CODING AND BIT-ERROR-RATE-TEST BLOCKS FOR A SERIAL DIGITAL MULTI-GIGABIT COMMUNICATION SYSTEM

by

David T. Carney

A Report Submitted to the Faculty of the

Milwaukee School of Engineering

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Milwaukee, WI

May 11, 2005

Abstract

The project involves serial digital multi-gigabit communication systems that are emerging for use in chip-to-chip applications in digital systems. These systems communicate data using very high speed point-to-point serial links in a switch fabric architecture between processors and peripherals in digital systems. The major purpose of this project is to develop two reusable building blocks for use on projects containing serial digital multi-gigabit communication systems. The reusable building blocks are an error correcting code (ECC) encoder and decoder appropriate for these systems and a bit-error-rate tester (BERT). Both of these building blocks are designed using digital logic in Very High Speed Integrated Circuit Hardware Description Language (VHDL) to be implemented in a field programmable gate array (FPGA) that contains multi-gigabit serial transceivers.

The project includes a detailed investigation of serial digital multi-gigabit communication systems that was required to determine an appropriate ECC design. Elements of the investigation include the communication channel and bandwidth, random and deterministic noise sources and effects, characteristics of the transmitted data, and a comparison of different types of ECCs. The ECC designed in this project consisted of a maximum run length code stage inside of a 2-error correcting primitive BCH code. The overall code word size is 63 bits, the data word size is 48 bits, and a single padding bit is added to make the code word size 64 bits. A detailed description of the logic design for this code is provided.

The project also includes some investigation into bit-error-rate test methodologies. Some information on the statistical nature of bit-error-rate measurements is developed as well as discussion of different types of data patterns. Three bit-error-rate test patterns are implemented in the BERT block and they are a programmable data word pattern, a $2^{11} - 1$ pseudorandom bit sequence (PRBS) pattern, and a $2^{31} - 1$ PRBS pattern. A detailed description of the logic design for the BERT block is provided. The design of the BERT block is efficient enough to support data rates up to the maximum of the Altera Stratix GX FPGA.

In the project, the ECC block is implemented in an Altera Stratix GX FPGA. The BERT block is used as the data source and also to measure the bit error rate. The bit error rate performance is compared for the coded data and uncoded data running at approximately the information data rate of the coded data. Two physical channels are used in the comparison, one 10-inch backplane channel and one 40-inch backplane channel. The ECC block design of this project is not effective in the 40-inch backplane channel and results in a higher bit error rate than uncoded data. The ECC block is effective in the 10-inch backplane channel, but the bit error rate without coding is already much lower than the target rate. The test results indicate that the ECC block may be more effective when used with equalization. They also indicate that a code with similar error correcting capabilities but a higher code rate may also improve performance, but detailed investigation of this is left as future work.

Table of Contents

List of Figures

List of Tables

Nomenclature

Term or Acronyn	Meaning
AC	Alternating Current
ASIC	Application Specific Integrated Circuit
BCH	Bose-Chaudhuri-Hocquengham
BER	Bit Error Rate
BERT	Bit Error Rate Test
BGA	Ball Grid Array
BPSK	Binary Phase Shift Keying
CDF	Cumulative Distribution Function
CML	Current Mode Logic
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
dB	Decibel
DC	Direct Current
DDR	Double Data Rate
dip	dual in-line package
ECC	Error Correction Code
ECL	Emitter Coupled Logic
FEC	Forward Error Correction
FIFO	First In First Out
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GaAs	Galium Arsenide
Gbps	Gigabits Per Second (10^9 bits per second)
GHz	Giga-Hertz (10^9 Hertz)
Hz	Hertz
IC	Integrated Circuit
IO	Input/Output
kHz	kilo-Hertz (10^3 Hertz)
LED	Light Emitting Diode
LFSR	Linear Feedback Shift Register
Mbps	Megabits Per Second (10^6 bits per second)
MESFET	Metal-Semiconductor-Field-Effect-Transistor
MOSFET	Metal-Oxide-Semiconductor-Field-Effect-Transistor
MRL	Maximum Run Length
nF	nanofarad (10^{-9} farad)
NRZ-L	Nonreturn-to-Zero Level
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCM	Pulse-Code Modulation
PDF	Probability Density Function
PRBS	Pseudo Random Bit Sequence

QPSK	Quadrature Phase Shift Keying
RLGC	Resistance, Inductance, Conductance, Capacitance elements of a distributed model of a transmission line
rms	Root Mean Square
SDRAM	Synchronous Dynamic Random Access Memory
SERDES	Serializer / Deserializer
Si	Silicon
SDRAM	Synchronous Dynamic Random Access Memory
SMA	Subminiature Version A – type of connector for coaxial cables with a threaded interface.
SONET	Synchronous Optical Networking
UI	Unit Interval – refers to the bit period of a serial transmission
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

1 Introduction

The project is to design and implement error correction coding (ECC) and bit error rate test (BERT) functional blocks for a serial digital multi-gigabit communication system[1]. In recent years there has been a proliferation of fast serial protocols for interconnection in digital systems between chips within boards, across backplanes, and through cables. Some examples are PCI Express, Serial RapidIO, XAUI, and Infiniband [1]. There are two purposes for this project. The first is to increase at Plexus the knowledge and understanding of digital data communications as applied to these types of serial digital interconnections that are becoming common. The second is to develop ECC and BERT functional blocks that can be reused in future designs to shorten the product development cycle.

1.1 General Background

The emerging serial digital multi-gigabit communication systems are the result of the evolution of the digital bus architecture from asynchronous to synchronous to source-synchronous and then to serial. A description of each of these types of digital bus architectures is given in section 2. These systems have some common characteristics with each other especially in the area of digital signaling technology. The signaling is typically baseband pulse-code modulation (PCM) in a nonreturn-to-zero level (NRZ-L) format. A typical system block diagram for chip-to-chip communications using serial

[1] High-speed digital communication links or systems operating at rates above one gigabit per second are referred to herein as multi-gigabit communication systems.

digital multi-gigabit communications is shown in Figure 1. In the system, data and commands can be passed between devices through the switch fabric. The data can be passed directly between Input/Output (IO) devices through the switch fabric without being delayed by transfers to and from the CPU, and without limiting the CPU throughput to other IO devices.

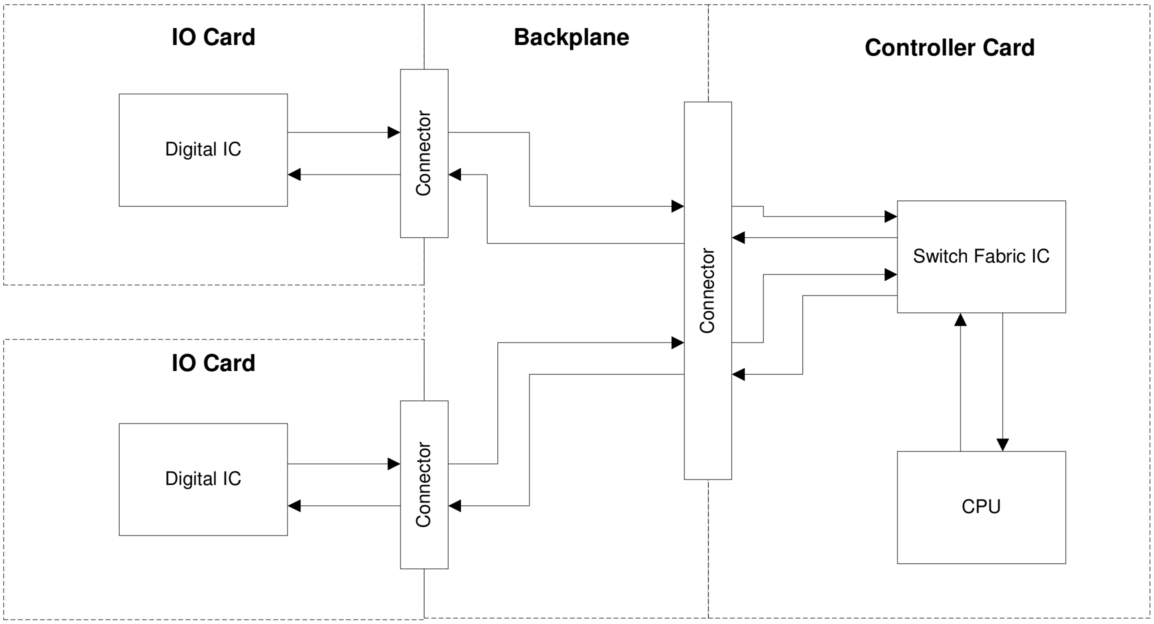


Figure 1 - Typical digital system block diagram for systems using serial digital multi-gigabit communications.

The connections are made by serial links and may actually consist of multiple serial links, operating in parallel, in each direction. The XAUI standard for example uses four independent serial links, each operating at 3.125 Gbps, for a total rate of 12.5 Gbps. The XAUI standard uses 8B/10B coding to assure DC balance so the effective data rate is 10 Gbps [2].

The digital signaling used is similar in format to the older synchronous bus architectures such as Peripheral Component Interconnect (PCI), but in order to operate at the high data rates differential signaling is now commonly used in serial digital multi-gigabit communication systems [2] [3] [4] [5]. The type of buffer commonly used is Current Mode Logic (CML) and this is the type of signaling that is supported by both Xilinx and Altera devices [6] [7].

The transmission channel consists of several elements that connect the differential transmitters on one IC to the differential receivers on another IC. These elements include the connection from the die of the IC to the board referred to as the package connection, the connections called vias in the printed circuit board, used to transition between routing layers, the traces in the printed circuit board, and the connectors and cables used to connect printed circuit boards together. For the system shown in Figure 1 the worst-case transmission channel is from the IC on an IO card to the switch fabric IC on the controller card. This will be the transmission channel that will be discussed throughout this project report. A block diagram of this transmission channel is shown in Figure 2. The transmission channel for each serial link consists of two connections, one for each end of the differential pair (positive and negative). In addition, not shown in the diagram is that the signals on each end of the differential pair propagate between that pair and a reference return plane. Throughout the channel, it is assumed that the traces are far enough apart that there is little coupling between them and they can be referred to as a loosely coupled differential pair.

The primary advantages of differential signaling in a printed circuit board (PCB) transmission channel are immunity to power and ground noise, reduction of and immunity to simultaneous switching noise, and lower emissions. As long as the differential outputs are well balanced and the loads seen by each signal in the pair are symmetrical, the degree of coupling does not have much effect on these advantages. A secondary benefit of differential signaling in printed circuit board routing is that of immunity to crosstalk from nearby neighbors since the crosstalk will affect both traces in the pair. Tighter coupled differential pairs would have slightly more immunity to crosstalk and take up less space because the traces are closer together. The disadvantages of tighter coupling are smaller trace widths required to achieve the same characteristic impedance that make skin effect loss worse, and impedance discontinuities caused when the traces break apart to go around obstacles such as vias. The disadvantages usually outweigh the advantages especially at multi-gigabit data rates where high frequency loss and reflections off impedance discontinuities are concerns. There are trade offs in choosing loosely coupled or tightly coupled differential pairs and those must be weighed carefully in system design [8].

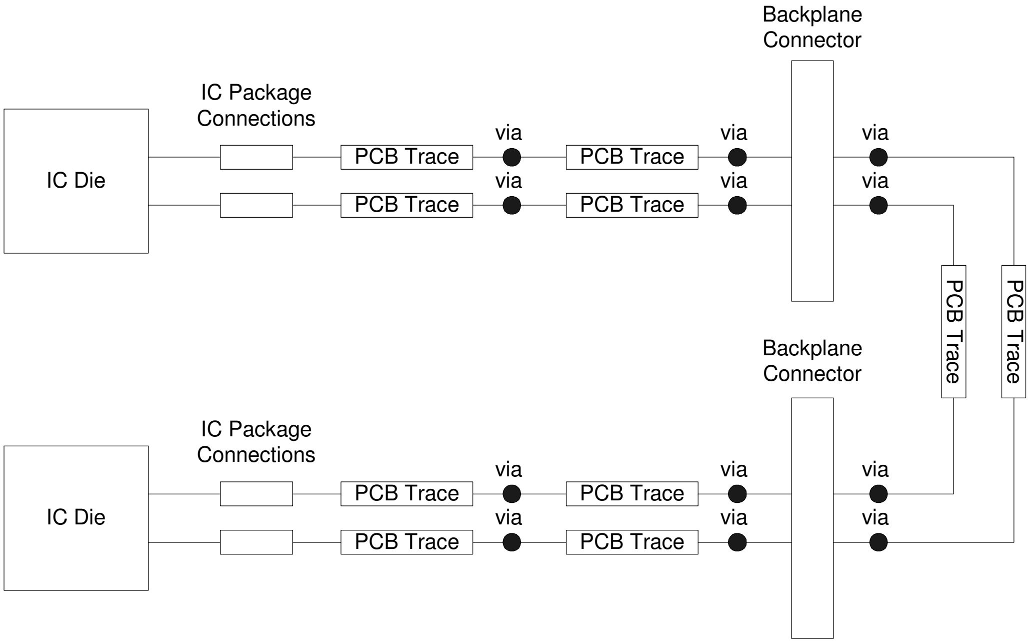


Figure 2 - Worst-case transmission channel.

Another common feature in serial digital multi-gigabit communication systems is the use of AC coupling between transmitter and receiver. The use of AC coupling requires that the data be coded in a way to limit the frequency content of the signal to what can be passed through the AC-coupled transmission path. There are three main advantages to using AC coupling. The first is that AC coupling allows for different DC bias voltage levels at the transmitter and receiver. This provides a lot of flexibility in system design and allows many different parts to operate together. Next, AC coupling also allows the outputs to be shorted to ground either accidentally or intentionally without causing damage to the drivers at an interface with a removable connection. Finally, AC coupling allows for the interconnection of two devices without requiring any common return connection between them and this can be useful when connecting different systems together. This can be very important for networking applications.

The use of AC coupling, as well as the associated requirement for a maximum run length of logic 0s or 1s to ensure proper clock recovery, leads to the use of run-length-limited and DC-free codes in serial digital multi-gigabit communication systems. One such code that is commonly used is referred to as 8B/10B coding and was developed by Widmer and Franaszek [9]. This code maps 8-bit data words into 10-bit transmission words. It guarantees a maximum run length of 5 bits at any given time and a maximum running disparity between 0s and 1s of 6 at any given time [9].

1.2 Serial Digital Multi-Gigabit Communication System Standards

Some metrics are presented in Table 1 to compare some of the serial digital multi-gigabit communication standards. The information in Table 1 was obtained from the applicable standards for each protocol [2] [3] [4] [5] [10] [11].

Table 1 - Comparison of serial digital multi-gigabit communication standards.

	XAUI	**Infiniband**	**Serial Rapid IO (3.125 Gbps, Long Run)**	**Fibre Channel**	**PCI Express**
Transmission rate per channel	3.125 Gbps	2.5 Gbps	3.125 Gbps	2.125 Gbps	2.5 Gbps
Effective transmission rate per channel after coding	2.5 Gbps	2.0 Gbps	2.5 Gbps	1.7 Gbps	2.0 Gbps
Total channels	4	1, 4, 12	1, 4	1	1, 2, 4, 8, 12, 16, 32
Maximum total transmission rate	12.5 Gbps	30 Gbps	12.5 Gbps	2.125 Gbps	80 Gbps
Maximum total transmission rate after coding	10 Gbps	24 Gbps	10 Gbps	1.7 Gbps	64 Gbps
Maximum data packet size (bits)	12144	32768	2048	16896	32768
Minimum data packet size (bits)	512	160	32	32	32
Bit error rate	1.0e-12	1.0e-12	1.0e-12	1.0e-12	1.0e-12
ECC method	CRC error checking when used with 10Gbps Ethernet, 8B/10B code error detection	CRC error checking, 8B/10B code error detection	32- / 16-bit CRC error checking, 8B/10B code error detection	CRC error checking, 8B/10B code error detection	32- / 16-bit CRC error checking, 8B/10B code error detection
DC-free code	8B/10B coding	8B/10B coding	8B/10B coding	8B/10B coding	8B/10B coding
Signaling output standard	Differential – max 1.6 Vp-p swing	Differential – 1.0 to 1.6 Vp-p swing	Differential – 1.0 to 1.6 Vp-p swing	Differential – 0.325 to 1.0 Vp-p swing	Differential - 0.8 to 1.2 Vp-p swing
Minimum receiver input thresholds	Differential - 0.2 Vp-p	Differential – 0.175 Vp-p	Differential – 0.175 Vp-p	Differential – 0.2 Vp-p	Differential – 0.175 Vp-p
AC coupling (or coupling capacitance)	Yes	470 nF minimum	Yes	Yes for inter-enclosure, Optional for intra-enclosure	75 – 200 nF
Equalization	Not required	Pre-emphasis or passive equalization	Not required	Required in some applications	De-emphasis
Transmitter signal rise time	60 ps – 130 ps	100 ps minimum	40 ps minimum	75 ps - 192 ps	50 ps minimum
Bit period	320 ps	400 ps	320 ps	470.59 ps	400 ps
Maximum total jitter tolerance at receiver	0.65 Unit Interval (UI)	0.65 UI	0.6 UI	0.62 UI	0.6 UI
Ratio of minimum signal rise time to bit period	0.1875	0.25	0.125	0.159	0.125

1.3 Altera Stratix GX Development Board

There are two major suppliers of programmable devices that contain multi-gigabit serial transceivers. They are Xilinx and Altera. Xilinx uses multi-gigabit serial transceivers in their Virtex II Pro line of Field Programmable Gate Arrays (FPGAs) and Altera has them in their Stratix GX line of FPGAs. This project will use an Altera Stratix GX FPGA as the implementation and hardware test platform. Some comparisons between the Altera Stratix GX and the Xilinx Virtex II Pro transceivers are shown in Table 2. The device with the largest number of transceivers from each manufacturer has been chosen for comparison and the actual Altera device (EP1SGX25FF1020) on the Stratix GX development board to be used in this project has also been included. The information in Table 2 was obtained from the datasheets for these devices from Xilinx and Altera [6] [7].

A development board from Altera has been used for testing of the bit-error-rate-test block and the error correction code block in this project. The development board contains many different features described in detail in its datasheet [12]. The parts of the development board used for this project were the Altera Stratix GX transceiver FPGA device (part number EP1SGX25FF1020-5ES), the HM-ZD backplane connector and the SMA connectors connected to the FPGA transceiver pins, the 7-segment displays and other LEDs for status information display, and the dip switches for test setup.

The primary external connections for the Altera Stratix GX development board to be used in this project are the connections from one of the quad transceivers to a Tyco HM-ZD style backplane receptacle. This will provide a good approximation of a real transmission channel including a backplane connector. A test backplane card containing a mating header for the receptacle on the Altera board will be used. This test backplane card will loop the transmit differential pairs back to the receive differential pairs and function as a loopback connector approximating an actual transmission channel.

Table 2 - Comparison of Altera and Xilinx multi-gigabit serial transceivers.

	Altera Stratix GX (EP1SGX25FF1020)	**Altera Stratix GX (EP1SGX40GF1020)**	**Xilinx Virtex II Pro (XC2VP70-7FF1704C)**
Maximum transmission rate per channel	3.1875 Gbps		3.125 Gbps
Total channels	16	20	20
Parallel data width	8, 10, 16, 20 bits		8,10,16,20,32,40 bits
Bit error rate	1.0e-12		1.0e-12
DC-free code built in	8B/10B Coding		8B/10B Coding
Error detection built in	None		CRC
Maximum run length for clock recovery	80 UI		75 UI
Signaling output standard	Differential – 0.35 to 1.6 Vp-p swing		Differential – 0.8 to 1.6 Vp-p swing
Receiver input thresholds	Differential 0.17 Vp-p swing		Differential 0.175 Vp-p swing
Internal termination	50, 60, 75 Ohm		50, 75 Ohm
AC coupling	Required - Output common voltage is different than input so AC coupling is required		Not required – Output and input common voltages are programmable
Equalization	Dynamically programmable preemphasis and equalization		Programmable preemphasis
Transmitter signal rise time	60 ps – 130 ps		120 ps typical
Bit period	313.7 ps		320 ps
Maximum total jitter tolerance at receiver	0.65 UI (not explicitly stated, but compliance to XAUI jitter specification is stated)		0.65 UI
Total jitter output from transmitter	0.3 UI		0.35 UI
Random jitter output from transmitter	0.16 UI		0.18 UI
Intra differential pair skew	10 ps		15 ps

2 Literature Review

The earliest found reference to gigabit serial digital communications was at the IEEE International Microwave Symposium in 1972. Gray presented a paper describing a gigabit digital communication system. The system used QPSK to modulate two 500 Mbps bit streams on a 1.5 GHz carrier. The system contained a multiplexer and demultiplexer that converted from 250 Mbps parallel Emitter Coupled Logic (ECL) signals to 500 Mbps digital streams for modulation [13].

The state of gigabit digital communications by 1979 was that development was still mostly in the conceptual stages. A lot of work had been done at that time on basic circuit building blocks using technologies such as bipolar transistors in Si, GaAs MESFETs and MOSFETs, and charge-coupled devices. Some applications where gigabit electronics were taking hold around this time were in measurement and test, radar and sensing systems, and communication systems. It was noted around 1979 that use in computing applications was probably still a long way off because large scale computing clock rates were still around 10 MHz and the processing power to handle all the data bandwidth available in a gigabit serial link did not exist at that time [14]. It is interesting that some of the signaling technologies described in 1979 such as ECL and Current Mode Logic (CML) have become very common in recent years in multi-gigabit digital communication applications.

