algorithm to data.

port(   clk,system_reset,crc_data_in:   in bit;
       rcv_active,trm_active:   in bit;
       crc_reg         :out bit_vector(31 downto 0);
       crc_reg_b       :out bit);
end crc-register;


architecture crcreg of crc-register is
signal nextreg, temp :bit_vector(31 downto 0);
signal reset-crc        :bit;

begin

reset-crc <= '1' when system-reset = '1' else
            '1' when  ( (rcv_active = '0') and (trm_active = '0')) else
            '0';
```

```
crc_reg <= nextreg ;
DFFC_V(temp, reset-crc, clk, nextreg);
temp <= (nextreg(30 downto 26) &
                        (nextreg(25) xor crc-data-in) &
                        (nextreg(24 downto 23)) &
                        (nextreg(22) xor crc-data-in) &
                        (nextreg(21) xor crc-data-in) &
                        (nextreg(20 downto 16)) &
                        (nextreg(15) xor crc-data-in) &
                        (nextreg(14 downto 12) ) &
                        (nextreg(11) xor crc-data-in) &
                        (nextreg(10) xor crc-data-in) &
                        (nextreg(9) xor crc-data-in) &
                        nextreg(8) &
                        (nextreg(7) xor crc-data-in) &
                        (nextreg(6) xor crc-data-in) &
                        nextreg(5) &
                        (nextreg(4) xor crc-data-in) &
                        (nextreg(3) xor crc-data-in) &
                        nextreg(2) &
                        (nextreg(1) xor crc-data-in) &
                        (nextreg(0) xor crc-data-in) &
                        crc-data-in);
                        crc-reg-b <= nextreg(31);

end crcreg;



entity crc-data-out is
-- This is the crc-data-out module that checks for errors.

   port (rcv_active,trm_active: in bit;
        trm-data, crc-reg-b: in bit;
        trm_crc_start,rcv_crc_chk: in bit;
        crc-reg: in bit_vector(31 downto 0);
        trm_bit_out,rcv_crc_error: out bit);

end crc-data-out;

architecture archi of crc-data-out is
signal crc-constant : bit_vector(31 downto 0);

begin


trm_bit_out <= (not crc-reg-b) when ((trm_active = '1')and (trm_crc_start = '1')) else
          trm_data when (trm_active = '1')else
          '0';
-- checking for constant magic number
crc-constant <= x"c704dd7b";
```

```
rcv_crc_error <= '0' when (rcv_active = '0') else
        '0' when ((crc_reg = crc_constant) and (rcv_crc_chk = '1')) else
        '1' when (rcv_crc_chk = '1') else
        '0';
end archi;

-- This is the top-level module that binds all entities together.
entity enetcrc is

   port(clock,t_crc_start,r_bit_in,t_bit_out :in bit;
      s_reset,r_active,t_active,r_crc_chk :in bit;
      final_bit_out,r_crc_error     :out bit);
   end enetcrc;


architecture structure of crc_design is

component crc-data-in
       port (rcv_active,trm_active                    : in bit;
          rcv_bit_in,trm_bit_out,crc_reg_b,trm_crc_start  : in bit;
          crc_data_in                                 : out bit);

end component;

component crc_register
   port (clk,system_reset,crc_data_in:   in bit;
          rcv_active,trm_active:   in bit;
          crc_reg         :out bit_vector(31 downto 0);
          crc_reg_b       :out bit);


end component;
component crc_data_out
          port (rcv_active,trm_active: in bit;
             trm_data, crc_reg_b: in bit;
             trm_crc_start,rcv_crc_chk: in bit;
             crc_reg: in bit_vector(31 downto 0);
             trm_bit_out,rcv_crc_error: out bit);

end component;

for all: crc_data_out use entity work.crc_data_out(archi);
for all: crc-data-in use entity work.crc_data_in(archi);
for all: crc_register use entity work.crc_register(crcreg);

signal bit31,data_in      :BIT;
signal crc32bit_reg       :bit_vector(31 downto 0);
```

```
begin

   U1:crc_data_in port map
   (r_active,t_active,r_bit_in,t_bit_out,bit31,t_crc_start,data_in);