By the 1980s, digital gigabit serial communication was being proposed as part of the SONET (Synchronous Optical Networking) standard developed by Exchange Carriers Standards Association. The standard provides for a uniform way of implementing optical telecommunications networks [15]. One of the earliest commercially available devices implementing gigabit serial communications was developed by Vitesse Semiconductor Corp. in conjunction with Bell Communication Research Laboratory, Livingston, NJ. The device was a serializer and deserializer (SERDES) for converting 8-bit parallel data to serial data at 1.24 Gbps. The device's part number was the VS8010 and it first went into production in May of 1988 [16]. Apparently this device was targeted at the OC-24 SONET data rate and the earliest uses of gigabit serial data communications were in networking applications.

The use of gigabit serial data communications soon expanded from networking applications to chip-to-chip interconnection applications. As system interconnection bandwidth demands grew, gigabit serial data communications began to be applied to the problem of interconnecting devices locally within printed circuit boards or within chasses. One of the earliest demonstrated examples of this was presented by researchers at the Hewlett Packard Lab, Palo Alto, CA. They presented an entire system operating with serial links at 1.5 Gbps in 1991. The system still used fiber optics for the main transmission channel but was targeted more at computer communication than telecom networking. The chipset was noteworthy because it was one of the first silicon bipolar designs operating at gigabit data rates instead of some of the other chipsets presented up to this point that were designed in GaAs [17].

One of the oldest types of busses for interconnecting devices was the asynchronous multi-drop parallel bus. In this type of architecture many devices are connected together on a shared bus where only one device can send data at a time. A key advantage of such busses in early systems was the efficient use of pins without too much complexity. As Input/Output (IO) bandwidth requirements increased, the parallel bus was enhanced with a move from asynchronous to synchronous bus structures. This allowed techniques such as pipelining and bursting and also allowed for a widening of busses to improve data throughput. A familiar example of a synchronous parallel bus is PCI. The synchronization of the parallel bus allowed for increased communication speed at the expense of increased design complexity [1].

The next evolutionary advancement in the synchronous parallel bus was the introduction of bridges. The bridges allow for various segments of a large multi-drop bus to be isolated from each other to allow for larger overall bus structures with greater throughput possible [18]. The parallel bus was further enhanced by switching from multi-drop to source-synchronous, point-to-point architectures. This enhancement enables full duplex data flow at even higher speeds. The cost of the increased speed is increased design complexity since the signal paths must be more closely matched in length [1].

A limitation with both the bridge architectures and especially with the point-to-point source-synchronous architectures is the number of pins required on the devices. The bridge devices must duplicate the entire parallel interface for the number of busses connected to the bridge. In point-to-point systems with more than two devices connected together, the interface pins must be duplicated for each additional device. The high pin counts that these parallel architectures require along with the strict matching requirements in the printed circuit board make it difficult to scale them up in speed, and increase system cost [1]. An example of a source-synchronous parallel standard is HyperTransport and it allows for a maximum of 12.8 Gbytes/s of bandwidth [19].

The evolution from the parallel bus to the serial bus has been subtle. The source-synchronous parallel bus is close in many respects to the serial busses that are becoming popular. The main difference is that instead of a separate clock signal with data signals all synchronized to it, serial busses embed the clock with the data on a single transmission line. Serial standards allow for multiple lanes of serial data to be transmitted similar to parallel busses, but there are not strict synchronization requirements between lanes because each lane is its own complete communications link. Serial busses eliminate the problems of clock and data skew and are able to work at up to 10 times the rate of a source-synchronous bus. Another advantage of serial busses is that they lower device pin counts because equal amounts of data can be transmitted on many fewer pins as compared to a parallel bus [1].

The advantages of serial busses are not free and come with the added complexity of recovering the clock at the receiver, and requiring communication protocols that embed all the necessary link information in the serial bit stream. Serial busses also allow for much further data transmission than parallel by utilizing communication techniques such as equalization. One drawback that exists with serial communication is transmission latency due to all of the protocol and processing overhead required to serialize, transmit, and deserialize the data. Some recent examples of standards that have been developed for serial communication are SFI-5 (2.5 Gbps per lane with 16 lanes), XAUI (2.5 Gbps per lane with 4 lanes), Serial Rapid IO (2.5 Gbps per lane with 1 or 4 lanes), Fiber Channel (1.7 Gbps per lane with 1 lane), Infiband (2 Gbps per lane with 1, 4, or 12 lanes), and PCI Express (2 Gbps per lane with 1, 2, 4, 8, 12, 16, or 32 lanes) [1].

Source-synchronous standards such as HyperTransport still provide a viable solution for chip-to-chip interconnection, but experts agree that serial interfaces have an advantage in future applications. Some of the advantages of the serial bus are subtle such as the easier printed circuit board (PCB) routing with fewer length-matching requirements. Equalization at the receiver in serial busses allows for much greater transmission length and allows the use of the same dielectric materials such as FR4 currently used in PCBs. A further enhancement that some manufacturers have demonstrated is multi-level signaling in serial busses, which allows for even greater transmission rates [20].

3 Error Correction Code Functional Block

The design of the error correction code block consisted of several steps. The first step was to characterize the typical communication channel used in multi-gigabit digital communication systems. The next step was to describe the sources of noise and the types of bit errors that each source of noise is likely to cause. After that the type of data to be transmitted was defined. A general overview of various types of codes was then developed to help understand which type of code best fits this application. Finally, after all of these aspects of the system were characterized and the different types of coding schemes were understood, a specific error correction code was designed for this type of communication system.

3.1 Communication Channel

The capacity in bits per second of a communication channel was defined by Shannon in [21] and is given by (1), where

$$C = W \log_2 \left(1 + \frac{P}{N} \right). \qquad (1)$$

In this relationship, *C* is the capacity of the channel in bits per second, *W* is the bandwidth of the channel in Hz, *P* is the average signal power of the signal transmitted through the channel, and *N* is the average noise power in the channel. The bandwidth is the absolute frequency range within which a signal is passed. In this absolute-bandwidth definition, no signal energy is passed through the channel outside of the bandwidth. Equation (1) assumes that the noise is Gaussian white noise. The capacity of the channel represents the maximum amount of information that can be sent through the channel with an arbitrarily small rate of errors. In order to achieve this capacity, the data will have to be encoded in a sufficiently complex way.

A more typical measure used in digital communication systems than the signal power to noise power ratio in (1) is a normalized version of signal to noise ratio. This measure is the ratio of the energy per bit to the noise power spectral density and its relationship to the signal power to noise power ratio is given by (2), where

$$\frac{E_b}{N_0} = \frac{S}{N} \times \frac{W}{R} \cdot \tag{2}$$

In this relationship, *S*, *N*, and *W* are the same as in (1), and *R* is the transmission bit rate [22].

3.1.1 Channel Bandwidth

The gradual roll-off bandwidth of a typical communication channel does not match the ideal or absolute bandwidth used in (1) for the channel capacity. In order to establish the capacity of the serial digital multi-gigabit communication system channel, a method of estimating the bandwidth must be used. The two methods considered in this project are the half power bandwidth for the channel and the bounded power spectral density bandwidth for the channel [22]. A block diagram of the system being modeled to determine the bandwidth in this project is shown in Figure 3. This block diagram shows the system without AC coupling capacitors. The analysis of the bandwidth has been organized into two separate analyses, one for the maximum frequency based on the frequency response of the channel without AC coupling capacitors, and one for the minimum frequency based on the frequency response of the AC coupling capacitors and a 100-Ohm differential load.

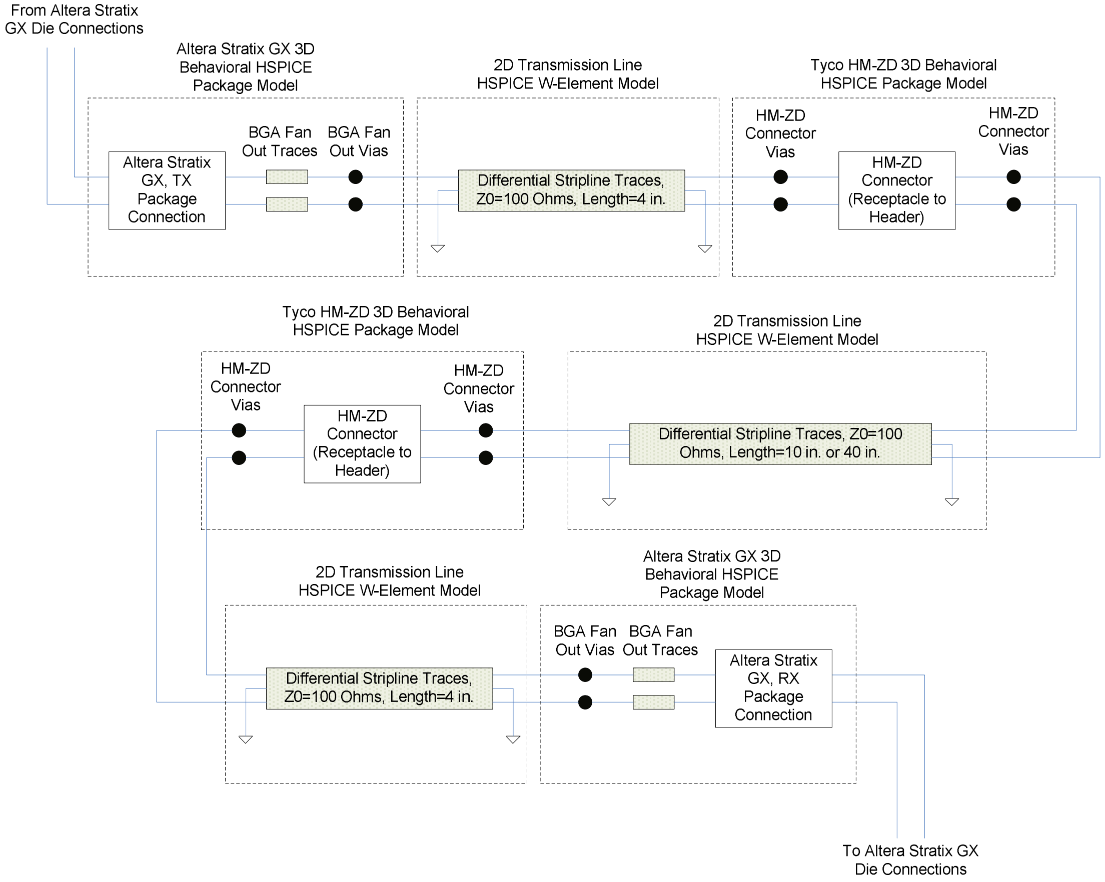

From Altera Stratix GX Die Connections
Altera Stratix GX 3D Behavioral HSPICE Package Model
Altera Stratix GX, TX Package Connection
BGA Fan Out Traces
BGA Fan Out Vias
2D Transmission Line HSPICE W-Element Model
Differential Stripline Traces, Z0=100 Ohms, Length=4 in.
Tyco HM-ZD 3D Behavioral HSPICE Package Model
HM-ZD Connector Vias
HM-ZD Connector (Receptacle to Header)
HM-ZD Connector Vias
Tyco HM-ZD 3D Behavioral HSPICE Package Model
HM-ZD Connector Vias
HM-ZD Connector (Receptacle to Header)
HM-ZD Connector Vias
2D Transmission Line HSPICE W-Element Model
Differential Stripline Traces, Z0=100 Ohms, Length=10 in. or 40 in.
2D Transmission Line HSPICE W-Element Model
Differential Stripline Traces, Z0=100 Ohms, Length=4 in.
Altera Stratix GX 3D Behavioral HSPICE Package Model
BGA Fan Out Vias
BGA Fan Out Traces
Altera Stratix GX, RX Package Connection
To Altera Stratix GX Die Connections

Figure 3 - Differential transmission channel model block diagram.

The model used for the package is a behavioral HSPICE model available from Altera for the Stratix GX device used on the Altera Stratix GX development board used in this project. The models are for a transmitter differential pin pair and a receiver differential pin pair. They cover the connection from the die of the Stratix GX device through the via used to attach the ball grid array (BGA) package to the printed circuit board (PCB).

The models used for the IO and controller card PCB traces are based on typical system implementations. The traces are typically designed to match the 100-Ohm differential impedance of the driver and receiver. The traces are copper differential edge coupled stripline traces that are embedded in PCBs constructed with FR4 dielectric material. The trace parameters and view of the 2-dimensional cross section are shown in Figure 4. From this cross section and the parameters shown, the 2D field solver in Cadence Specctraquest was used to create a SPICE model of the differential trace. The dielectric constant (*Er*) and dielectric loss tangent for FR4 are approximations based on a PCB material comparison document found on Merix Corporation's web site [23]. Merix Corporation is a PCB fabrication company. The trace model is a lossy distributed transmission line model in the form of a w-element model for use with HSPICE. The Specctraquest field solver predicted the differential impedance to be 100 Ohms.

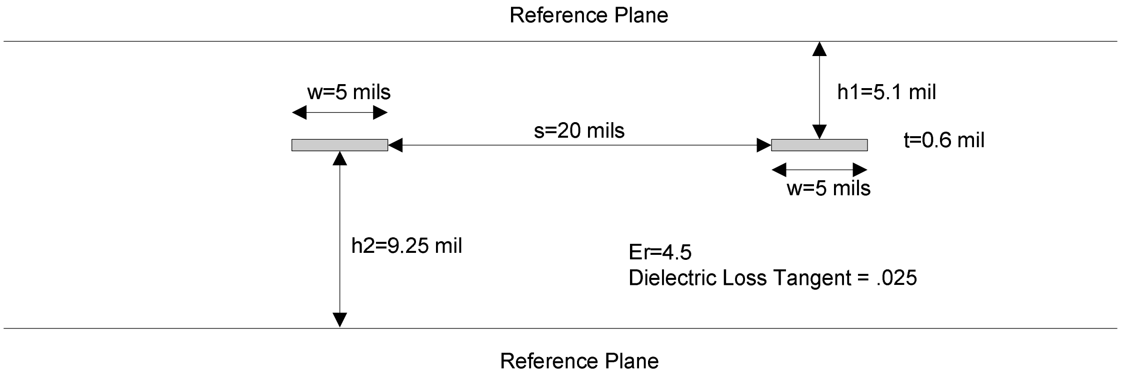


Figure 4 - IO and controller card differential trace, 2-dimensional cross section.

The model used for the backplane traces was created similar to the IO and controller card traces and also based on typical system implementations. The trace parameters and view of the 2-dimensional cross section of the backplane trace are shown in Figure 5.

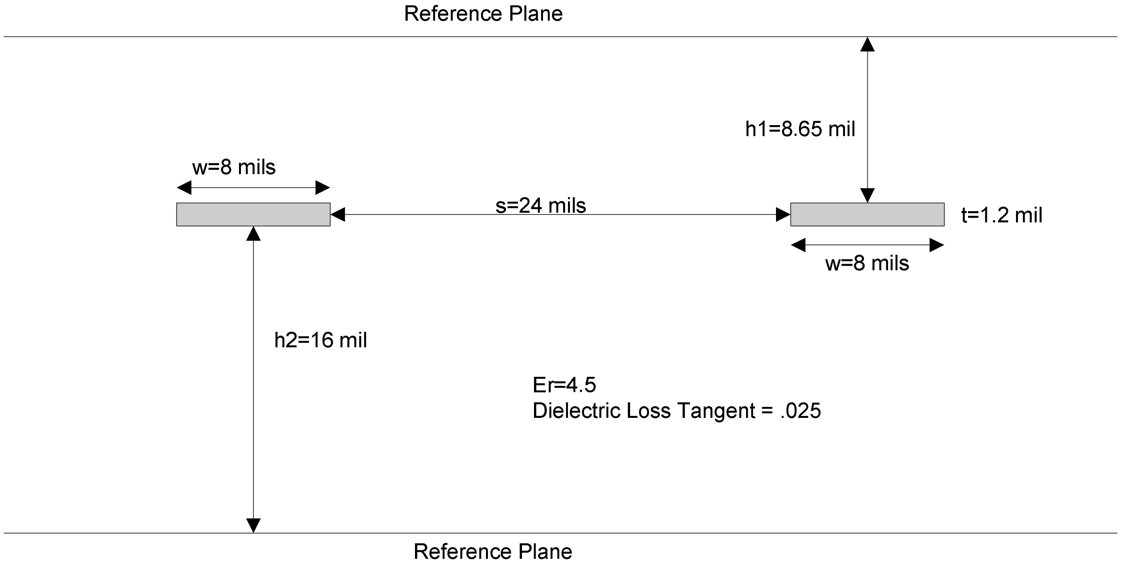


Figure 5 - Backplane card differential trace, 2-dimensional cross section.

The model for the connector is based on a connector model available from Tyco for the HM-ZD backplane connector. The specific HM-ZD connector model used in the simulation is for an 8-row and 4-column version of the connector. The pins that were simulated are differential pairs located in rows E and F and in the middle columns. The Altera Stratix GX development board contains an 8-row, 10-column HM-ZD connector as previously described. The connector model is a behavioral SPICE model for use in simulations with the Synopsys HSPICE tool. The model includes the vias on both sides of the connector.

The models were connected together and simulated in HSPICE with an AC frequency sweep from 1 Hz to 15 GHz. The HSPICE circuit that was simulated is shown in Figure 6. The circuit was simulated for two different lengths of backplane traces which were 10 inches and 40 inches. The simulated power gain of the channel is plotted in Figure 7.

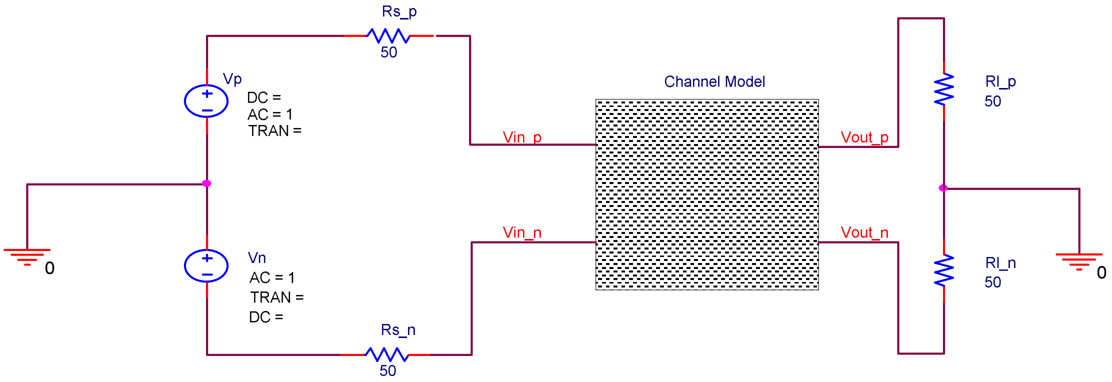


Figure 6 - Channel frequency response SPICE simulation circuit.

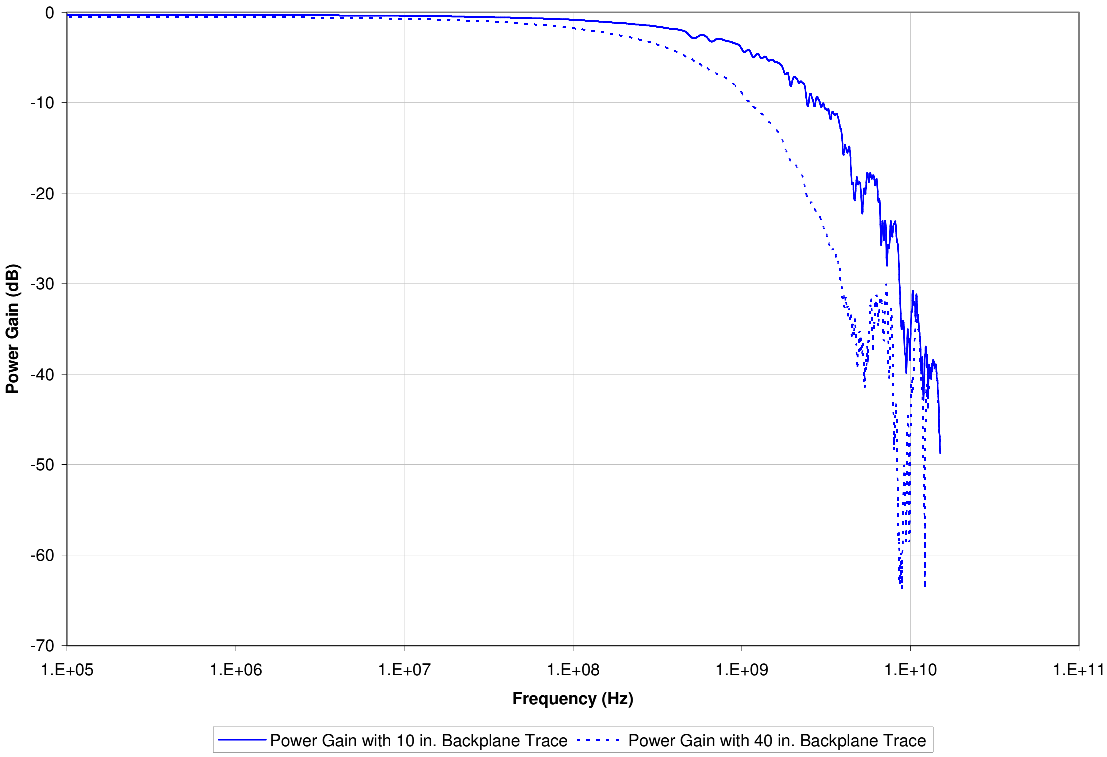


Figure 7 - Backplane channel power gain response excluding AC coupling capacitors.

The point on the gain response where the gain has dropped to –3dB is the maximum frequency used in the definition of the half power bandwidth. For the 10-inch backplane trace, the –3dB point corresponds to 703.0 MHz. and for the 40-inch backplane trace, the –3dB point corresponds to 246.1 MHz. The point on the gain response where the gain drops below and stays below –35dB is the maximum frequency used in the definition of the bounded power spectral density bandwidth [22]. For the 10-inch backplane trace, this corresponds to 11.27 GHz. And for the 40-inch backplane trace, this corresponds to 11.16 GHz.

Another common feature in the transmission channel of serial digital multi-gigabit communication systems is the use of AC coupling capacitors. The AC coupling capacitors perform high pass filtering on the data signal. This leads to a lower frequency limit that is above zero, for the transmission channel. This lower frequency limit has been considered separately from the high frequency limit that was analyzed without including the AC coupling capacitors in the channel simulation model. A SPICE simulation circuit to analyze the low frequency limit due to the AC coupling capacitors for the differential transmission channel is shown in Figure 8. The AC coupling capacitors are *C1* and *C2*. An AC sweep simulation was performed and the differential voltage gain ((Vout_p-Vout_n) / (Vin_p-Vin_n)) was plotted versus frequency. The simulation was performed for two values of AC coupling capacitors, 75 nF and 470 nF to represent the range of capacitor values used in the various serial digital multi-gigabit communication system standards. The plot is shown in Figure 9.

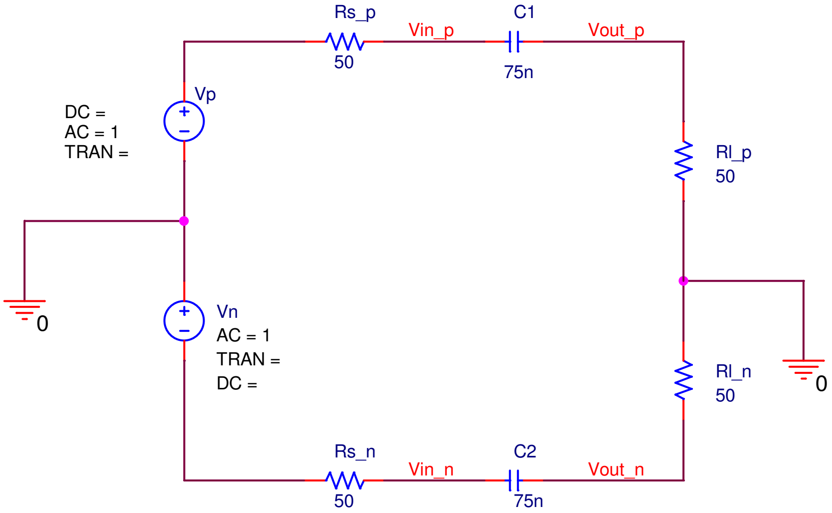


Figure 8 - AC coupling capacitor low frequency cut off simulation circuit.

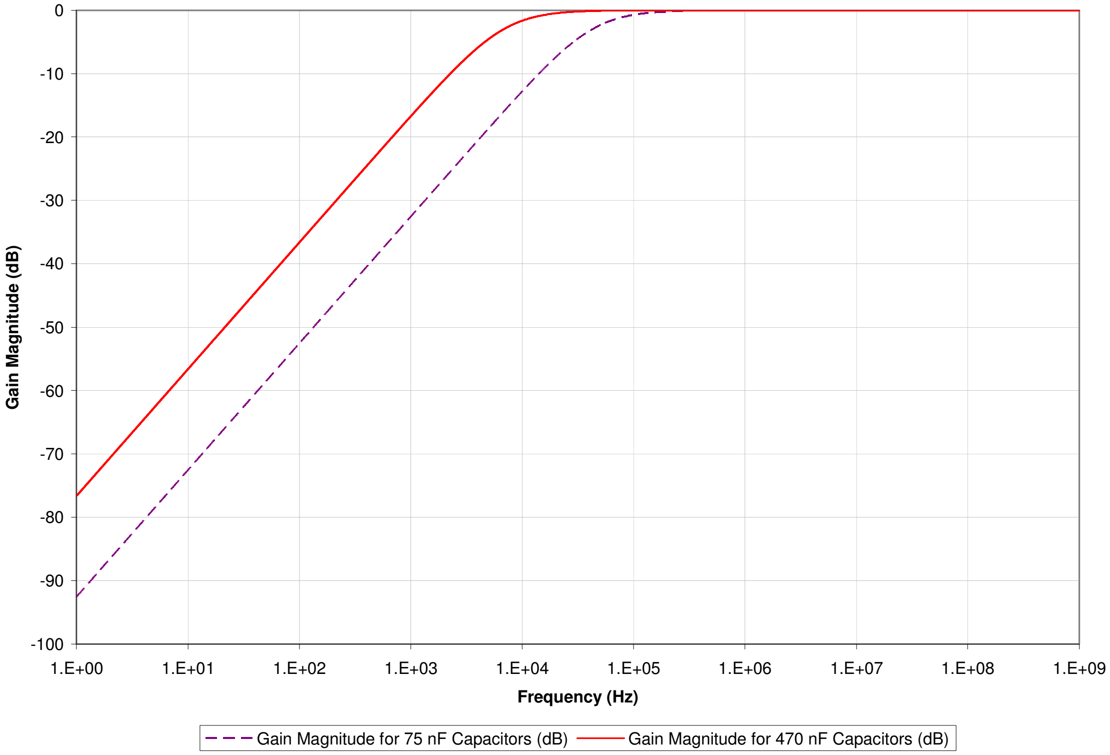


Figure 9 – Frequency response with AC coupling capacitors.

The point on this gain response where the gain has dropped to –3dB is the minimum frequency used in the definition of the half power bandwidth. For the 75 nF capacitors, the –3dB point corresponds to 42.46 kHz and for the 470 nF capacitors, the –3dB point corresponds to 6.776 kHz. The point on the gain response where the gain drops below –35dB is the minimum frequency used in the definition of the bounded power spectral density bandwidth. For the 75 nF capacitors, this corresponds to 754.8 Hz and for the 470 nF capacitors, this corresponds to 120.5 Hz.

The bandwidth for each combination of backplane trace and AC coupling capacitor is shown in Table 3. The AC coupling capacitor has very little effect on the overall channel bandwidth because the low frequency cut off is very small compared to the high frequency cut off.

Table 3 - Channel bandwidth results.

Backplane Trace Length	AC Coupling Capacitor	Half Power Minimum Frequency (Hz)	Half Power Maximum Frequency (Hz)	Bounded Power Spectral Density Minimum Frequency (Hz)	Bounded Power Spectral Density Maximum Frequency (Hz)	Half Power Bandwidth (MHz)	Bounded Power Spectral Density Bandwidth (GHz)
10 in.	75 nF	4.246E+04	7.030E+08	7.554E+02	1.127E+10	703.0	11.27
40 in.	75 nF	4.246E+04	2.461E+08	7.554E+02	1.116E+10	246.1	11.16
10 in.	470 nF	6.776E+03	7.030E+08	1.205E+02	1.127E+10	703.0	11.27
40 in.	470 nF	6.776E+03	2.461E+08	1.205E+02	1.116E+10	246.1	11.16

It is clear from looking at the bandwidth results for the backplane channel that any error correction code chosen should be as high a rate as possible since there is significant attenuation of the higher frequency content of a multi-gigabit PCM NRZ-L signal.

3.1.2 Channel Capacity

Equations (1) and (2) can be combined to give an expression of *E_b* / *N₀* in terms of channel bandwidth and the channel capacity. With the transmission rate (*R*) assumed to equal the channel capacity (*C*), the following equation is the result of this combination:

$$\frac{E_b}{N_0} = \frac{W}{C}\left(2^{\frac{C}{W}} - 1\right). \qquad (3)$$

Using (3), the minimum required value of E_b / N_0 for a transmission rate of 3.125 Gbps corresponding to the maximum rate of the Xilinx Virtex II Pro and Altera Stratix GX transceivers is calculated and shown in Table 4.

Table 4 - Minimum Eb / N0 for transmission channel at 3.125 Gbps data rate.

Backplane Trace Length	Half Power Bandwidth	Bounded Power Spectra Density Bandwidth (GHz)	Minimum Eb / N0 Assuming W = Half Power Bandwidth	Minimum Eb / N0 Assuming W = Bounded Power Spectral Density Bandwidth
10 in.	703.0 MHz	11.27 GHz	6.70 dB	-1.17 dB
40 in.	246.1 MHz	11.16 GHz	27.19 dB	-1.16 dB

More discussion will be given in subsequent sections regarding estimates of E_b / N_0 for serial digital multi-gigabit communication systems and the capacity of the channel based on those estimates.

3.2 Noise Sources

There are many sources of noise in a serial digital multi-gigabit communication system including both random sources and deterministic sources. The various sources of noise occur both within the transmitter and receiver integrated circuits and within the communication channel. Some sources of noise are limited to the transistors in the transmitter and receiver and other sources occur in both the transistors and in the transmission channel interconnection. The noise sources that are referred to as deterministic can be predicted exactly if enough information is known about the communication channel characteristics and the transmitted data characteristics. They are also bounded by the worst-case scenario whereas random noise is not bounded and is assumed to follow a Gaussian distribution.

3.2.1 Random Noise

The first source of random noise is thermal noise and it occurs within the transmitter and receiver transistors, the internal termination resistors, and all of the electrical interconnections. The random motion of electrons in conductors causes differences in voltage across a conductor even if the average current through the conductor is zero. These differences in voltage are the thermal noise. The spectrum of thermal noise is assumed to be flat over the frequencies of interest in serial digital multi-gigabit communication systems and is therefore Gaussian white noise. For MOSFETs, there is also thermal noise within the transistor itself, especially in the channel of the MOSFET [24].

Besides thermal noise, another form of random Gaussian white noise that occurs within current carrying interconnections is called shot noise. This type of noise results from an electrical charge crossing the potential barrier in a transistor. Since all charge exists in discrete electrons, the charge is not continuously crossing the charge barrier but each electron crosses the barrier discretely and the crossing is random in nature. The random distribution of barrier crossings results in the apparent constant flow of current that is observed. Statistical variations in the constant flow of current due to the random nature of barrier crossings results in noise [25].

Another source of random noise occurs within the transistors in the transmitter and receiver devices called flicker noise. A cause of this noise is that at the interface between materials in an integrated circuit there are atoms that have unfilled bonds available because they are not bonded with the differing material. These bonds serve as traps because the free bonds allow extra energy states to trap passing electrons. Electrons are trapped and released some time later. This type of noise has a spectral density that follows a 1/f relationship with frequency [24]. This type of noise is sometimes referred to as pink noise.

3.2.2 Deterministic Noise

The first source of deterministic noise is on the voltage rails that supply power to the transistors in the transmitter and receiver ICs. This noise comes from various sources including the circuits that generate the voltage supplies and voltage drops across the inductance of the power distribution system during switching events (called simultaneous switching noise). Noise on the power supplies causes changes in the behavior of the transistors and leads to noise on the electrical signals used to send the information in serial digital multi-gigabit communication systems. This noise is considered to be uncorrelated to the data [26].

Another type of noise is due to the proximity of the transmission channel elements to other signals and is called crosstalk. Crosstalk occurs in connectors and between traces and the problem increases as design density increases. For the Tyco HM-ZD backplane connector on the Altera Stratix GX development board used in this project, the maximum amount of crosstalk specified by the manufacturer is 1.6% of the signal strength [27]. The amount of crosstalk between the traces can usually be limited in serial digital multi-gigabit communication systems because there are fewer traces to route than in traditional parallel busses and the spacing between traces can be kept large. Crosstalk noise can be correlated or uncorrelated to the data. Crosstalk noise can also be sinusoidal in nature [26].

Intersymbol interference is not actually noise but its effects are similar to noise. The frequency dependent loss of the channel, as illustrated in the bandwidth plots in Figure 7, causes the higher frequency content of the digital signal to be attenuated more than the lower frequency content. This causes a dispersion of a pulse from one bit period into adjacent bit periods. The spreading of a bit into subsequent bit periods degrades the signal to noise ratio and can be thought of as noise. The largest contributor to frequency dependent loss would be the traces that suffer from the skin effect and the dielectric effect. The skin effect loss is proportional to the square root of frequency and the dielectric loss is proportional to frequency [8]. The skin effect is caused by the crowding of high frequency current to the outside surface of conductors, lowering the effective conducting cross sectional area and increasing the effective resistance of the conductor. The dielectric loss is caused by an increase in the conductance with frequency of the dielectric medium surrounding a transmission line [8]. The effects of intersymbol interference can be mitigated through the use of pre-emphasis or equalization. Either of these methods works by amplifying the high frequency components of the signal to counteract the high frequency component attenuation by the transmission channel. Pre-emphasis works by performing the amplification before the signal enters the transmission channel at the transmitter, and equalization occurs by performing the amplification after the signal exits the transmission channel at the receiver. The Altera Stratix GX development board used in this project supports both pre-emphasis and equalization.

Another source of deterministic noise is duty cycle distortion. Duty cycle distortion is usually the result of characteristics of the transistors in the transmitter and receiver devices. Differences in the rise and fall times of signals inside the devices cause a distortion of the bit period depending on if the bit is a 1 or a 0. Both intersymbol interference and duty cycle distortion noise are considered to be correlated to the data [26].

3.2.3 Noise Effects

Crosstalk and intersymbol interference are typically the two largest noise sources in serial digital multi-gigabit communication systems [28]. In an analysis and simulation of similar communication channels in [28], the crosstalk and intersymbol interference are shown to be equal contributors of interference at 1.15 Gbps when no equalization is used. Below 1.15 Gbps, crosstalk is the dominant source of interference and above 1.15 Gbps intersymbol interference becomes the dominant source of interference. Using equalization can shift the crossover point out to 6.25 Gbps.

Noise due to crosstalk and intersymbol interference is bounded and is referred to in the literature as being deterministic [28]. Because the noise is bounded, serial digital multi-gigabit communication systems are designed so that these two factors and all the rest of the deterministic noise sources will leave sufficient noise margin for the system to operate without bit errors. The addition of the Gaussian random unbounded noise on top of the deterministic noise is what will lead to violations of noise margin and bit errors. This conclusion is based on the way in which bit error rates are defined for serial digital multi-gigabit communication systems as explained in [26]. The methodology of defining bit error rates will be explained in more detail in section 3.4.4.

The bit error rate for serial digital multi-gigabit communication systems is defined in terms of jitter instead of the classic E_b/N_0. Jitter is the error in the timing of a signal. Jitter is, strictly speaking, not noise, but a result of noise. Noise causes the transitions of bits to shift in time from their ideal position and this shifting is referred to as jitter. Jitter leads to the closure of the eye diagram because of horizontal variations in signal transitions, and is categorized as deterministic or random jitter. Deterministic jitter is caused by the deterministic noise and random jitter is caused by the random noise. Deterministic jitter sources are classified into the categories of duty cycle distortion, data dependent (ISI or crosstalk), sinusoidal (power supply noise), and uncorrelated bounded (crosstalk) by *Information Technology - Fibre Channel - Methodologies for Jitter Specification* [26]. The total jitter is the combination of the deterministic and random jitter. Just as random noise is unbounded, random jitter is also unbounded. Models are used with defined bit error rates in standards such as Fibre Channel, XAUI, and PCI-Express, to set the maximum jitter allowed and the maximum jitter the receiver must be able to tolerate [26]. It is noted that noise also affects the magnitude of the signal as well as the position in time of the transitions. The magnitude effects show up as the vertical variations in signal transitions, which lead to the closure of the eye diagram. Since the standards define bit error rate in terms of jitter, they are based on the assumption that the probability of bit errors due to magnitude noise must be much smaller than the probability of bit errors due to jitter.

3.3 Transmitted Data

The source data in chip-to-chip communications can come in many different forms such as streaming video, network data packets, and simple data-word read and write requests. This diversity of source data is a reason why all of the serial digital multi-gigabit communication standards reviewed in section 1.2 allow for variable length data payloads. The PCI-Express standard encompasses the whole range of data payload sizes, from 32 bits up to 32768 bits. Any error correction code developed for chip-to-chip communications must permit small enough block sizes so that when used in typical applications, data packets do not have to be padded with non-data bits in order to be transferred. This would be inefficient use of the available data bandwidth. A reasonable data block size for an error correction code would not exceed 32 bits to correspond with the smallest available data block size in the standards. With typical memory bus widths being 64 bits and the future use of processors with 64-bit and 128-bit data paths, choosing 64 or 128 bits as the data block size would also be reasonable.

In addition to the requirement of a small enough block size so that data transfer can occur without needing a lot of padding, another similar requirement relates to the latency for data transmission from the information source to the information sink. The tolerable latency again depends on the type of information being sent, and chip-to-chip communication systems need to support many different types of information. A classic parallel bus standard used in low latency chip-to-chip interconnections is the PCI bus. The latency defined in *PCI-X Addendum to the PCI Local Bus Specification* is 16 clock cycles [29]. Since the PCI bus is a parallel architecture, each clock period represents one block period with a block size as small as 32 bits. A reasonable maximum latency would be 16 times the minimum data block period for PCI-X, which is 7.5 ns (corresponding to 133 MHz PCI-X specification). This would be a maximum latency of 120 ns.

The PCI-Express and Serial Rapid IO specifications both use 8B/10B coding for maximum run length limiting and a 16-bit or 32-bit CRC for error checking depending on the block size and data type. For small block sizes of 32 bits, a 16-bit CRC with 8B/10B coding provides an overall code rate of 0.533. For a larger block size (the largest with Serial Rapid IO) of 2048 bits, the code rate would be 0.788. These numbers provide some insight into the code rates that would be acceptable for the error correction coding in this design project. A code rate closer to 0.788 would be much more desirable.

There are three aspects of the system that are affected by the maximum run length of the coded data. The first is the clock and data recovery that must be performed by the receiver on the transmitted data pattern. The maximum run length allowed by the Xilinx and Altera devices for clock recovery is 75 bits. The second is the magnitude of voltage at the receiver. The transmitted voltage will initially be passed through the AC coupling capacitor because it is transient and not a DC signal. As the transmitted voltage remains in one state though, corresponding to a long string of 1s or 0s, the AC coupling capacitor will block the DC content and the voltage at the receiver will decay according to the RC time constant. For the Xilinx and Altera Devices, the worst-case voltage margin (minimum output voltage of 0.35 V minus minimum required input voltage of 0.17 V) is

0.18 V. A conservative requirement is that voltage margin not drop by more than 1%, due to the coupling capacitor effect, which corresponds to a maximum voltage drop of the voltage at the transmitter of approximately 0.5%. Assuming a 100-Ohm load resistance, a worst case capacitance of 150 nF (two 75 nF capacitors), and a maximum allowed voltage decay of 0.5%, the argument of the logarithmic function in (4) is 0.995 and the maximum time between transitions is calculated using (4) to be 75.2 ns, where

$$t = -R_l C \ln\left(\frac{V_{out} - V_{droop}}{V_{out}}\right). \qquad (4)$$

This corresponds to 150 bits at 2 Gbps and because the maximum run length (MRL) must be limited to 75 bits for clock recovery, the voltage droop due to the AC coupling capacitor will not impose any additional requirement. However, even with the MRL limited to 75 bits, the degradation of noise margin could still exceed 1% if the running disparity is not appropriately limited, that is, if the DC level of the data signal is not appropriately bounded. For example, a data signal that repeatedly contains the 64-bit pattern of 63 logic-1 bits (at + 0.35V) followed by one logic-0 bit (at -0.35V) has an MRL of 63 bits, but has such a large DC level that it will cause the voltage on the other side of a coupling capacitor (after several repetitions of the data pattern) to never reach the minimum level of +0.17V needed to be recognized as a logic-1, and will not meet the noise margin requirement.

The third consideration is deterministic jitter caused by the voltage droop at the receiver during a long string of 1s or 0s. The problem is illustrated in Figure 10, which shows an approximation of the jitter based on the parameters of peak-to-peak voltage, voltage droop, and the 10% to 90% rise time.

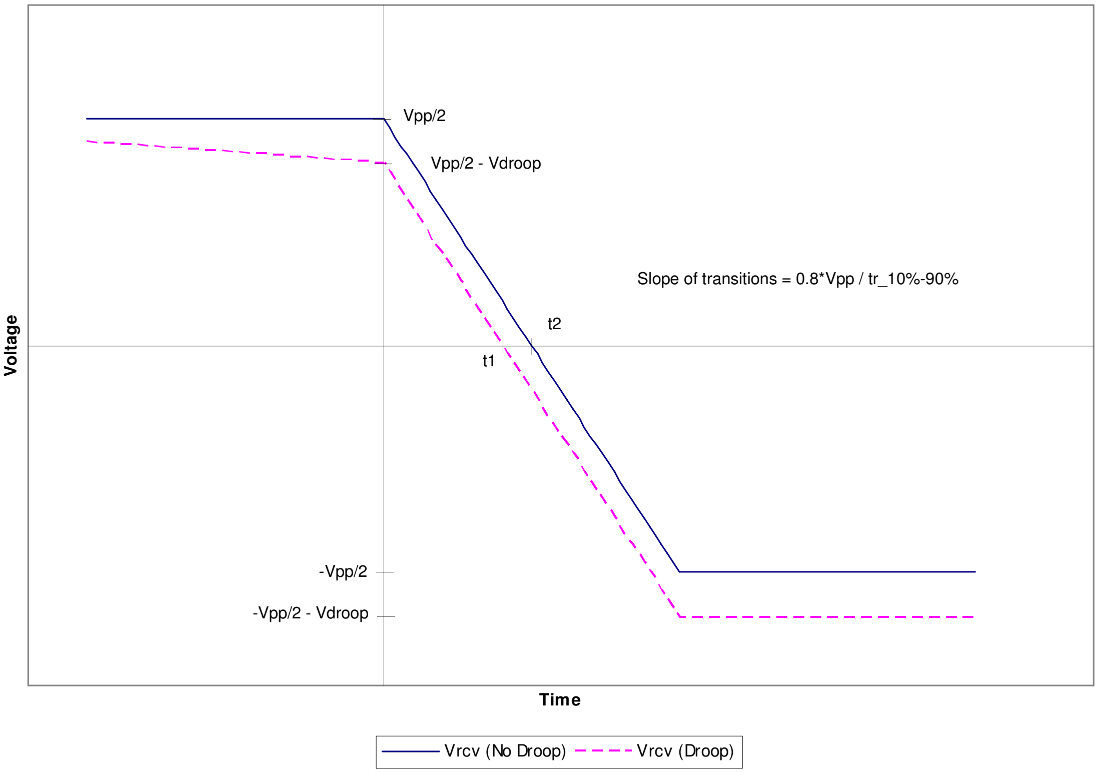


Figure 10 - Deterministic jitter from AC coupling.