   U2:crc_register port map
   (clock,s_reset,data_in,r_active,t_active,crc32bit_reg,bit31);

   U3:crc_data_out port map
   (r_active,t_active,t_bit_out,bit31,t_crc_start,r_crc_chk,crc32bit_reg,
    final_bit_out,r_crc_error);

end structure;
```

The entity crc_data_in acts as a switch between receiver and transmitter. The code in this module is designed to use the same circuit for the data receiver CRC check and the CRC code generator at transmission time. In this module, when t_active is enabled, it enables the CRC code generation. The T_bit_out bit will be half added (XOR)with bit 32 of the CRC register and will be shifted in the CRC register for CRC code generation. T_bit_out will also be sent on line serially. At the end of the Byte Count field the t_crc_start bit in the top level will be set. This indicates that the generated CRC code needs to be transmitted. The 4 bytes of the CRC code are transmitted with bit 31 first and bit 0 last.

The crc–register entity generates the CRC code at transmission time and applies the polynomial division on data bits at receiver time. In this module, the DFFC_V predefined procedure of ACTmap_VHDL is used. The cyclic redundancy check is computed at transmission time as a function of all the frame data fields except Preamble and FCS (CRC). The CRC algorithm applies modulo 2 division on data; this means it uses the XOR operation instead of the normal add and subtract. Note that bit 31 of the crc–register is folded back into the polynomial operation. When t_crc_start is set, the uniquely calculated CRC code will be shifted on line at every clock period. As the most significant bit is shifted out, the least significant bit will be replaced by 1.

In the receiving side, all previously mentioned fields except for Preamble (including FCS) run through the CRC algorithm again. When the last CRC code is received, the CRC check flag will be set (r_set_chk).