The difference between $t1$ and $t2$ in Figure 10 is the peak-to-peak magnitude of the deterministic jitter and can be approximated by (5), where

$$t2 - t1 = (1.25)\frac{Vdroop}{Vpp}tr\ . \tag{5}$$

From (5), with a maximum voltage droop of 0.5% of the peak voltage, or 1% of the peak-to-peak voltage, and a rise time of 130 ps, corresponding to the maximum rise time of the Altera and Xilinx transceivers, a peak deterministic jitter of 1.625 ps results. This is very small (corresponding to 0.00508 UI at 3.125 Gbps), so the 150-bit maximum run length that would cause a voltage droop producing this amount of jitter is sufficiently small. The overall MRL limitation is imposed by the first aspect discussed above, and is 75 bits.

3.4 Probability of Bit Error

There are a few different ways to characterize the probability of a bit error in a serial digital multi-gigabit communication system. The first method is the traditional plot of bit error rate versus E_b / N_0, which assumes that all the noise in the channel is Gaussian. As discussed in section 3.2, the dominant noise sources are deterministic so an alternative approach is to plot the probability of a bit error versus E_b / N_0 with the deterministic disturbances considered. The use of both of these first two methods provides a good way to approximate the true transmission capacity of the channel. The final method is the one used within communication standards such as Fibre Channel in which the probability of a bit error is plotted versus jitter [26].

The current standards for serial digital multi-gigabit communication systems use 8B/10B coding for maximum run length limiting and CRC coding for error detection.

3.4.1 Bit Error Rate versus Gaussian Noise

An important measure of performance of a digital communication system is the probability of a bit error or the bit error rate. For different types of modulation and coding schemes, the probability of a bit error can be plotted versus the signal to noise ratio for Gaussian noise channels [22]. The signal to noise ratio in these plots is typically expressed as E_b/N_0, as used in (2). The required bit error rate for all of the serial standards reviewed in section 1.2 is 1.0e-12. The modulation and signaling type used in the serial digital multi-gigabit communication systems under consideration in this project is differential-voltage baseband PCM with the NRZ-L signal format. An expression for the bit error rate versus *E_b/N₀* for this signaling is given in (6). In (6), *Q*(*x*) and *erfc*(*x*) are two forms of the complementary error function used to compute the area under the tail of a Gaussian function [22]. Thus,

$$P_B = Q\left(\sqrt{2\frac{E_b}{N_0}}\right) = \frac{1}{2}erfc\left(\frac{\sqrt{2\frac{E_b}{N_0}}}{\sqrt{2}}\right). \qquad \textbf{(6)}$$

Using (6), a plot of the probability of a bit error versus E_b/N_0 has been plotted in Figure 11. In this figure, the value of E_b/N_0 corresponding to a bit error rate of 1.0e-12 is shown to be 13.9 dB.

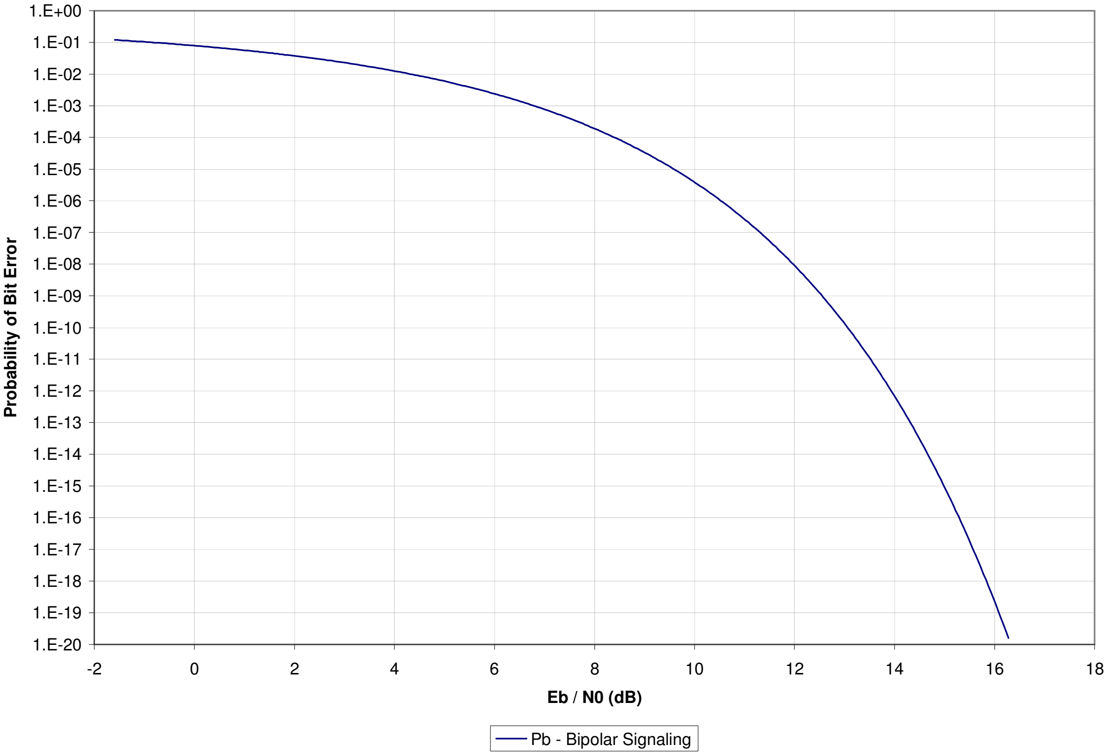


Figure 11 – Probability of bit error for Gaussian noise channel.

3.4.2 Bit Error Rate versus Gaussian Noise with Deterministic Noise

The largest noise sources in serial digital multi-gigabit communication systems are not Gaussian but are deterministic as pointed out in section 3.2.3. The Gaussian noise in a serial digital multi-gigabit communication system operating at 3.125 Gbps, corresponding to a bit error rate of 1.0e-12, was shown in [30] to be much lower than that predicted by (6). In [30], Ahmad and Cain analyzed the performance of the communication channel and showed the noise due to intersymbol interference and crosstalk were the largest noise sources. A plot of the probability of bit error was plotted versus Gaussian E_b / N_0 assuming the added distortion due to intersymbol interference was present. The result was a curve of very similar shape to Figure 11 but shifted to the right significantly. At a bit error rate of 1.0e-12 the resulting value of E_b / N_0 was about 28 dB. Another thing shown in [30] was that when intersymbol interference and random jitter of the receiver sampling clock are both considered, there is a floor to how low the probability of a bit error can go, and increasing E_b / N_0 does not further reduce the probability of bit error. This is a similar effect to that discussed by Sklar in [22] about how intersymbol interference distorts the received signal in a communication system and creates a lower limit to the achievable bit error rate. The largest floor shown in [30] occurred at a bit error rate of around 1.0e-39. This provides confidence that serial digital multi-gigabit communication systems operating at rates up to 3.125 Gbps and typically requiring a bit error rate of approximately 1.0e-12 are not operating near the bit error rate floor, and that error correction coding can be an effective means to lower the bit error rate in such systems.

3.4.3 Transmission Capacity of the Channel

Using the two values of bandwidth, *W*, determined from the SPICE simulation described in section 3.1.1, for a 40-inch backplane transmission channel, (3) was used to plot the capacity of the channel, *C*, as a function of E_b / N_0, and the results are shown in Figure 12. Assuming that a backplane communication channel operates with E_b / N_0 high enough to achieve the target bit error rate of 1.0e-12 without the use of coding, the channel capacity of the backplane communication channel (with the use of coding) lies somewhere between the two plots of capacity versus E_b / N_0 on Figure 12, at that operating value of E_b / N_0. The value of E_b / N_0 needed to achieve the target bit error rate of 1.0e-12 without coding will be between 13.9 dB and 28 dB, as determined in the previous two sections. The value of 13.9 dB is needed by the channel without coding to achieve a target bit error rate of 1.0e-12 for a Gaussian noise channel. The value of 28 dB is needed by the channel without coding to achieve a target bit error rate of 1.0e-12 for an intersymbol interference dominated channel. When deterministic noise is present, if all of the deterministic noise could be eliminated through equalization filtering, advanced crosstalk cancellation techniques, or any other methods making use of the deterministic nature of the noise, then the capacity of the channel at an E_b / N_0 of 28 dB could in theory be reached. In reality it is not possible to eliminate all of the deterministic noise. If on the other hand the channel is operating with only random Gaussian noise, which Shannon showed is the worst-case type of noise in [31], then the capacity would be at least that corresponding to E_b / N_0 of 13.9 dB. In Figure 12, the maximum transmission rate of 3.125 Gbps supported by both the Xilinx Virtex II Pro and the Altera Stratix GX is shown on the graph at E_b / N_0 of 13.9 dB and 28 dB.

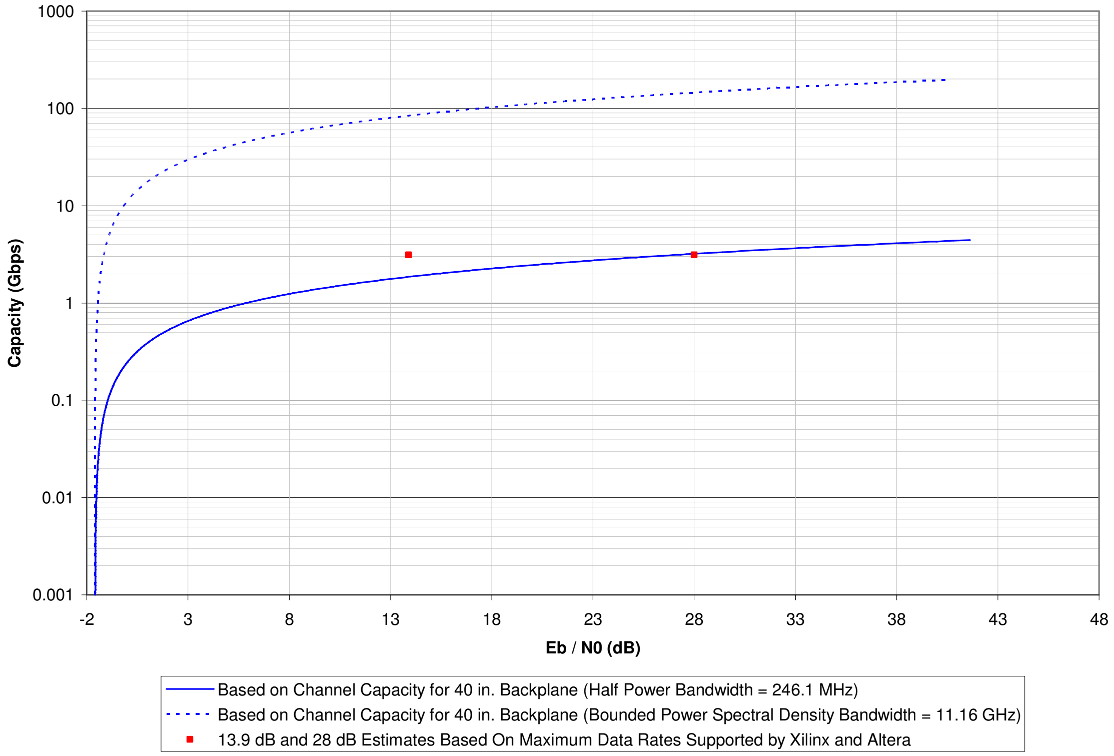


Figure 12 - 40-inch backplane channel capacity.

Table 5 shows the values of capacity from Figure 12 corresponding to 13.9 dB and 28 dB. The serial digital multi-gigabit communication systems implemented using Xilinx and Altera devices are operating quite a bit below the channel capacity defined by the bounded power spectral density bandwidth. Since reliable communication can occur without coding at the maximum transmission rate for Xilinx and Altera devices and even the next generation of devices supporting 10 Gbps communication will still be well below the capacity, it is unlikely that coding is necessary to allow data transmission to occur in serial digital multi-gigabit communication systems. The goal of coding in these systems is therefore to lower the bit error rate.

Table 5 - Channel capacity.

Channel Type	Eb / N0	Half Power Bandwidth	Bounded Power Spectral Density Bandwidth	Capacity Based on Half Power Bandwidth	Capacity Based on Bounded Power Spectral Density Bandwidth
10 in. Backplane	13.9 dB	703.0 MHz	11.27 GHz	5.30 Gbps	85.0 Gbps
	28 dB			9.14 Gbps	146.5 Gbps
40 in. Backplane	13.9 dB	246.1 MHz	11.16 GHz	1.86 Gbps	84.1 Gbps
	28 dB			3.20 Gbps	145.1 Gbps

3.4.4 Bit Error Rate versus Jitter

In most of the current serial digital multi-gigabit communication system standards described in section 1.2, the bit error rate is specified in terms of jitter instead of in terms of E_b / N_0. Jitter is classified as either random or deterministic as previously discussed. Deterministic jitter is bounded and is characterized by a peak-to-peak value. Random jitter is unbounded and is characterized by an rms value. This rms value is equivalent to the standard deviation of the gaussian distribution of the jitter. A given number of standard deviations corresponds to a bit error rate, so a peak-to-peak value of random jitter that is sometimes referred to is the rms value multiplied by a number of standard deviations that corresponds to a specific bit error rate. The definition of bit error rate versus jitter described in this section is based on *Information Technology - Fibre Channel - Methodologies for Jitter Specification* [26].

If only random jitter is present, the bit edge position can be plotted as a Gaussian probability density function (PDF) versus the time at which the edge occurs. The time value is usually normalized to the unit interval (UI). A value of 1 UI corresponds to the bit period. The mathematical representation of this Gaussian PDF is given as (7) [26], where

$$JT(t) = \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sigma} \cdot e^{\frac{-t^2}{2 \cdot \sigma^2}} . \qquad (7)$$

In (7), *t* is the jitter time and *σ* is the standard deviation of the random jitter, both in units of UI. The function is plotted, centered at 0 UI for the leading edge of a bit, and centered at 1 UI for the trailing edge of a bit.

Receiver jitter tolerance is defined as the time window centered about the middle of the bit period, such that if both leading-edge and trailing-edge transitions occur outside of the window, the receiver will make the correct decision as to whether the bit is a 1 or a 0. If either the leading-edge or trailing-edge transition occurs within the window, then it is assumed that the receiver will make the wrong decision and a bit error will occur. The probability of a bit error is the area under the tails of the PDFs from (7) that fall beyond jitter tolerance thresholds as shown in Figure 13 times the likelihood that a transition occurs. The likelihood of a transition, or transition density (*TD, referred to as α*), is usually assumed to be 0.5 meaning that half of the bits result from a transition. The expression for the total bit error rate is given by (8) [26], where

$$Pb = \alpha \cdot \int_{Lth\_l}^{\infty} JT(\tau)d\tau + \alpha \cdot \int_{-\infty}^{Rth\_l} JT(\tau - 1UI)d\tau . \qquad (8)$$

In (8), *α* is the transition density for the data, *JT* is given by (7), Lth_l is the left side receiver jitter threshold in units of UI, and Rth_l is the right side receiver jitter threshold in units of UI. The receiver jitter tolerance is usually specified in terms of the maximum amount of tolerable jitter that results in the error window however. For example, the Altera and Xilinx devices specify a receiver jitter tolerance of 0.65 UI and this corresponds to a window opening of (*Rth_l* – *Lth_l*) = 0.35 UI. The average value of *Lth_l* and *Rth_l* is equal to 0.5 for any system where the jitter PDF is an even function, so in this example, *Lth_l* would be 0.325 UI and *Rth_l* would be 0.675.

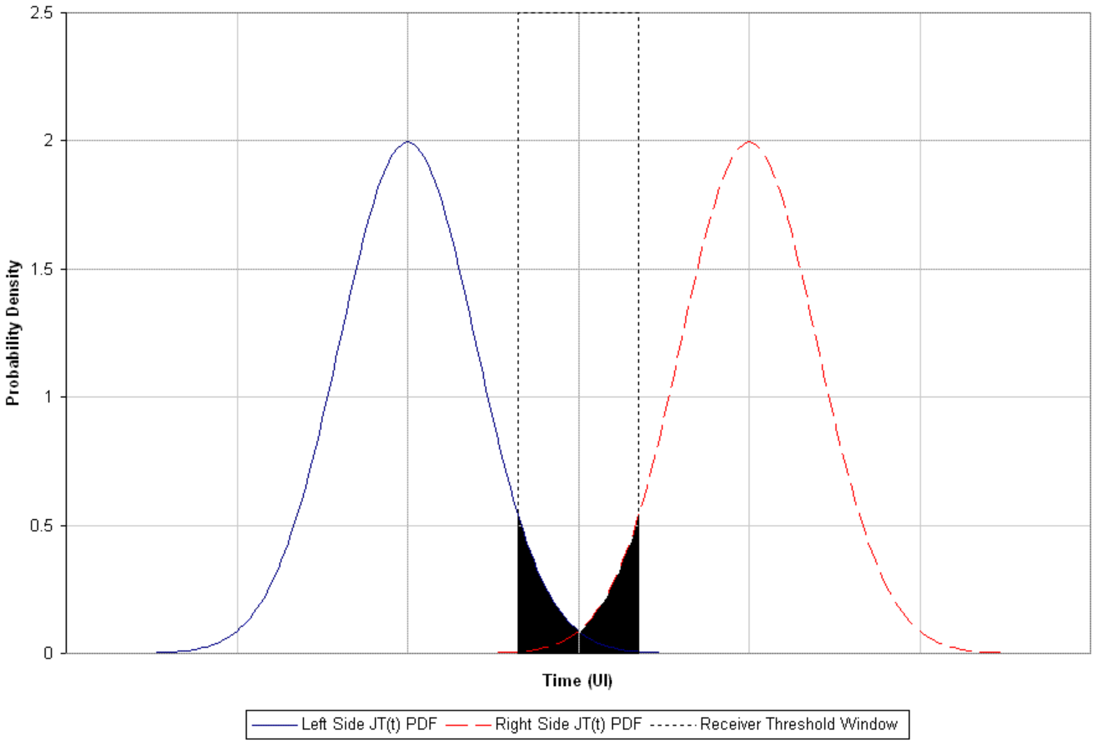


Figure 13 - Gaussian random jitter probability density function at each bit edge.

One other thing to note is that for a receiver jitter tolerance of 0.65 UI, the maximum allowed peak-to-peak jitter is 0.65 UI. At a given bit error rate, this total jitter budget can be divided between deterministic and random jitter and both values would be referred to as peak-to-peak values. However, the random jitter is unbounded, so it does not have a peak-to-peak value, but the allocation of the total peak-to-peak jitter budget to random jitter can be converted into a maximum allowed rms jitter for any given bit error rate as a design criteria for a system. For a bit error rate of 1.0e-12 the value of peak-to-peak random jitter is 14 times the rms jitter. The rms jitter is the same as the standard deviation from (7).

A worst-case model for the deterministic jitter that is to be added to the random jitter assumes that the transition edge is equally likely to occur at either of the two peaks of the peak-to-peak deterministic jitter value. The PDF of deterministic jitter would then consist of two impulse functions, each of weight 0.5, one at each of the two peak values. The combined PDF of the random and deterministic jitter is the convolution of the random jitter PDF from (7) and the deterministic jitter PDF. The resulting expression is given by (9) [26], where

$$JT(t) = \frac{1}{2\cdot\sqrt{2\pi}}\cdot\frac{1}{\sigma}\cdot\left(e^{\frac{-\left(t-\frac{W}{2}\right)^2}{2\cdot\sigma^2}} + e^{\frac{-\left(t+\frac{W}{2}\right)^2}{2\cdot\sigma^2}} \right). \qquad (9)$$

In (9), *t* is the jitter time, *W* is the peak-to-peak magnitude of the deterministic jitter, and σ is the standard deviation of the random jitter, all in units of UI. A plot of the PDF for the case of the jitter requirements of the XAUI standard is shown in Figure 14. The XAUI standard specifies a maximum total jitter of 0.65 UI at the receiver, with 0.47 UI being the peak-to-peak deterministic jitter and 0.18 UI being the specified maximum peak-to-peak random jitter. At a bit error rate of 1.0e-12, 0.18 UI of peak-to-peak random jitter corresponds to a value of σ equal to 0.0129 UI.

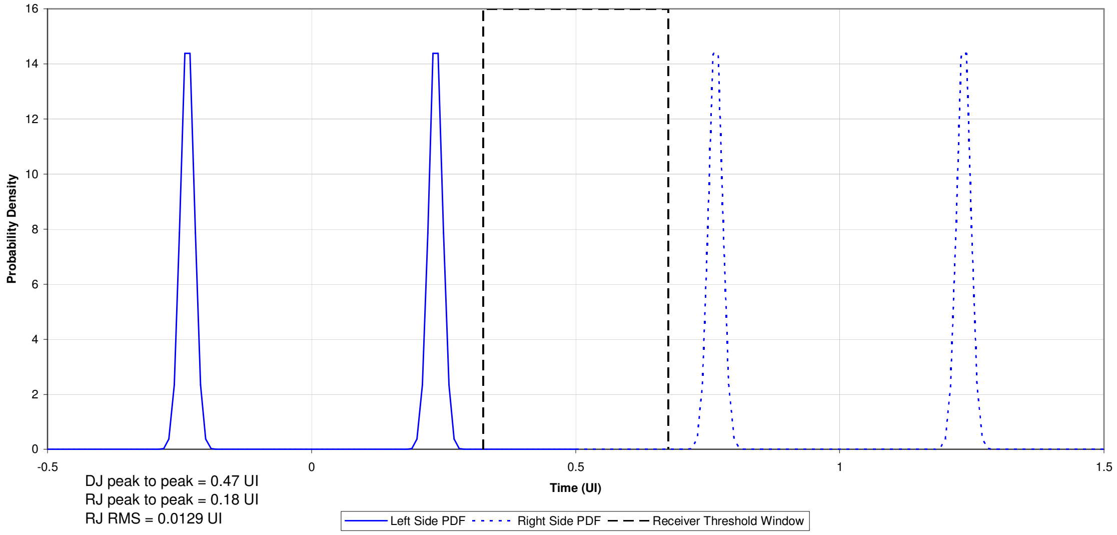


Figure 14 - XAUI PDF of bit transition times.

The areas under the PDFs for the left and right transitions were computed numerically from the sample point to infinity for the left PDF of Figure 14 and from negative infinity to the sample point for the right PDF of Figure 14 as indicated by (8). Normally in statistics, the area under each PDF from negative infinity to infinity is equal to 1 since the event must occur somewhere. For this analysis, though, when determining the probability of a bit error, each integrated PDF is multiplied by the transition density which is assumed to be 0.5. The sum of the areas under the two PDFs shown in Figure 14 (each one including two Gaussian-shaped sections), across the entire PDFs, would be 2, but since each is multiplied by the transition density of 0.5 the sum is 1. The overall probability of bit errors for a given sampling point is the sum of the two areas under the PDFs, over the appropriate intervals, times the transition density of 0.5. One important assumption of this mathematical model is that transitions from neighboring bit positions do not contribute to the probability of a bit error. The total bit error rate is plotted in

Figure 15 and this type of plot is typically referred to as a bathtub curve because the shape resembles a bathtub.

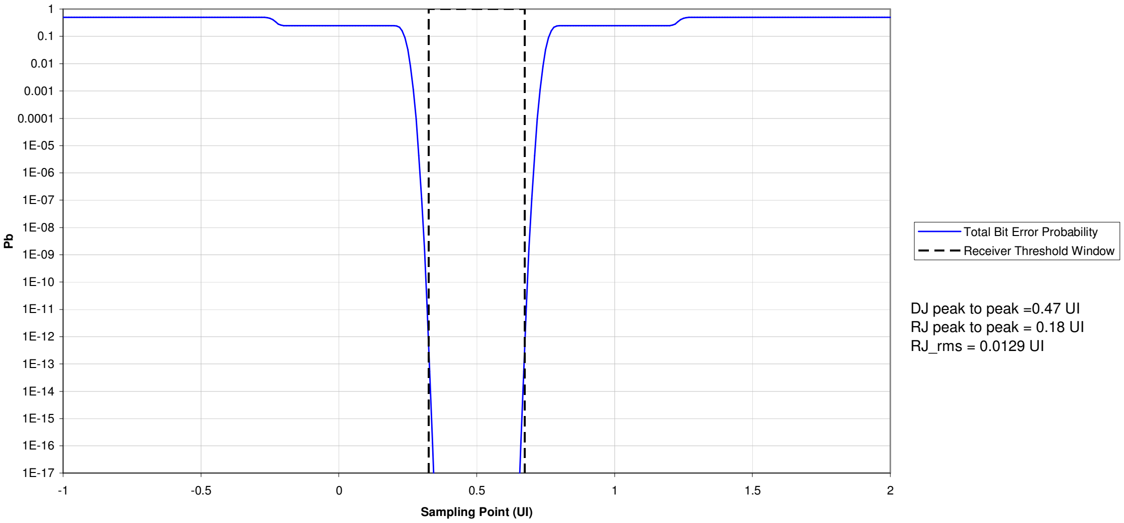


Figure 15 - Probability of bit error plot for XAUI.

To evaluate whether the error correction code employed is effective at reducing the bit error rate, a plot such as in Figure 15 will be made for the uncoded and coded systems for comparison purposes.

There are some possible drawbacks to the jitter method of specifying the bit error rate. The first is that it assumes that bit errors only occur on bits that are the result of a transition or have a transition as the next bit. There must also be some probability in NRZ-L for a bit error to occur for a bit with no transitions on either side of it. This probability must be assumed to be small enough to not matter significantly. The next drawback is that the transition density of the data is assumed to be 0.5 and this will vary in real data. Another drawback is that the method assumes an unrealistic distribution of deterministic jitter. The real distribution will be continuous over the range of the peak-to-peak deterministic jitter. The distribution assumed is a good worst-case approximation and it is likely that the true distribution would lead to a lower bit error rate.

Finally, the bit error rate analysis based on jitter assumes that if a transition occurs within the receiver threshold window that it will cause a bit error. In reality, it probably depends on where in the threshold window it occurs as to whether or not a bit error is made and there will be a probability density function for whether or not a bit error is made versus the transition location within the receiver threshold window. Despite these drawbacks this method should provide an upper bound on the probability of a bit error, provides design criteria for deterministic jitter, and provides a means to compare coded and uncoded systems. In reality, the systems specified with a bit error rate of 1.0e-12 will likely be operating at a lower bit error rate because of these drawbacks. This may present a problem for measuring the bit error rate due to the large amount of time needed to count enough bit errors for a valid measurement.

3.5 Error Correction Code Selection

The goal of the error correction code design for serial digital multi-gigabit communication systems in this project is to lower the bit error rate for a given data rate and specified total jitter, or allow the same bit error rate but with more deterministic jitter. This section contains a summary of the specific code requirements as previously discussed and a discussion about the choice of error correction code to be implemented.

3.5.1 Summary of ECC Requirements

The requirements of the error correction code (ECC) and the parameters of the communication channel, the transmitter, and the receiver are summarized in Table 6. The parameters of the channel, transmitter, and receiver, given in Table 6, are based on the XAUI specification and the Altera and Xilinx FPGA device capabilities. The requirements have been established based on the discussion in sections 3.1, 3.2, 3.3, and 3.4. The ECC designed in this project will be based on these parameters and requirements.

Table 6 - ECC design criteria.

	Description	Value
Parameters	Transmitter rms random jitter	0.18 UI / 14 = 0.0129 UI
	Deterministic jitter at 3.125 Gbps	0.47 UI
	Receiver jitter tolerance	0.65 UI
	Coded data block size	Multiple of 8 or 10 bits
ECC Requirements	Maximum data block size	128 bits
	Uncoded data block size	Multiple of 8 bits
	Maximum decoding latency	120 ns
	Minimum code rate	0.64
	Target code rate	0.79
	Maximum run length of constant 0 or 1 data	75 bits
	Error correcting capabilities	Random bit errors
	Target corrected bit error rate (at deterministic jitter specification)	1.0e-17
	Target corrected bit error rate (at 10% increase in deterministic jitter specification	1.0e-12

3.5.2 Maximum Run Length Limit

In order to limit the maximum run length of 0 bits or 1 bits, extra bits must be added such that transitions can be forced. When maximum run length (MRL) coding is used in conjunction with error correction coding, the performance of the error correction code (ECC) can be affected by the MRL code. This occurs if the MRL code occurs after the ECC on the transmitter side. The decoding of the MRL code at the receiver can multiply single bit errors into multiple bit errors. In the case of 8B/10B coding, a single bit error into the 8B/10B decoder could be multiplied into a multiple bit error out of the decoder, or cause a code word to be invalid, or cause a disparity error. In the latter two cases the 8B/10B decoder algorithm cannot even produce a valid output sequence and usually just signifies the appropriate error type to the next communication layer. If the MRL code occurs before the ECC at the transmitter, this problem is eliminated but a new problem is created. The problem is that the ECC can then upset the MRL characteristics imposed by the MRL code.

Regarding the problem with having the MRL code before the ECC in the transmitter, there are ways to construct the error correction output code word to limit the impact on the MRL code as discussed in [32]. These techniques work well with block error correction codes. A block diagram of the proposed coding for the transmission system is shown in Figure 16. The MRL code to be used in this ECC system will be designed so that it meets the required MRL constraints and matches the required data widths, that is block sizes, on the data source side and the error correction code side.

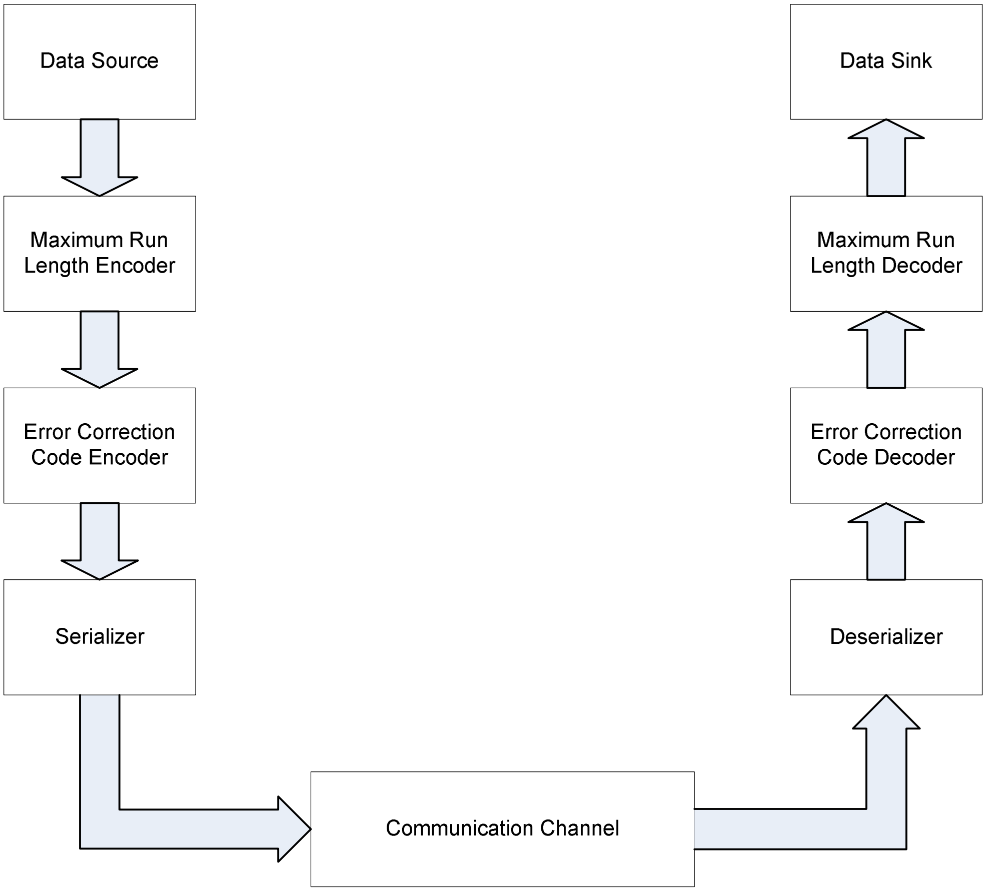

Data Source
Maximum Run Length Encoder
Error Correction Code Encoder
Serializer
Communication Channel
Deserializer
Error Correction Code Decoder
Maximum Run Length Decoder
Data Sink

Figure 16 - MRL and ECC system design block diagram.

3.5.3 Discussion of Code Choice

The large number of error correction codes available from research going on since Shannon presented his first work on digital communications and information theory makes code selection difficult for any new application. Generally error correction codes fall into two main categories, block codes and convolutional codes. There are also advanced coding schemes that use concatenated codes and iterative decoding such as turbo codes. The following discussion on selecting a code to use in a serial digital multi-gigabit communication system begins by narrowing down the choices, and then focusing on the specific type of codes that will be most applicable and making a selection from them.

Turbo codes were developed in 1993 by Berrou, Glavieux, and Thitimajshima [22]. The specific code demonstrated had a rate of 0.5 and was able to provide a bit error rate of 1.0e-5 at E_b / N_0 of 0.7 dB using binary phase shift keying (BPSK) modulation in a random error channel. A convolutional turbo code contains two convolutional codes and an interleaver / deinterleaver, and the decoder operates by iteratively going through two decoding stages, using soft inputs and soft outputs [22]. At high signal-to-noise ratios, the decoded bit error rate for this type of code has been shown to level off at a relatively low bit error rate because of the low hamming distance, and increasing the signal-to-noise ratio does not further reduce the probability of bit error. Therefore, the code performance is relatively poor at very low bit error rates [33]. Turbo codes based on concatenated convolutional codes are not suitable for this application for many reasons. First, there is no soft decoding information available from the transceivers in the Altera and Xilinx devices. The code rate of 0.5 is too low and the decoding latency required for convolutional decoding, plus interleaving and deinterleaving, and then iteration of this process would be too long. In addition, the complexity would make it very difficult to implement in an FPGA processing bits at multi-gigabit rates. Also, this application is operating at bit error rates much lower than where this code performs well. Convolutional turbo codes are thus ruled out.

Another form of turbo coding described in [34] is to perform iterative decoding on two block codes concatenated together. This has been shown to have good performance at low bit error rates and can use a higher code rate. The complexity for implementation is probably beyond the capabilities of an FPGA for multi-gigabit data rates, but more detailed analysis would be needed to determine this for sure. The main problem would still be the latency introduced by the iterative decoding. This rules out block turbo codes.

Convolutional codes have been used extensively for moderate bit error rate and low E_b / N_0 applications such as deep space and satellite communications [33]. They are different from block codes in that there are not discrete output code blocks that are independent of the blocks on either side of them. Output code words depend on the present and previous input code words. Every output code word consists of n bits and every input code word consists of k bits, which is similar to a block code, but one additional parameter defining the number of previous input code words an output depends on is called the constraint length v. The performance of the code depends on the constraint length v and increases as v increases. Also, convolutional codes can be implemented using hard decisions or soft decisions with a performance increase possible by using soft decisions [22]. The performance of convolutional codes at very low bit error rates is not widely published. The sources available seem to consider them in the context of moderate bit error rate and high noise applications. The rates commonly discussed are 0.5 or less. Convolutional codes having rates above 0.75 are uncommon, and higher rate convolutional codes appear to be difficult to implement due to decoding complexity. One exception is the use of puncturing, where a lower rate parent code is used but specific code bits are punctured or removed prior to transmission, thus resulting in a higher rate code. Punctured codes allow for higher rate convolutional codes while using a lower rate decoder because the punctured bit locations are filled in at the receiver with what are called erasure bits that effectively have a level midway between logic 0 and 1 when soft decisions are used [35]. These aspects of convolutional codes do not make them ideal candidates for the serial digital multi-gigabit communication system.

There are many different ways to decode convolutional codes including the maximum likelihood Viterbi algorithm, sequential decoding, and majority-logic decoding [35]. Viterbi decoding is optimal but complexity increases exponentially with the constraint length *v* and there are practical limits on how large *v* can be in real implementations [35]. The number of computations required per bit for Viterbi decoding is 2^v. The decoding delay for a Viterbi decoder can also be long and is $h + m$ where *m* is the memory order of the code and *h* is the length of a frame usually assumed to be much larger than *m* [35]. Each input bit to a convolutional code is shifted through a number of memory stages and each output bit from a convolutional code is a logical combination of the contents of the memory stages after each shift. The longest number of shift stages for any of the bits is referred to as the memory order, *m* and the sum of the shift stages for all the bits is referred to as the constraint length *v* [35]. For an infinite length data stream, valid data bits will be transmitted out of a convolutional encoder once all of the shift stages are loaded and will continue forever. For a finite length data stream, data must be formatted into frames to be transmitted and extra padding bits must be shifted into the encoder after the valid data bits so that the code bits corresponding to those last data bits are shifted out. It is desirable to limit the amount of padding bits in relation to the frame size so the amount of overhead that they add to the code rate, referred to as fractional loss, is small. This is what is meant by the frame size *h* [35]. The number of computations required for multi-gigabit data rates would be much too large for an FPGA implementation using a reasonable constraint length to achieve the performance necessary. Also, the decoding delay is too long and therefore Viterbi decoding of convolutional codes is ruled out.

Sequential decoding of convolutional codes is less than optimal but since its complexity depends on the noise level in the received data and not *v*, large constraint lengths and thus very good performances can be realized. The decoding complexity is variable depending on the noise in the received sequence and averages about 1 to 2 computations per bit [35]. A drawback is that for noisy received sequences the increased number of computations and thus time taken to decode the information can cause buffer overflows and data could be lost in practical implementations. While the processing power required for sequential decoding is less than Viterbi, more memory is usually required. The decoding delay for a sequential decoder is the same as that for a Viterbi decoder [35]. The processing complexity is still too large to implement at multi-gigabit rates, the memory requirements are too great, and the decoding delay is too long. Sequential decoding of convolutional codes is ruled out.

Majority logic decoding of convolutional codes is much simpler than Viterbi or sequential decoding but the simplification comes at the expense of code performance. Only one computation is required per bit. The decoding delay is also much smaller than that for the Viterbi or sequential decoding and is equal to the memory order *m*. For applications requiring a large minimum distance to achieve the desired bit error rate performance, very long constraint lengths are required and Viterbi or sequential decoders may provide better performance to complexity trade offs [35]. It may be possible to implement a majority logic decoder with convolutional coding in an FPGA implementation at multi-gigabit data rates and meet the latency requirements. The performance at low bit error rates as previously mentioned is not widely published. Because of the unknown performance of convolutional codes at low bit error rates, and potential difficulties in implementing even the simpler majority logic decoder in an FPGA, block codes are expected to provide a better solution for the serial digital multi-gigabit communication system.

Linear block codes are codes that systematically transform a block of *k* bits into a larger block of *n* bits and thus have a code rate of *k*/*n*. The term block length is used to describe *n,* the number of bits in a coded block for a block code. The meaning of linear is that the result of adding any two code words together with modulo-two additions is also a code word. Linear block codes vary widely in block size, complexity, decoding latency, and bit error correction and detection performance. They can also be applied effectively to random error channels, burst error channels, or combinations of the two [35]. Block codes have been applied extensively in the area of data storage applications [33]. The interesting thing about data storage systems, especially SDRAM storage, is that they typically transfer data at data rates in the multi-gigabit range. For example a typical Double Data Rate (DDR) SDRAM interface can run at up to 400 Mbps per data bit or, for a common 64 bit wide interface, at 25.6 Gbps. The other similarity is in the block size since the typical SDRAM block size is 64 bits. The backplane communication channel is not very similar to the channel typically used in data storage applications so the criteria used to choose a block code is not the same. Some of the block codes used in the SDRAM storage application are Hamming codes, shortened versions of extended Hamming codes, and Bose-Chaudhuri-Hocquengham (BCH) or Reed Solomon codes

[33]. The specific codes used have error detection and correction capabilities that fall into one of the following classes:

- Single error correcting / double error detecting;
- Single error correcting / double error detecting / single b-bit byte error detecting;
- Single b-bit byte error correcting / double b-bit byte error detecting.

A single b-bit byte error-detecting code is one for which data is broken into b-bit wide bytes. Any number of errors within a single byte can be detected by the code. Similarly, a single b-bit byte error-correcting code has the ability to correct any number of bit errors within a single b-bit byte. These types of codes are useful in memory systems made up of multiple memory chips, each chip storing a certain number of bits at each address location. A failure of a single chip could be detected or corrected with one of these codes [33]. This leads to another similarity between SDRAM storage systems and serial digital multi-gigabit communication systems and that is the data size being a multiple of the typical 8-bit byte.

For this design project, the BCH code has been chosen. This type of code has relatively low complexity for decoding and allows for correcting random bit errors with a high code rate as required by the application [35]. Another characteristic of a cyclic code such as the BCH code is that it can be shortened so that a code that matches the data length constraints on both the MRL and the SERDES sides (see Figure 16) can be chosen without the need to add padding bits. Other codes that were considered were the Reed Solomon code and the simple Hamming code. The BCH code can provide some additional error correction power over a simpler Hamming code. The Reed Solomon code provides the ability to correct burst errors and byte errors, but these types of errors are not anticipated in the backplane communication channel and the Reed Soloman code would therefore have more capabilities than needed.

3.6 Error Correction Code Design

The code chosen consists of a 48b/51b MRL code and a two-error correcting primitive BCH code of order 2^6. In this context, order is defined as the number of elements in a finite field. A field is a set of elements and for the binary case the elements are 1s and 0s. Some properties of fields are that it is possible to define addition, subtraction, multiplication, and division operations that satisfy commutative, associative, and distributive laws [36]. Such a BCH code consists of 63 total bits with 51 data bits and 12 parity bits. The block length of 63 is obtained from $2^6 - 1$, and the minimum distance of the code is given by (10) where *t* is the number of errors that can be corrected [35]:

$$d_{\min} \geq 2t + 1. \qquad \textbf{(10)}$$

The minimum distance for the (63,51) BCH code is therefore 5. A single padding bit is added to the code word to make it 64 bits, which is a multiple of 16 bits, for easy connection to the Altera Stratix GX transceiver. This padding bit could be used as a synchronization bit in a system design, and the implementation of the BCH code in this project will toggle this bit between 1 and 0 for each successive code block.

The data will be encoded and decoded in parallel logic at a much slower clock rate than the serial data transfer clock rate. The output data from the encoder is 64 bits so it can be seen that the serial data clock rate is 64 times the block encoding and decoding logic clock rate. Since all of the data, MRL, and parity bits are available at the same time in the slower parallel clock domain, their ordering for actual serial transmission is not critical. The ordering has been chosen to maximize the number of chances for transitions to occur in most data sequences by dispersing the MRL code bits and the ECC code parity bits throughout the code word as shown in Figure 17.