The last entity, crc_data_out, will check the number remainder in the crc–register for the constant magic number C704DD7B hex. Any other number produces an error.

The top-level design, enetcrc, instantiates these modules and binds them together.

Synthesis and Optimization

The VHDL code described in the previous section is synthesized and optimized by ACTmap, which generates a gate level description optimized for Actel's architecture. ACTmap can output the results in any of the following netlist formats: EDIF, Verilog, ADL, and Viewlogic. The following results were obtained from ACTmap targeting an ACT 3 device.

Figures 29 through 34 show the schematic representation of the netlist generated by ACTmap.

Conclusion

Trying to find better ways to quickly bring high performance products to market is not new. What is new is a design methodology combining the use of Field Programmable Gate Arrays (FPGAs) with high-level design entry in high-speed systems. New design flows use sophisticated synthesis tools to translate generic high-level design description into device specific netlists. Synthesis targets specific FPGA architectures and devices for optimum fit and performance. The Cyclic Redundancy Check circuit described in this application note takes advantage of high-level design entry capabilities of Actel tools.

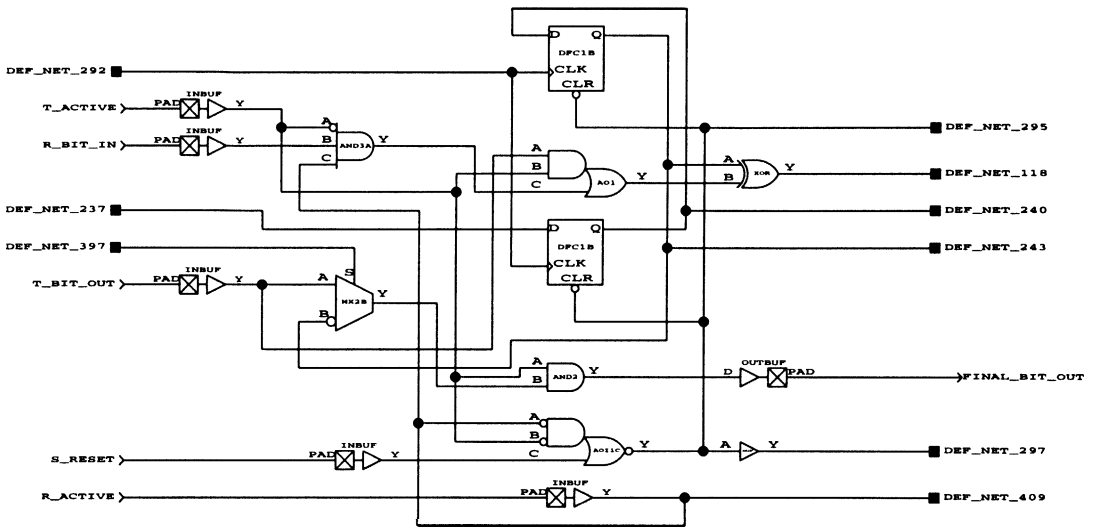


Figure 29. Schematic Representation of the Top Level Design, enetcrc

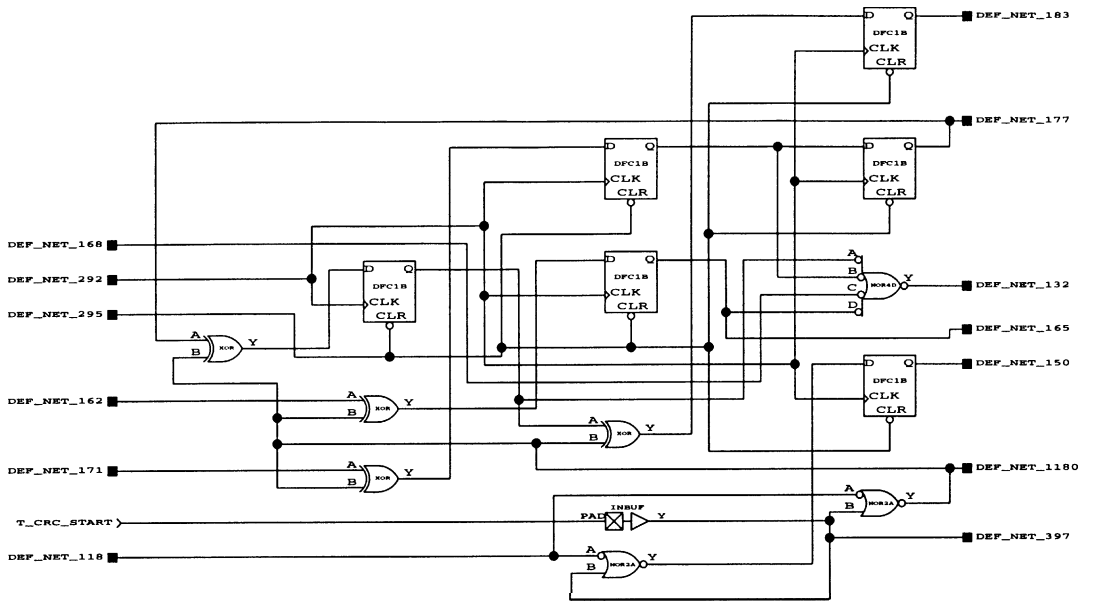


Figure 30. Schematic Representation of the Top Level Design, enetcrc (Continued)

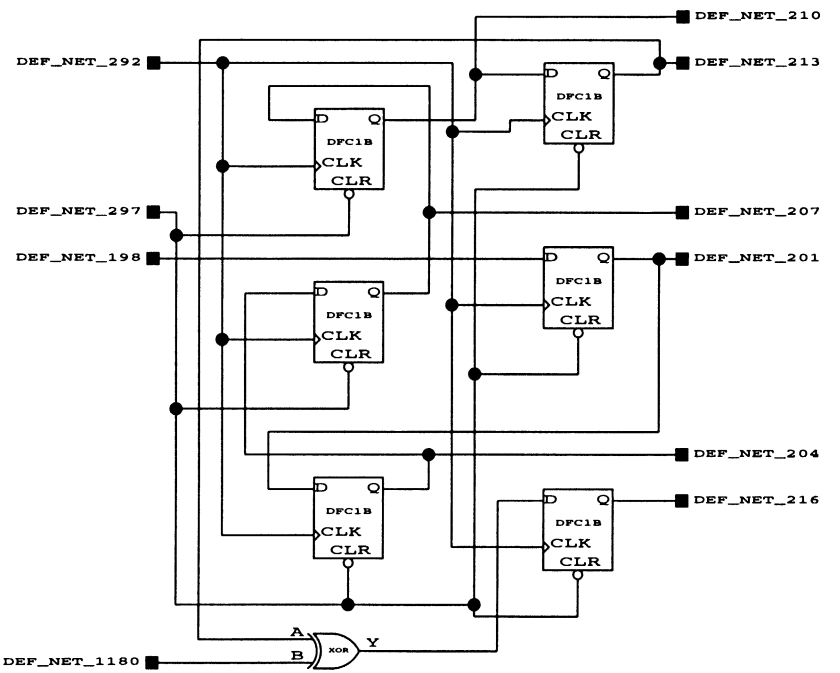


Figure 31. Schematic Representation of the Top Level Design, enetcrc (Continued)

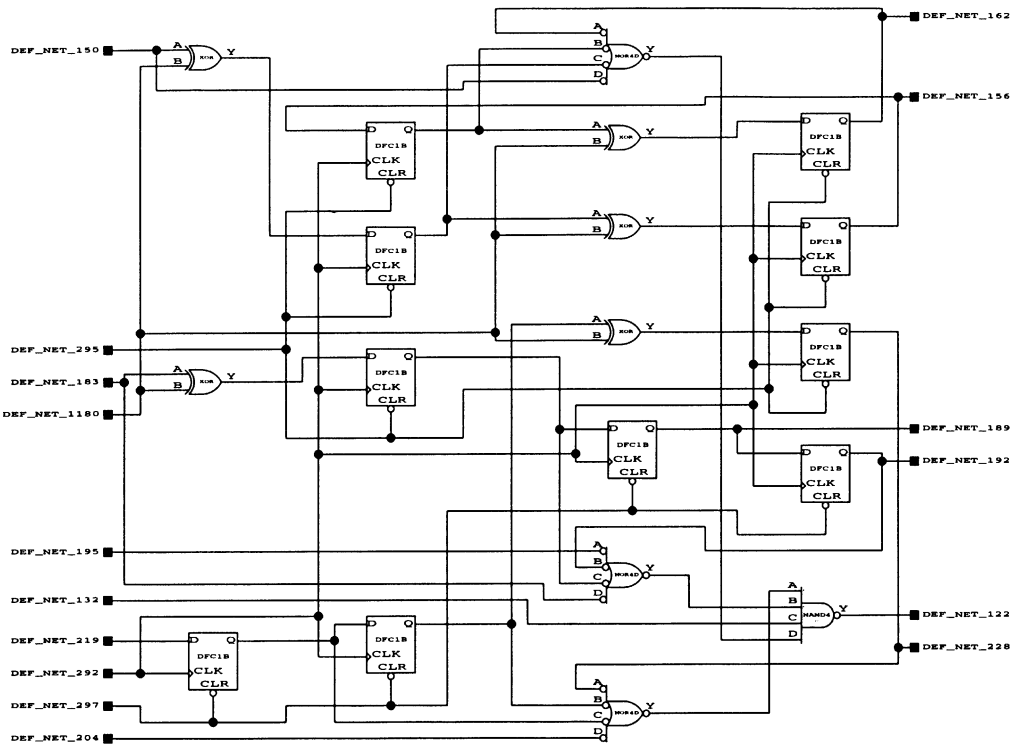


Figure 32. Schematic Representation of the Top Level Design, enetcrc (Continued)

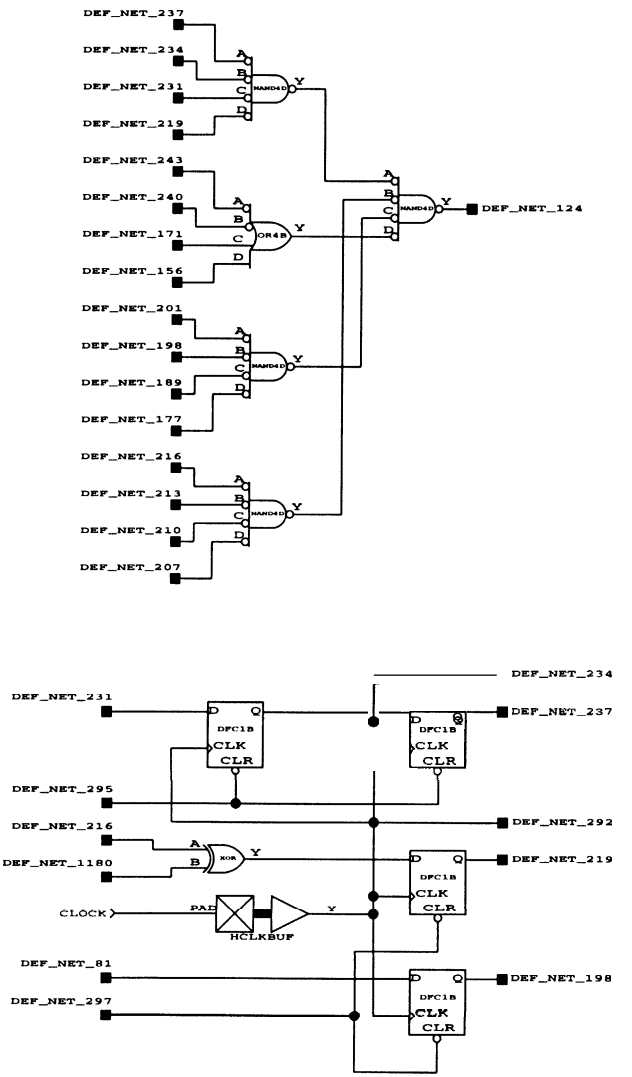


Figure **33.** *Schematic Representation of the Top Level Design, enetcrc (Continued)*

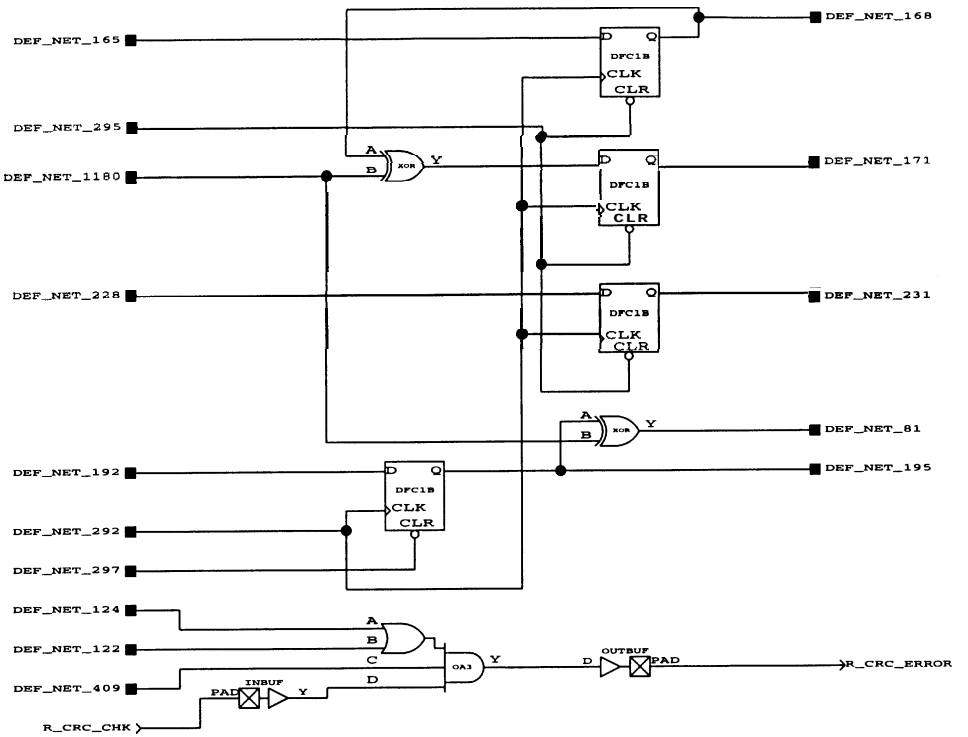


*Figure 34. Schematic Representation of the Top **Level** Design, enetcrc (Continued)*