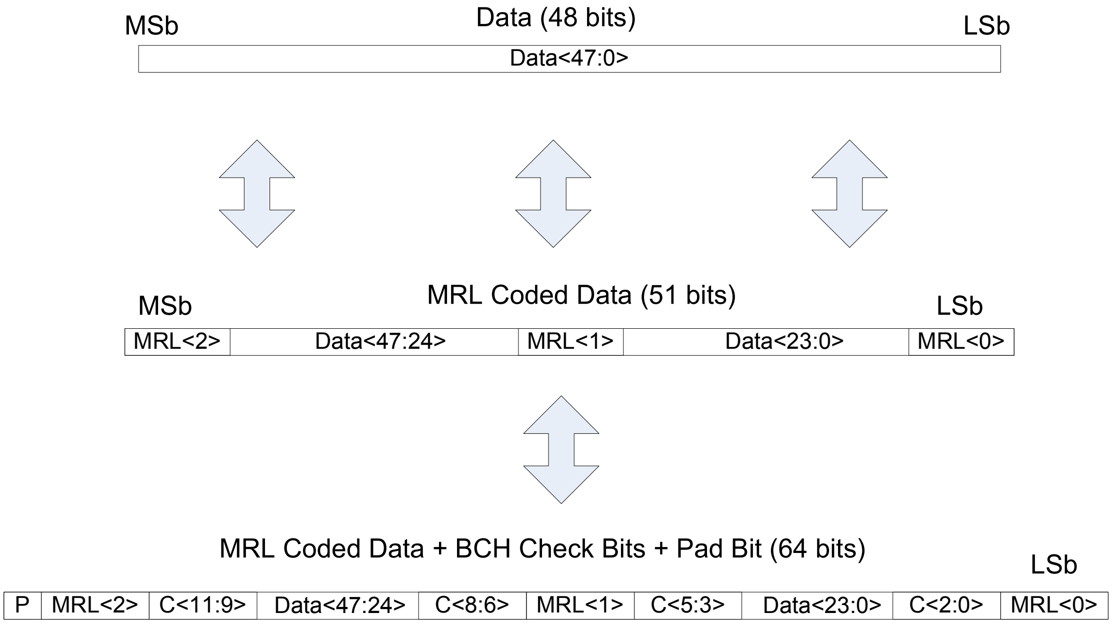


Figure 17 - Code bit order diagram.

3.6.1 MRL + BCH (63,51) Encoder Design

The encoder consists of two steps which are the 48b/51b MRL encoder block and the BCH encoder block. Each step takes a single parallel clock cycle for a total encoding latency of 2 parallel clock cycles. A block diagram of the encoder is shown in Figure 18.

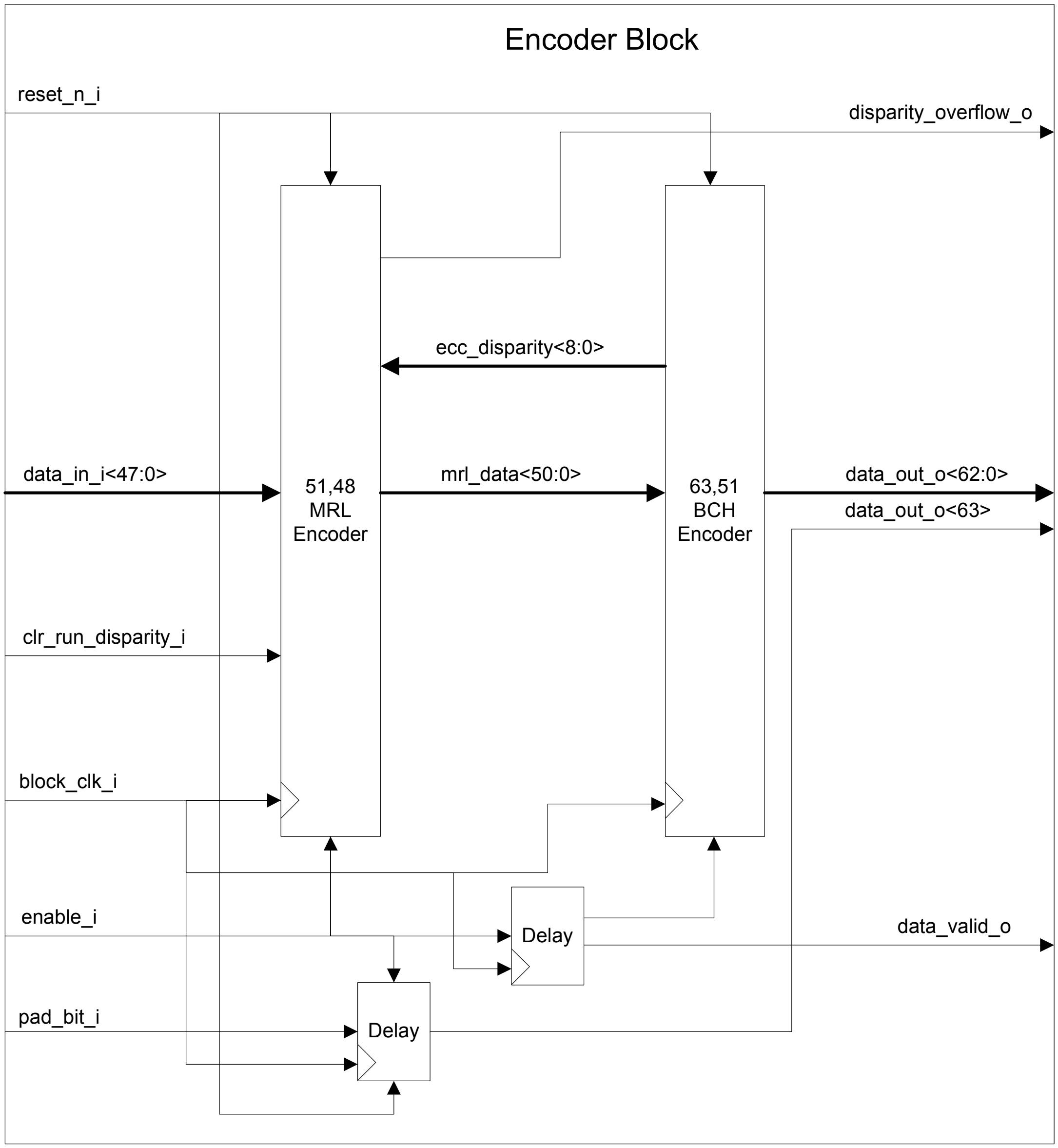


Figure 18 - Encoder block diagram.

All of the inputs and outputs from the block are described in the datasheet in section 9, which is Appendix A – ECC Block Datasheet.

The 48b/51b MRL encoder is based on [37] and contains a running disparity register keeping track of the difference between the number of 1 bits and 0 bits that have been output by the encoder. The running disparity is based on the 51 output bits of the MRL encoder and on the disparity of the 12 ECC parity bits, which is fed back to the

MRL encoder from the BCH encoder. The running disparity is initially set to 0 and the MRL encoder either inverts or leaves alone the 48 input data bits and sets the other three MRL bits based on the following rules, which are repeated for each 48-bit block of data received by the encoder.

- If the running disparity is greater than or equal to 0
    - If the disparity of the current 48 bit input data word is greater than or equal to 0, then invert the data bits and set MRL bits 0 and 2 to 0
    - Else leave the 48 bit input data word alone and set MRL bits 0 and 2 to 1
- Else if the running disparity is less than 0
    - If the disparity of the current 48 bit input data word is greater than or equal to 0, then leave the data bits alone and set MRL bits 0 and 2 to 1
    - Else invert the data bits and set MRL bits 0 and 2 to 0
- Set MRL bit 1 to the inverse of MRL bit 2

The code from [37] is slightly different and works with only two MRL bits instead of three. The maximum run length of the code from [37] is given by (11) and the maximum running disparity is given by (12) where X is the length of the input data word in bits [37]. Thus,

$$\text{Maximum Run Length} = 2.5X + 2 \qquad (11)$$

and

$$-(1.5X + 2) \leq \text{Running Disparity} \leq 1.5X + 2\,. \qquad (12)$$

Using the third MRL bit limits the maximum run length of the code designed here to only 51 bits (prior to FEC); however, it does so at the expense of losing the guarantee of a bound on the running disparity. Because the ECC encoder follows the MRL encoder, and the ECC encoder adds an additional 12 parity bits that are never inverted by the MRL encoder, there is inherently no bound to the maximum running disparity in this design. The MRL design provides a mechanism for limiting the maximum run length of the output code word to the coded word length of 64 bits which meets the requirement of the design. It also provides some ability to limit the DC spectral content of the code words even though some data patterns could be passed through with such content. A system requiring a limit on the DC spectral content could use the disparity overflow output bit from the encoder block to trigger the transmission of some all 0 or all 1 data words that would allow the encoder to bring the running disparity back down. However, this would have to be handled outside of the encoder block in this design. The MRL block that has been designed has the advantages of being simple to implement, it provides some control of the DC spectral content of the data thus improving the ability to operate with AC coupling, and it provides a limit on the maximum run length that meets the requirement given in Table 6.

The encoder for a cyclic block code is based on dividing the polynomial representing the input data sequence by the generator polynomial. The coefficients of the remainder of this division are the parity bits. Equation (13) shows this operation, where *u(X)* is the message polynomial, *n* is the coded data size, *k* is the data size, *a(X)* is the quotient of the division, *g(X)* is the generator polynomial, and *b(X)* is the remainder of the division [35]. Thus,

$$X^{n-k}u(X) = a(X)g(X) + b(X)\,. \qquad (13)$$

The degree of *b(X)* is less than or equal to *n – k – 1* which results in a polynomial with *n – k* coefficients. The generator polynomial for the selected (63,51) BCH code is given in (14) [35], where

$$g(X) = X^{12} + X^{10} + X^{8} + X^{5} + X^{4} + X^{3} + 1\,. \qquad (14)$$

The remainder of the division can be computed in digital hardware using a feedback shift register with modulo-2 addition, having taps in the positions corresponding to the exponents of the terms in *g(X)*. The remainder is contained in the shift register bits after the entire 51-bit input data sequence *u(X),* labeled as D(50:0) in Figure 19, is shifted in, most significant bit (MSb) first. The circuit that performs this computation of the parity bits for the (63,51) BCH code is shown in Figure 19 [35]. The circuit in Figure 19 does not require logic 0 bits to be appended to the end of the data word to shift out the parity bits at the end. After 51 shifts, the circuit will simply contain the 12 parity bits in the registers b0 through b11. [35].

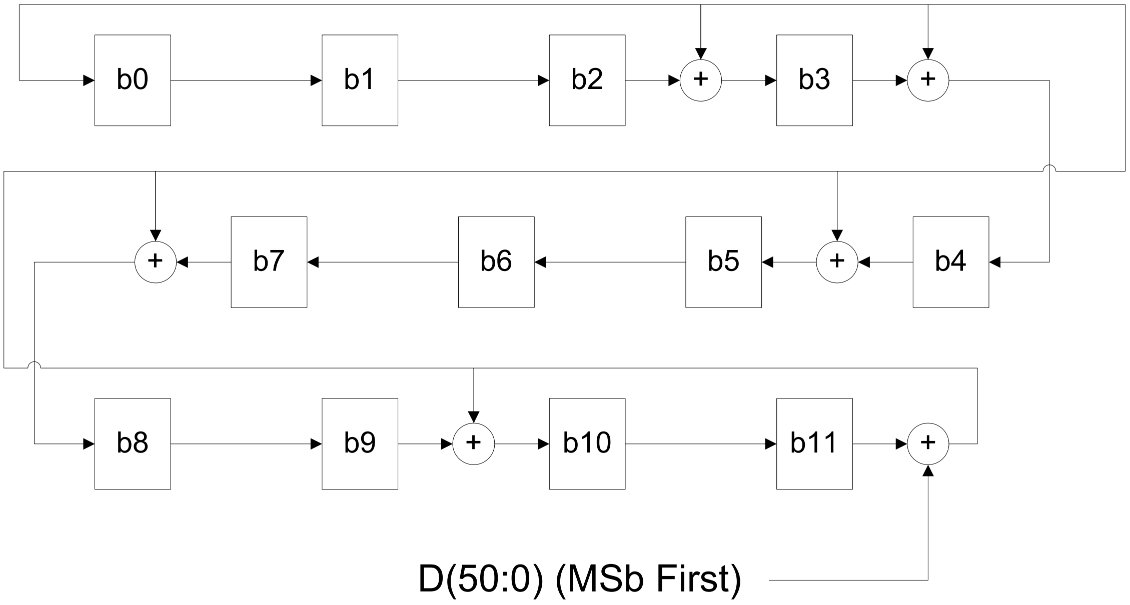


Figure 19 - (63,51) BCH code encoder circuit.

A direct hardware implementation of the circuit in Figure 19 would require that shift registers be constructed in hardware and data shifted at a clock frequency equal to the data rate of transmission. For multi-gigabit data transmission this is not possible in an FPGA. However, if all of the input data bits are available at once then the output of each memory cell within the encoder shown in Figure 19 can be computed as a modulo-2 sum of certain input data bits keeping in mind that every time the same input data bit gets added to itself, the sum is zero in modulo-2 addition. The expression for each of the 12 parity bits, or memory cell outputs for the encoder shown in Figure 19, was calculated symbolically using Mathcad. The resulting expressions for b8 and b9 each contained the largest number of modulo-2 additions at 31, and the resulting expression for b8 is shown in (15), where

$$\begin{aligned} b8 = & u_{15} + u_{42} + u_{18} + u_{35} + u_{32} + u_{12} + u_{24} + u_{39} + u_{4} + u_{31} \\ & + u_{49} + u_{27} + u_{33} + u_{28} + u_{10} + u_{17} + u_{19} + u_{11} + u_{45} + u_{40} \\ & + u_{38} + u_{46} + u_{30} + u_{41} + u_{47} + u_{14} + u_{26} + u_{0} + u_{2} + u_{8} + u_{3} . \end{aligned} \tag{15}$$

An equation of this form could be determined and implemented directly in hardware for each of the parity bits. VHDL provides a mechanism for generating combinational logic for multiple simultaneous shifts of a shift register such as this using a FOR GENERATE statement, and then the logic synthesis tool will automatically create the required equations such as in (15), so for purposes of implementation there is no need to determine these analytically.

3.6.2 General BCH Decoding Discussion

Encoding of the binary BCH codes is straightforward as has been described. Decoding is not as straightforward and there are several methods available for decoding binary BCH codes. Most of the algorithms involve three main steps as follows [35]:

- Compute the syndromes;
- Determine the error-location polynomial;
- Find the roots of the error-location polynomial as the error locations.

The syndrome computation is similar to the encoding process and is fairly straightforward. The syndrome computation is discussed in more detail in section 3.6.3. Determining the error-location polynomial coefficients is not as straightforward and there are at least three methods for doing so. The first is Peterson’s direct-solution decoding algorithm which involves the direct solution of the equations resulting from Newton’s identities [38]. The second is the Berlekamp algorithm which iteratively solves for the error-location polynomial coefficients [38]. The third method is Euclid's algorithm which recursively finds the greatest common divisor between two polynomials [38].

The Peterson direct-solution method is the simplest to implement for small values of *t*, but its complexity grows with the square of *t,* the number of errors that can be corrected, while the Berlekamp algorithm's complexity grows linearly with *t* [38]. The Berlekamp algorithm is the most efficient in general, but the difference between it and the Euclidean algorithm is not as great as between it and the Peterson direct-solution method [38]. The Peterson method is reasonable for decoders that correct up to 6 or 7 errors, while the Berlekamp and Euclidean algorithms are reasonable for decoders that correct many errors [38]. The Berlekamp algorithm is usually preferred to the Peterson method for decoders correcting more than 3 or 4 errors [36]. Another algorithm for decoding binary BCH codes uses frequency domain decoding which does not seem to be very straightforward [38]. The Peterson direct-solution method has been chosen in this design because of its efficiency and ease of implementation for a small t=2 error correcting code.

The error search is done by finding the roots of the error-location polynomial. The degree of the error-location polynomial is equal to the number of errors that the code can correct. A direct solution approach can be taken for BCH codes that correct 1 or 2 errors by using Galois field arithmetic to directly solve for the roots of the error-location polynomial [36]. The other method that is commonly used is to try all the elements in the Galois field of the code in the error-location polynomial and see if the result of the polynomial is 0. If it is, then that element must be a root. This systematic approach is called a Chien search [36]. The direct solution approach would work for the code of this design, but the Chien search is straightforward, easy to implement, and can be easily extended to codes in future designs with the ability to correct more than 2 errors.

BCH codes have the ability to correct more than *t* errors for certain error patterns. All of the decoding algorithms discussed, and the one implemented in this design, correct *t* errors. According to [36], complete decoding algorithms for all double- and some triple-error correcting BCH codes are available. The BCH code in this design will correct all two-error patterns and detect some error patterns of greater than two errors.

3.6.3 MRL + BCH (63,51) Decoder Design

The decoder consists of two main stages which are the BCH decoder block and the 48b/51b MRL decoder block. The MRL decoder takes a single clock cycle. The BCH decoder has three stages, the syndrome computation, the error-location polynomial determination, and the error-location computation each taking 1 clock cycle for a total of 3 clock cycles. The total decoding latency is 4 clock cycles. A block diagram of the decoder is shown in Figure 20. All of the inputs and outputs from the block are described in the datasheet in section 9, which is Appendix A – ECC Block Datasheet.

The syndromes S_i are computed by dividing the received message polynomial by the minimal polynomial $\varphi_i(X)$ evaluated at α^i [35]. A Galois field contains a set of minimal polynomials, one for every element in the field, where each minimal polynomial is the lowest degree polynomial that the evaluation of at that element results in the 0 element [36]. An equation describing this operation where $r(X)$ is the received message polynomial, $a_i(X)$ is the quotient of the division, and $b_i(X)$ is the remainder of the division is given in (16) [35], where

$$r(X) = a_i(X)\phi_i(X) + b_i(X)\,. \qquad (16)$$

For a BCH code there are 2*t* syndromes where *t* is the number of errors that can be corrected by the code. For the (63,51) BCH code, there are 4 syndromes [35].

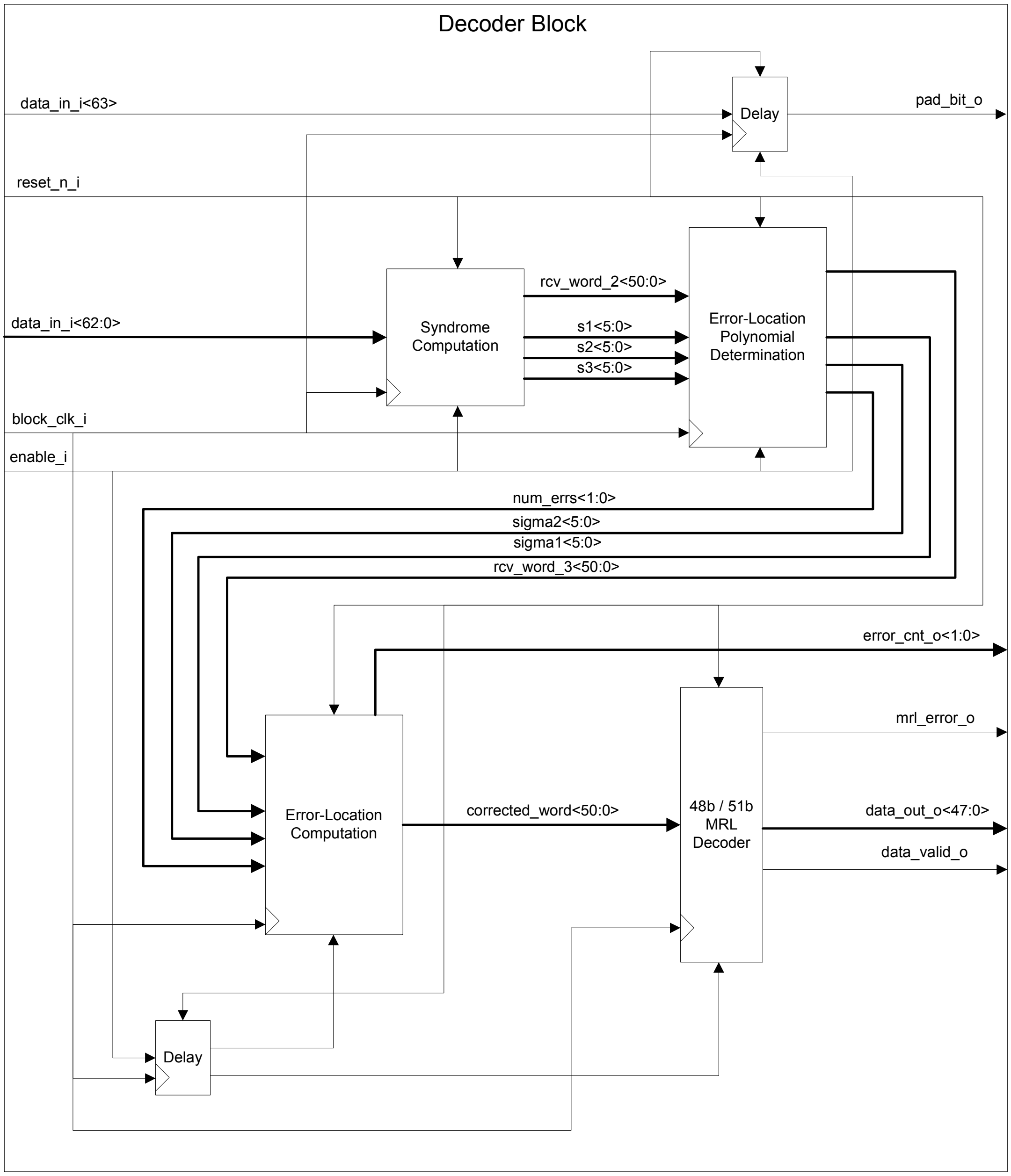


Figure 20 - Decoder block diagram.

For Galois field GF(2^6), the minimal polynomials of α, α^2, and α^4 are the same. The minimal polynomials of α, α^2, α^4, and α^3 are listed in (17) [35]. Thus,

$$\phi_{1,2,4}(X) = 1 + X + X^6$$

and **(17)**

$$\phi_3(X) = 1 + X + X^2 + X^4 + X^6 .$$

The remainder of the division, *$b_i(X)$*, can be computed using a feedback shift register circuit in digital hardware. Two such circuits are needed since there are two different minimal polynomials. Since both minimal polynomials are degree 6, the remainders are both degree 5 and are given by (18). The coefficients in this equation, *b0* through *b5* and *B0* through *B5*, are functions of the received word divided by the corresponding minimal polynomial [35]. Thus,

$$b_{1,2,4}(X) = b0 + b1X + b2X^2 + b3X^3 + b4X^4 + b5X^5$$

and **(18)**

$$b_3(X) = B0 + B1X + B2X^2 + B3X^3 + B4X^4 + B5X^5 .$$

The syndromes are computed as shown in (19) where α^i is an element in GF(2^6) [35]. Thus,

$$S_1 = b_{1,2,4}(\alpha)\,,$$

$$S_2 = b_{1,2,4}(\alpha^2)\,,$$

$$S_3 = b_3(\alpha^3)\,, \qquad \textbf{(19)}$$

and

$$S_4 = b_{1,2,4}(\alpha^4)\,.$$

Expanding the equations in (19) results in the 6-bit syndromes computed from the coefficients of the remainders of the two polynomial divisions by *φ*$_{1,2,4}$*(X)* and *φ*$_3$*(X)* and the result is shown in (20), where

$$S_1 = b0 + b1\alpha + b2\alpha^2 + b3\alpha^3 + b4\alpha^4 + b5\alpha^5\,,$$

$$S_2 = (b0 + b3) + b3\alpha + (b1 + b4)\alpha^2 + b4\alpha^3 + (b2 + b5)\alpha^4 + b5\alpha^5\,,$$

$$S_3 = (B0 + B2 + B4) + B2\alpha + B4\alpha^2 + (B1 + B3 + B5)\alpha^3 + B3\alpha^4 + B5\alpha^5\,, \qquad \textbf{(20)}$$

and

$$S_4 = (b0 + b3 + b4) + b4\alpha + (b2 + b3 + b5)\alpha^2 + (b2 + b5)\alpha^3 + (b1 + b4 + b5)\alpha^4 + b5\alpha^5\,.$$

One interesting thing to note is that based on the way these are calculated, S_2 is equal to S_1 squared and S_4 is equal to S_2 squared. For the Peterson direct solution method for a 2 error correcting binary BCH code, only the odd syndromes are required to find the error-location polynomial [36]. However, as will be seen later, S_1 squared will be required in the error-location polynomial stage, so to avoid having to perform this multiplication step in the error-location polynomial stage, S_2 will be computed in the syndrome computation stage. A block diagram of the syndrome computation circuit is shown in Figure 21. The shift register is implemented to shift the entire 63 bit received code word through on a single parallel data clock cycle and the addition blocks are also performed on that same clock cycle [35].

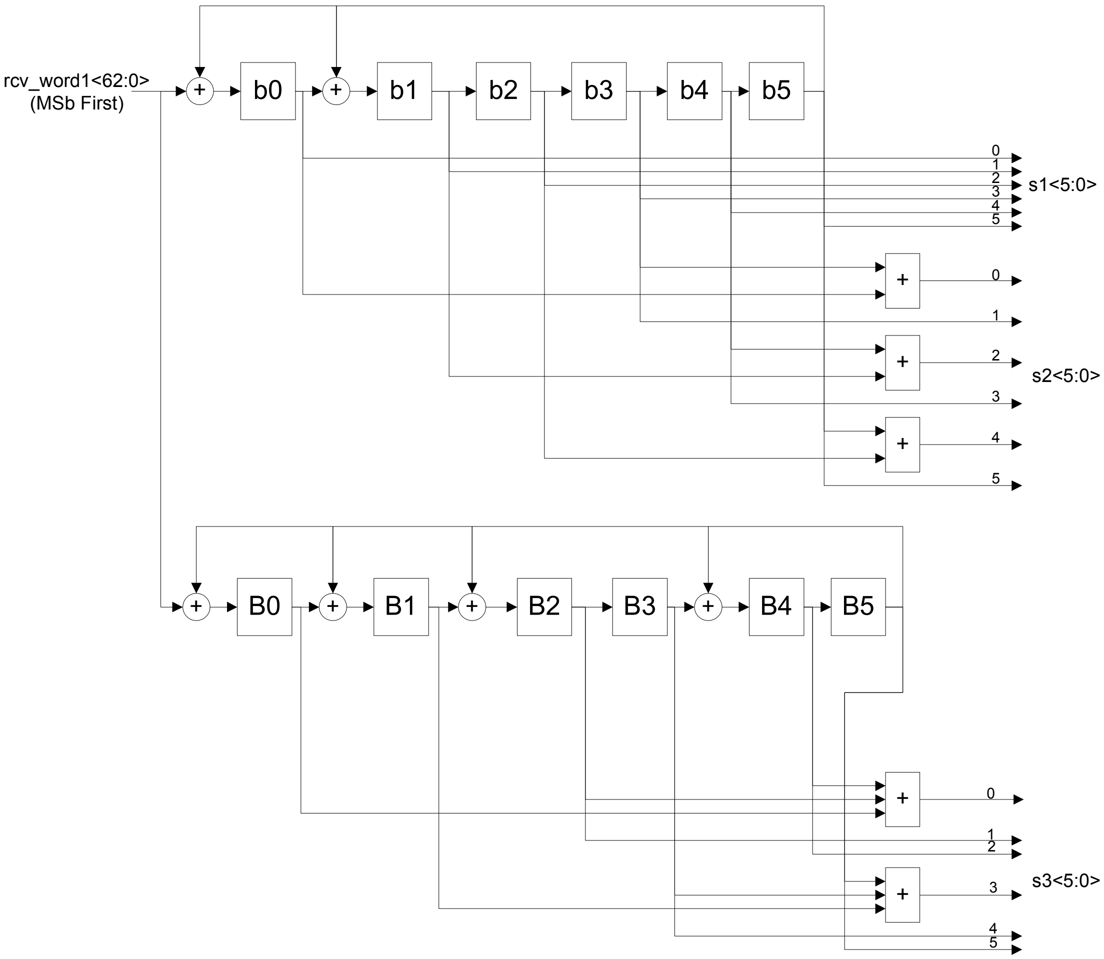


Figure 21 - Syndrome computation circuit.

The next step of the BCH decoder process is the computation of the error-location polynomial. The error-location polynomial in general form for any BCH code is given in (21) [35], where

$$\sigma(X) = \sigma_0 + \sigma_1 X + \sigma_2 X^2 + ... + \sigma_v X^v. \qquad \textbf{(21)}$$

The values of the coefficients σ_i are elements of the Galois field from which the BCH code was formed and the roots of (21) are the bit position locations of the errors in the received code word. Equation (22) shows the error-location polynomial for the (63,51) BCH code and the values of the coefficients solved using Peterson's direct-solution method [36]. Thus,

$$\sigma(X) = 1 + \sigma_1 X + \sigma_2 X^2,$$

$$\sigma_1 = S_1,$$

and **(22)**

$$\sigma_2 = \frac{S_3}{S_1} + S_1^{\ 2} = \frac{S_3}{S_1} + S_2.$$

The division in (22) can be performed by inverting S$_1$ and multiplying it by S$_3$. The inversion is performed by a lookup table built from combinational logic using the relationship in (23) for inversion in GF(2^6). For the purposes of this design, inversion of the 0 element will result in the 0 element even though it is technically undefined [36]. Thus,

$$0^{-1} = \mathit{undefined}\ ,$$

$$\alpha^{-0} = \alpha^0,$$

and **(23)**

$$\alpha^{-i} = \alpha^{63-i} \qquad i \geq 1.$$

The addition is defined as bitwise modulo-2 addition. The multiplication is defined by (24) and is performed using the circuit in Figure 22 [36]. The circuit is created using combinational logic to perform all 6 shifts on one clock cycle so that it completes within a single clock cycle in this design. Thus,

$$0 \cdot \alpha^{i} = 0,$$

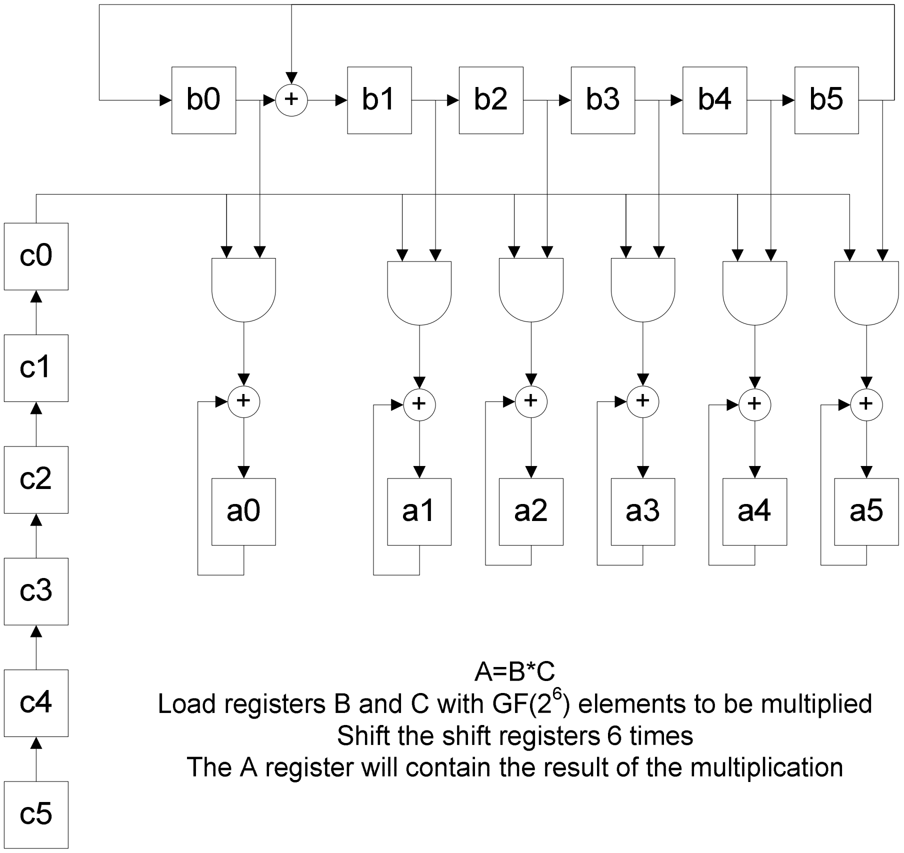
$$\alpha^{i}\alpha^{j} = \alpha^{i+j},$$

and

$$\alpha^{2^{m}-1} = \alpha^{0} \tag{24}$$

where

$$0 \le i, j \le 2^{m} - 2 .$$



A=B*C
Load registers B and C with GF(2^6) elements to be multiplied
Shift the shift registers 6 times
The A register will contain the result of the multiplication

Figure 22 - GF(2^6) multiplier circuit.

A block diagram for the whole error-location polynomial computation step is shown in Figure 23. One last thing to note about this step is that the values of the error-location polynomial coefficients indicate how many errors are in the received word by the following rules [36]:

- If $S_1 = 0$ and $S_3 = 0$ then there are no errors;
- Else if $S_1 = 0$ and $S_3 \neq 0$ then there are 3 or more errors;
- Else if $S_1 \neq 0$ and $\sigma_2 = 0$ then there is 1 error;
- Else there are 2 or more errors.

These rules are evaluated in this stage and the result passed to the error search stage for use in determining how many errors have occurred.

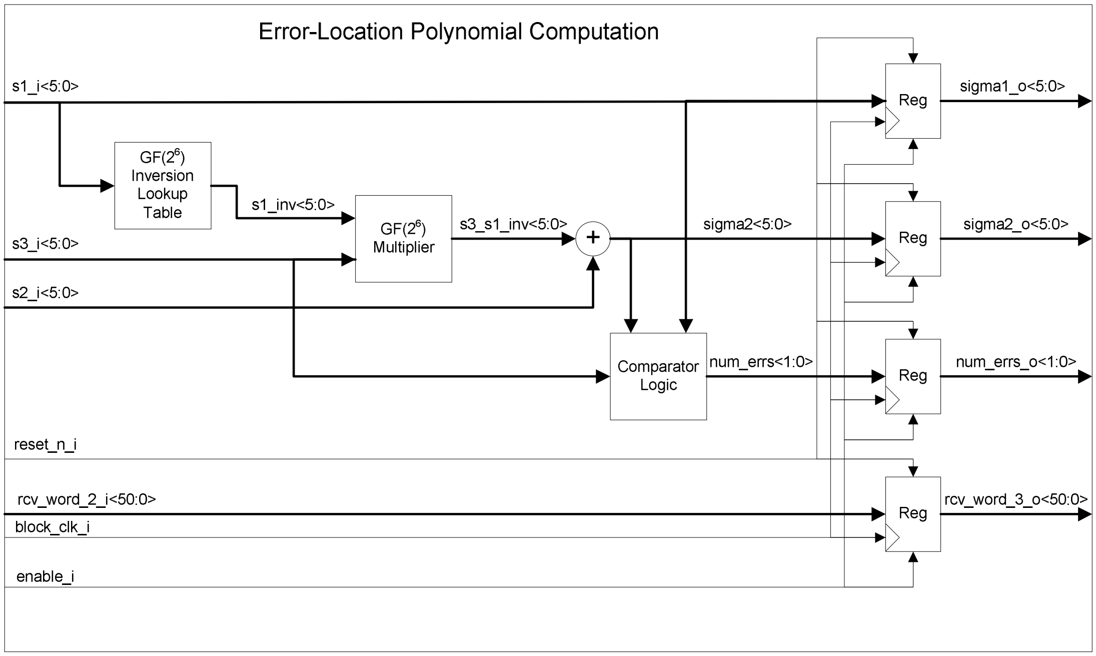


Figure 23 - Error-location polynomial computation circuit.

The final stage of the BCH decoder is the error-location search. This search is performed by trying each power of *α* successively in the error-location polynomial as demonstrated in (25). This algorithm is referred to as a Chien search [35]. Therefore,

$$\begin{gathered}\text{For i} = 1 \text{ to } 63, \\ \sigma\left(\alpha^i\right) = 1 + \sigma_1 \cdot \alpha^i + \sigma_2 \cdot \left(\alpha^i\right)^2. \\ \text{If } \sigma\left(\alpha^i\right) = 0 \text{, then an error has occurred in bit location (63-i).} \\ \text{If } \sigma\left(\alpha^i\right) \neq 0 \text{, then an error has not occurred in bit location (63-i).}\end{gathered} \qquad \textbf{(25)}$$

The circuit that implements this is shown in Figure 24.

The final stage of the decoder design is the (51,48) MRL decoder. The decoding operation is simple and is based on the following rules:

- If MRL<2:0> = 010 then invert the 48 data bits;
- Else if MRL<2:0> = 101 then leave the 48 data bits alone;
- Else set the mrl_error_o signal to a 1 to indicate a decoding error.

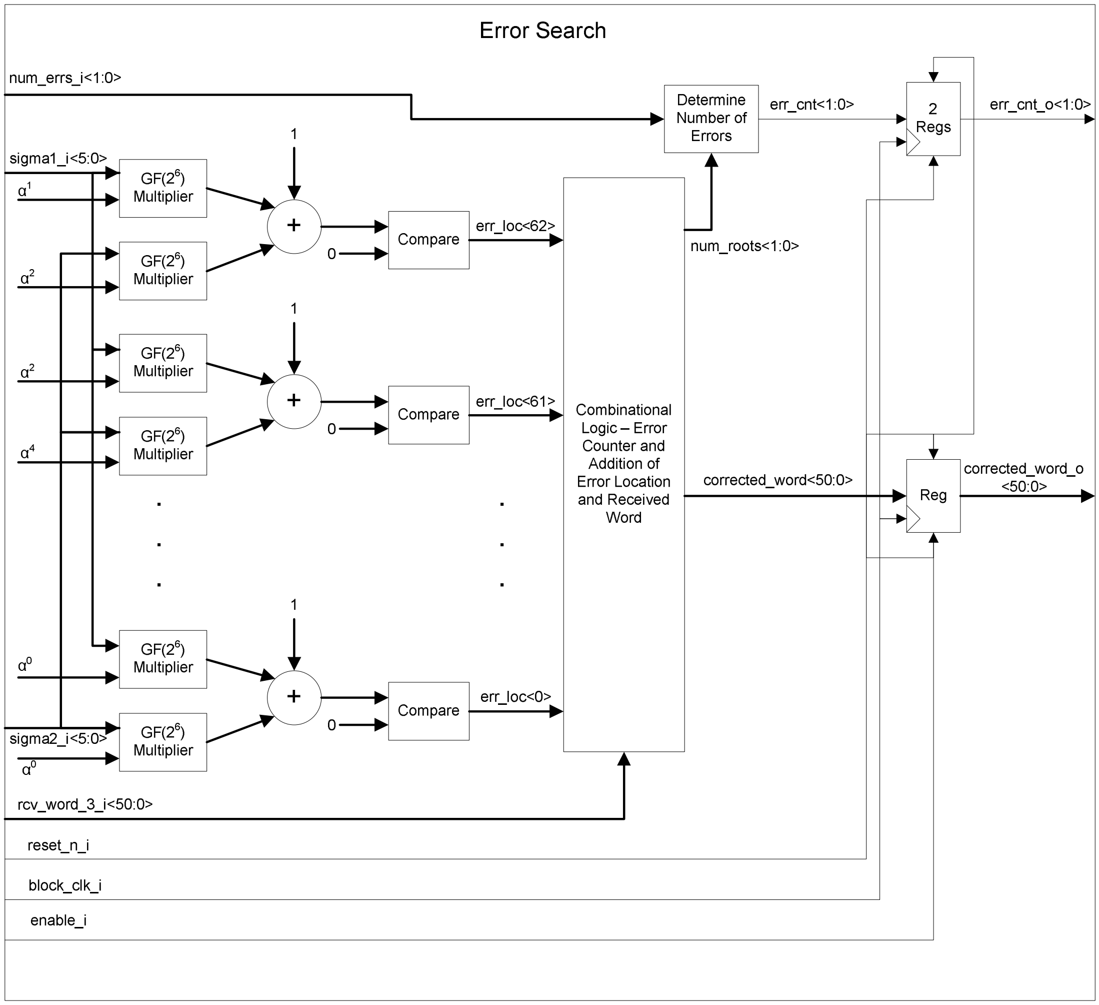


Figure 24 - Error-search circuit.

4 Bit-Error-Rate-Test Functional Block

The primary measure of performance of a digital communication system is the bit error rate. Bit error rate (BER) is defined by (26) [35], where

$$BER = \frac{\text{Number of Bit Errors}}{\text{Total Number of Bits}}. \qquad (26)$$

When comparing bit error rates between systems that include error correction coding and uncoded systems, it is necessary to make the comparison based on the BER for the information bits rather than the BER for transmitted bits which, for the coded system, will include redundant bits.

There is a lot of commercially availably test equipment for measuring bit error rate and characterizing serial digital multi-gigabit communication systems. Some vendors that have such equipment are Agilent Technologies, Synthesys Research, and Anritsu. The product offerings contain a lot of advanced features such as bit error location tracking, eye diagram creation, and jitter measurement [39]. The main drawback of such test equipment is that it is very expensive. An alternative to such expensive test equipment is to have built-in self test designed into the digital hardware in a serial digital multi-gigabit communication system. A field programmable gate array (FPGA) provides a good platform for built-in self test in these systems because it is configurable and easy to integrate such a self test block into the data path at any desired location to test not only the physical performance of the link but also to test coding and higher layer protocol blocks.

The development of the bit-error-rate-test functional block has been broken into a few steps. The first was to investigate how bit error rate measurements are made. The next was to determine what types of data patterns the block should support and identify which data patterns would be most useful in certain types of measurements such as jitter. After selection of appropriate data patterns, the ability to implement those patterns at multi-gigabit data rates was evaluated and then the block was designed and implemented.

4.1 Bit-Error-Rate-Test Measurement Background

Bit errors in a serial digital multi-gigabit communication system occur as the result of random Gaussian distributed noise. It is therefore impossible to predict exactly when they will occur because the errors will have a random distribution. Statistical methods are required to measure bit error rate. Three methods have been investigated for possible use in this project. The first method provides a confidence level for the true bit error rate being within some stated percent error of the bit error rate measurement based on the number of bit errors that were counted in that measurement. The second method provides a confidence level for the true bit error rate being better than a given bit error rate if a system is measured without any bit errors for a given amount of time. The third method provides an estimate and confidence level for the range of the true bit error rate based on making multiple measurements.

A binomial distribution can be used to describe the probability of counting a certain number of bit errors in a certain number of bits. The binomial distribution applies when the following conditions are met [40]:

- Bit errors are random;
- There are two outcomes, either a bit is correct or in error;
- All bits have the same probability of being in error;
- The number of bits measured must be the same regardless of the outcome of each bit measurement.

The first three conditions are met based on the jitter model of bit error rate described in section 3.4.4 and the last condition is a requirement of the measurement.

For a binomial distribution with a very large value of *b* (the number of bits measured) and with a very small value of *p* (the probability of an event such as a bit error), the Poisson distribution is a good approximation of the binomial distribution [40]. Some investigation using Mathcad showed that for a bit error rate of 10^{-12}, the binomial distribution could not be computed for very large values of b, the number of bits over which the distribution applies, to get enough errors to be statistically significant. The Poisson distribution did not suffer this problem, so it has been used in this analysis. For bit error rate applications, the Poisson distribution has a discrete probability density function that describes the probability of counting a certain number of bit errors in a certain number of bits. The distribution is given by (27) [40], where

$$\lambda = p \times b$$

and

$$\Pr(R = r) = \frac{\lambda^{(r)} e^{-\lambda}}{r!} . \qquad (27)$$

In (27), λ is a defined parameter of the Poisson process, p is the probability of a bit error, r is the number of bit errors counted when b bits are measured, and $\Pr(R=r)$ is the probability that the number of bit errors counted, R, will be equal to r when b bits are measured. The mean and standard deviation for the Poisson distribution are given in (28) [40], where

$$mean = \lambda$$

and **(28)**

$$\sigma = \sqrt{\lambda}\,.$$

Assuming that the measured number of errors is Poisson distributed, an inference can be made about the accuracy of a given measurement based on the number of errors that were measured. This is accomplished using (29) [40], where

$$C = \sum_{i=R-err\cdot R}^{R+err\cdot R} \Pr(R=i)\,. \qquad \textbf{(29)}$$

In (29), C is the confidence that the measured number of bit errors, R, is within the fractional error +/- *err* of the number of bit errors that would be expected based on the actual average bit error rate. For example, a particular measured bit error rate might be stated as being within +/- 5% of the actual bit error rate, with 90% confidence. One thing to note is that the value of *err* must either be chosen so that the limits of the summation in (29) are integers, or those limits must be rounded. Any rounding would introduce some error. For a data rate of 3.125 Gbps and a bit error rate of 10^{-12} some values of C, *err*, and test times required to make the measurements are shown for different values of *R* in Table 7.

Table 7 - Measurement accuracy versus number of bit errors measured, and corresponding test times, for 3.125 Gbps data rate and measured BER of 10^{-12}.

R **(number of errors that must be measured)**	***Err*** **(+/- error percentage between the measured and true bit error rates)**	***C*** **(confidence that the measured bit error rate is within +/-** ***Err %*** **of being accurate)**	**Average test time required to make the test (hr)**
10	10%	36.4%	0.889
10	30%	73.4%	0.889
50	10%	56.3%	4.44
50	20%	86.3%	4.44
100	10%	70.7%	8.89
100	20%	96.0%	8.89
400	5%	69.5%	35.56
400	10%	95.7%	35.56
1000	5%	89.0%	88.89
1000	10%	99.9%	88.89

Even for high data rates, the test times get quite long for a reasonable amount of confidence in the measurement for a bit error rate of 10^{-12}, as shown in Table 7. If the system was actually operating with some noise margin above and beyond that required for 10^{-12} operation, the true bit error rate would be even less, and test times to measure this lower bit error rate at a particular confidence level could greatly increase. In this situation, test times to merely demonstrate that the bit error rate is lower than 10^{-12} (rather than to measure the lower bit error rate) would not in general be greater than the test times needed for the system that operates without the noise margin. Bit error rate testing to demonstrate compliance with a stated limit is discussed in the next paragraph.

The second method provides a way to determine that the bit error rate is below some specified threshold with a certain amount of confidence. This method provides an improvement on test times, but does not allow exact measurement of the bit error rate.

This is usually sufficient in real designs since most design requirements are that the system operate at a bit error rate that does not exceed some level. For low bit error rate systems, this method is often the best choice. The confidence that a system is operating below some bit error rate threshold is given by (30) [41], where

$$T = \frac{-\ln(1-C)}{BER \times f_b} . \tag{30}$$

In (30), T is the time that the test must be conducted without any bit errors, C is the confidence that the true bit error rate is below a specified bit error rate *BER*, and f_b is the data rate at which the system is operating. Table 8 shows some test times required for different confidence levels and different values of specified bit error rate for a data rate of 3.125 Gbps.

Table 8 – Test time versus specified bit error rate threshold and confidence, at 3.125 Gbps.

***BER* (maximum bit error rate)**	***C* (confidence that the true bit error rate is below the maximum bit error rate)**	***T* (time required of bit error free measurement given *C* and *BER*) (hr)**
10^{-12}	70%	0.107
10^{-12}	95%	0.266
10^{-12}	99.9%	0.614
10^{-15}	70%	107.0
10^{-15}	95%	266.3
10^{-15}	99.9%	614.0
10^{-17}	70%	10702.0
10^{-17}	95%	26628.7
10^{-17}	99.9%	61402.3

From Table 8, test times for determining if a bit error rate is below 10^{-12} are vastly improved relative to those in Table 7 for measuring the bit error rate. Test times for determining if the bit error rate is below 10^{-15} span several days, which might be plausible for some testing applications, but test times for determining if the bit error rate is below 10^{-17} are greater than one year and would therefore be prohibitive.

The third method uses multiple tests of the same duration and assumes that the results are randomly distributed around the true bit error rate. Using statistical methods, a range of the bit error rate can be determined with a certain confidence interval. Ideally, a very large number of tests would be conducted and the normal distribution would apply to the results. Since the time required to count even small numbers of errors can be quite large for very low bit error rates, it is not usually practical to conduct a very large number of tests. Another factor is that the standard deviation of the bit error rate measurements will not usually be known, especially for a newly designed system. For cases where statistical inferences are made based on a limited number of tests and also on a standard deviation and mean estimated from a set of sample tests, the Student t-distribution applies. This distribution takes into account the uncertainty of the standard deviation calculated from the test samples [40].

When using the t-distribution, a number of tests are conducted for a fixed duration and the number of errors is counted for each test. The mean and standard deviation of the number of errors counted in the tests is computed with (31) and (32) respectively [40], where

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \textbf{(31)}$$

and

$$s = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2}{n-1}} \; . \qquad (32)$$

In (31) and (32), the x_i values are the numbers of errors counted in the tests and n is the number of tests conducted. A value then is found for the t-distribution from a computed table or computer program at a certain confidence level and a range for the bit error rate, based on these measurements, is then calculated using (33) [40], where

$$\overline{x} - t1 \cdot \frac{s}{\sqrt{n}} \le BER \le \overline{x} + t1 \cdot \frac{s}{\sqrt{n}} \; . \qquad (33)$$

In (33), *t1* is the value of the t-distribution for the desired confidence interval, $\overline{x}$ is the mean of the error count samples, and *s* is the standard deviation of the error count samples.

This method allows for shorter test times per test than the first method, but since multiple tests are required the actual test time will be similar to the first method. There may be some instances where this method has advantages, but generally it is just another way of approaching the determination of bit error rates from measurements.

4.2 Bit-Error-Rate-Test Data Patterns

The sources used to find appropriate bit-error-rate-test (BERT) data patterns to use in serial digital multi-gigabit communication systems were commercially available test equipment and industry recognized standards and recommendations. All test equipment investigated provided for relatively short programmable word patterns and relatively long pseudorandom bit sequences, both of which are repeated periodically. An example of a BERT for serial digital multi gigabit communication systems is the BERTScope 7500A from Synthesys Research [39]. The datasheets for the test equipment specify the details of some specific pseudorandom bit sequences (PRBSs), and those sequences are also specified in [42].

The patterns that have been chosen to be implemented in this design are listed in Table 9 with some details about each pattern.

Table 9 - Data patterns selected for BERT block design.

Pattern	**Length (bits)**	**Length in Time for 3.125 Gbps**	**Generator Polynomial for LFSR**	**Max 0s Run (bits)**	**Max 1s Run (bits)**	**Source**
Programmable Word	8 – 64 bits	2.56 ns – 20.48 ns	NA	64	64	NA
$2^{11} - 1$ PRBS	2047	655.04 ns	$X^{11} + X^{9} + 1$	10	11	[42]
$2^{31} - 1$ PRBS	2,147,483,647	687.2 ms	$X^{31} + X^{28} + 1$	30	31	[42]

The PRBS patterns are generated using linear feedback shift registers (LFSRs) with the feedback taps in the locations corresponding to the exponents on the terms in the generator polynomial.

4.3 *Bit-Error-Rate-Test Block Design*

The bit-error-rate-test block consists of two separate blocks, the pattern generator and the receiver. The pattern generator generates the pattern and the receiver synchronizes to the pattern and then counts bit errors in the received data after synchronization.

4.3.1 Bit-Error-Rate-Test Pattern Generator

The pattern generator block can generate any of the three selected patterns identified in Table 9. A functional block diagram for the pattern generator is shown in Figure 25. The inputs and outputs are described in the datasheet in section 10, which is Appendix B – BERT Block Datasheet.

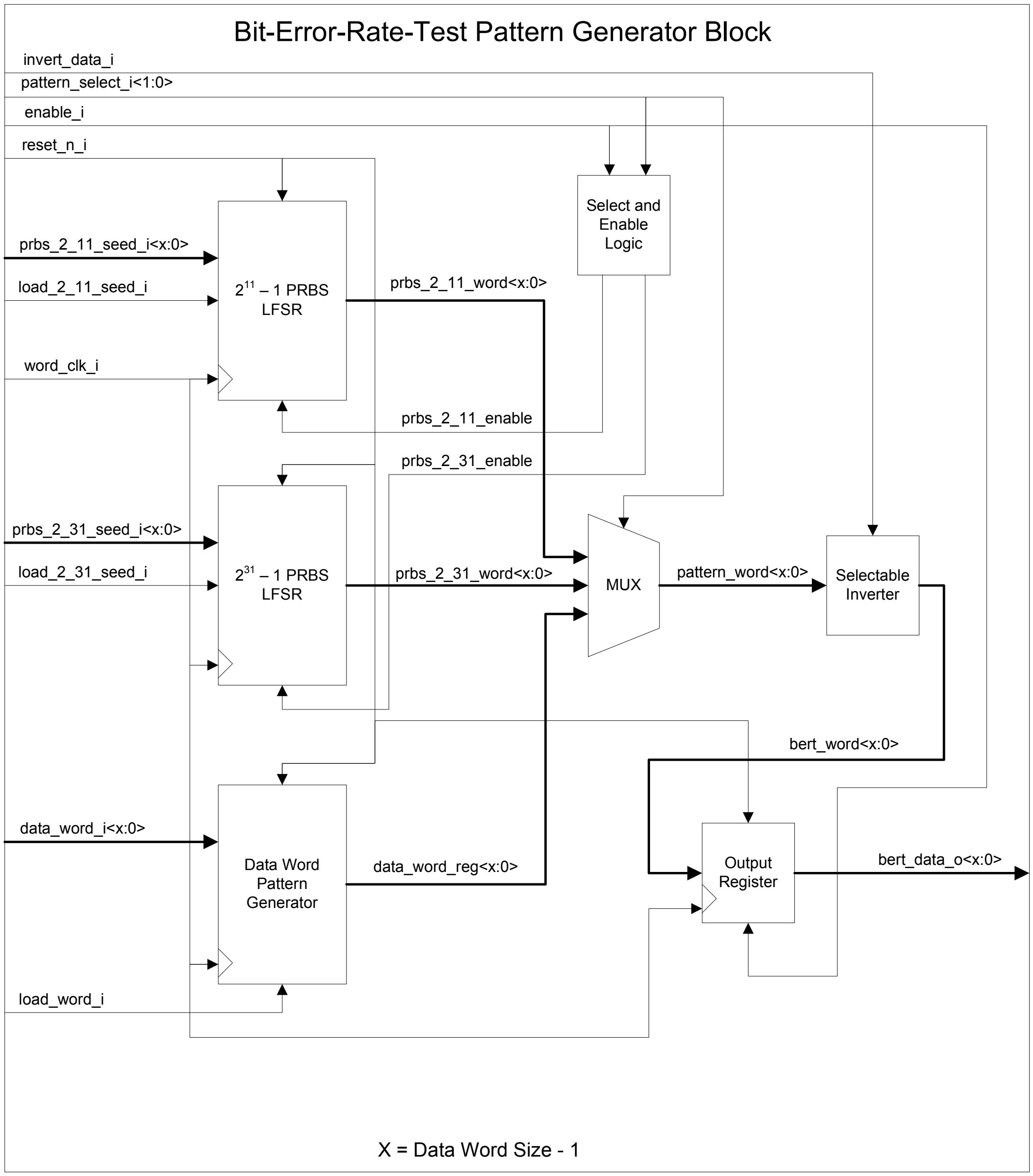


Figure 25 - BERT pattern generator block diagram.

The data word size of the pattern generator is configurable to values between 8 bits and 64 bits. The data word pattern generator simply captures a data word *x*+1 bits wide into a register, where *x* is a parameter used in the pattern generator shown in Figure 25, and then sends that word out on every clock cycle of word_clk_i. It is possible for the data word to be static or it can change on every clock cycle of word_clk_i in which case a custom user defined data pattern could be generated.

The two PRBS data pattern generators work in a very similar fashion. Each is constructed using a LFSR as shown in Figure 26 and Figure 27 [42].

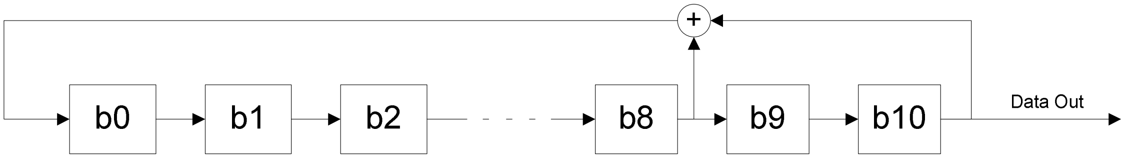


Figure 26 - 2^{11} - 1 PRBS LFSR block diagram, for $P(X) = X^{11} + X^{9} + 1$.

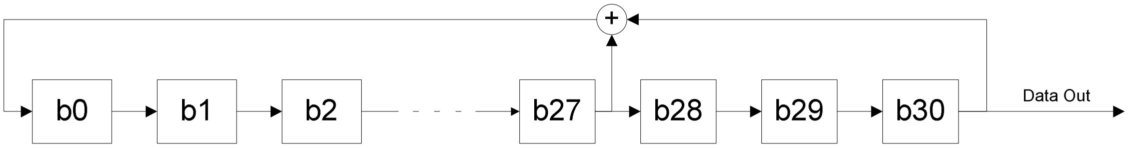


Figure 27 - 2^{31} - 1 PRBS LFSR block diagram, for $P(X) = X^{31} + X^{28} + 1$.

Since the pattern generator block must operate at the word clock rate and not at the bit clock rate, the hardware implementation is not a simple LFSR as shown. In hardware the LFSR for each pattern must be simultaneously shifted *x*+1 times for each clock cycle of the word clock. This is accomplished with combinational logic. The hardware must compute the next bit value for each register in the LFSR after *x*+1 shifts, and all *x*+1 of the output bits within a single clock cycle of the word clock. For the $2^{11} – 1$ generator, the worst case equation for each of these calculations was computed symbolically for a word size of 64 bits using Mathcad and is given in (34), where

$$b0_{next} = b2 \oplus b5 \oplus b1 \oplus b3 \oplus b9 \oplus b7 \oplus b10 \oplus b8$$

and **(34)**

$$data\_out_{38} = b5 \oplus b1 \oplus b3 \oplus b10 \oplus b8\,.$$

For the $2^{31} – 1$ generator, the worst case equation for each of these calculations was computed similarly and is given in (35), where

$$b0_{next} = b29 \oplus b23 \oplus b26 \oplus b20$$

and **(35)**

$$data\_out_{59} = b2 \oplus b30 \oplus b27\,.$$

The $2^{11} – 1$ pattern generator actually requires more combinational logic for each bit, but only contains 11 shift register bits versus 31 for the $2^{31} – 1$ PRBS generator. For hardware implementation in VHDL it is not required to know the equations for all of the next shift register bits and output data bits because a for loop can be used in which the synthesis tools will automatically convert into the correct equations.

4.3.2 Bit-Error-Rate-Test Receiver

The receiver block can synchronize to and count errors in any of the three selected patterns. A functional block diagram for the receiver is shown in Figure 28. The inputs and outputs are described in the datasheet in section 10, which is Appendix B – BERT Block Datasheet.

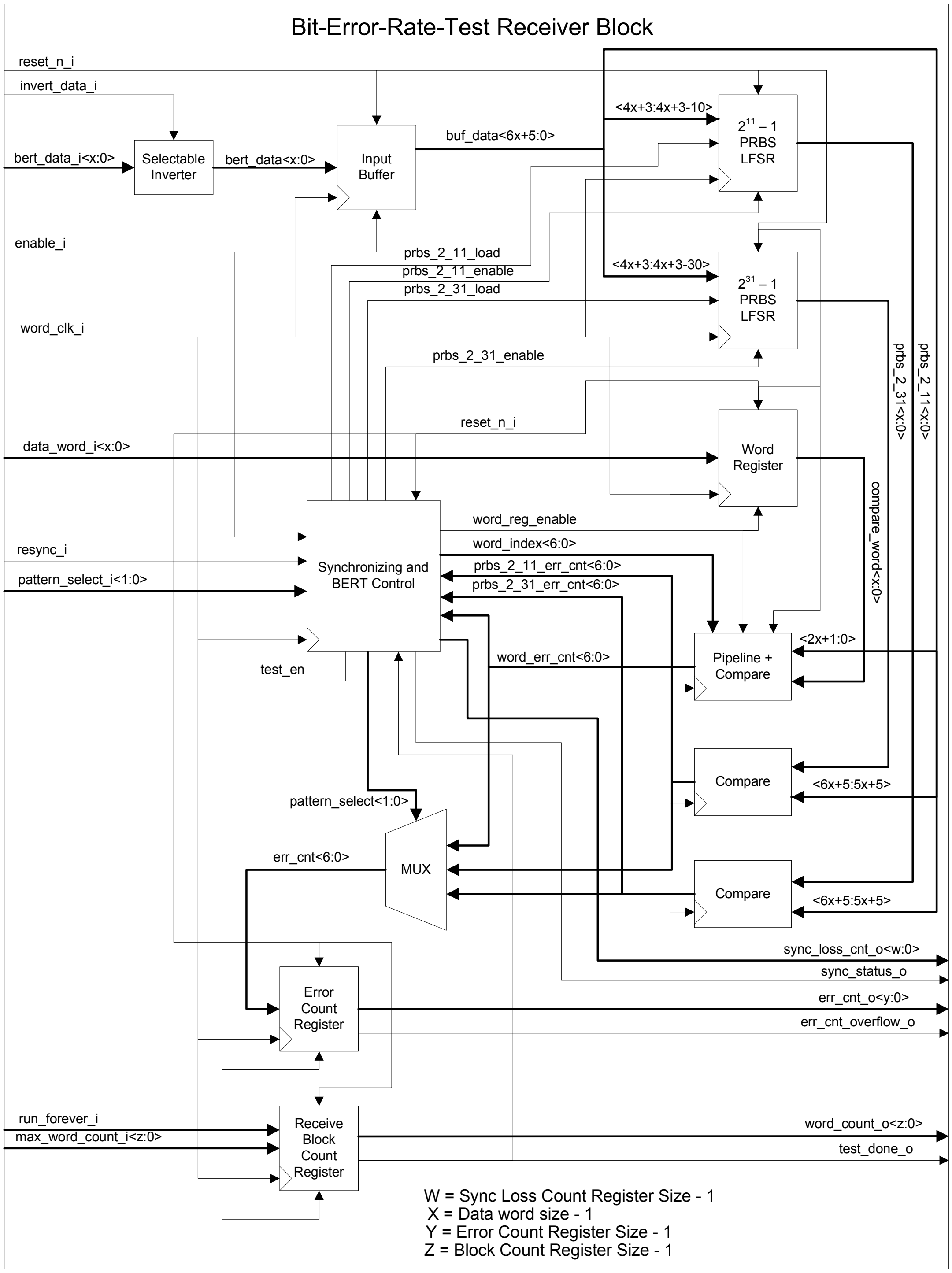


Figure 28 - BERT receiver block diagram.

The BERT receiver block consists of several sub-blocks including the input buffer, the pattern generators, the error comparators and counter, the synchronization and control block and the output registers. The basic process flow of the BERT receiver block is shown in Figure 29.

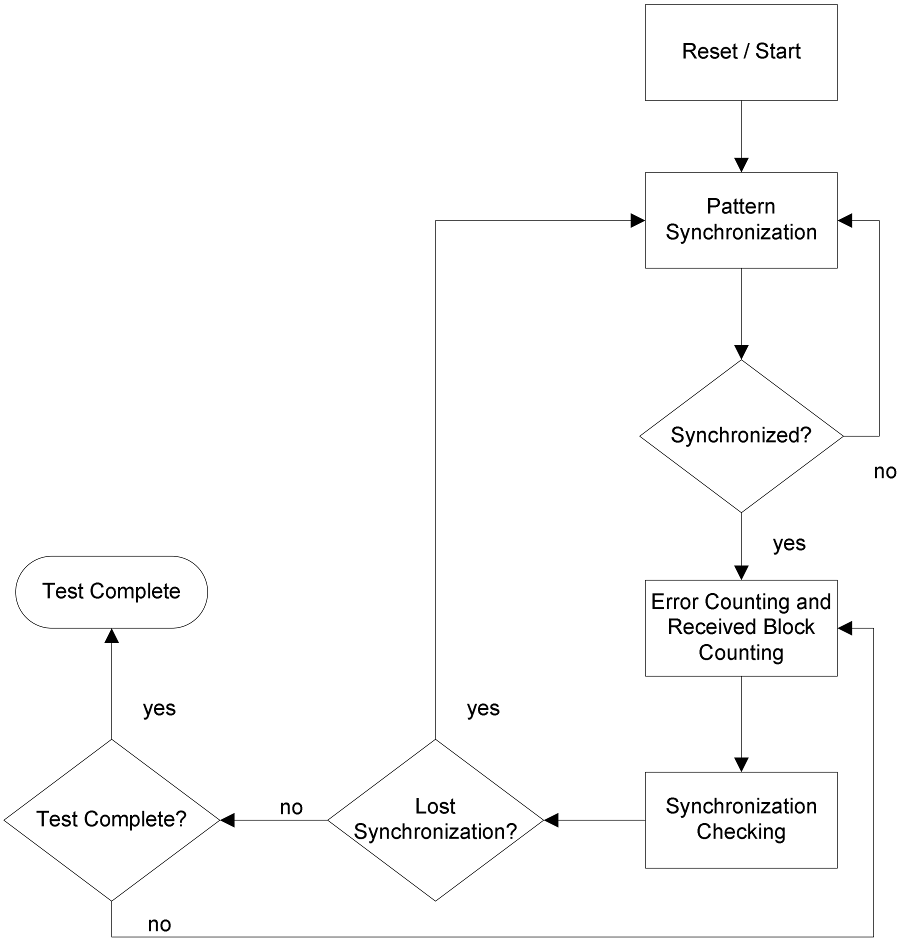


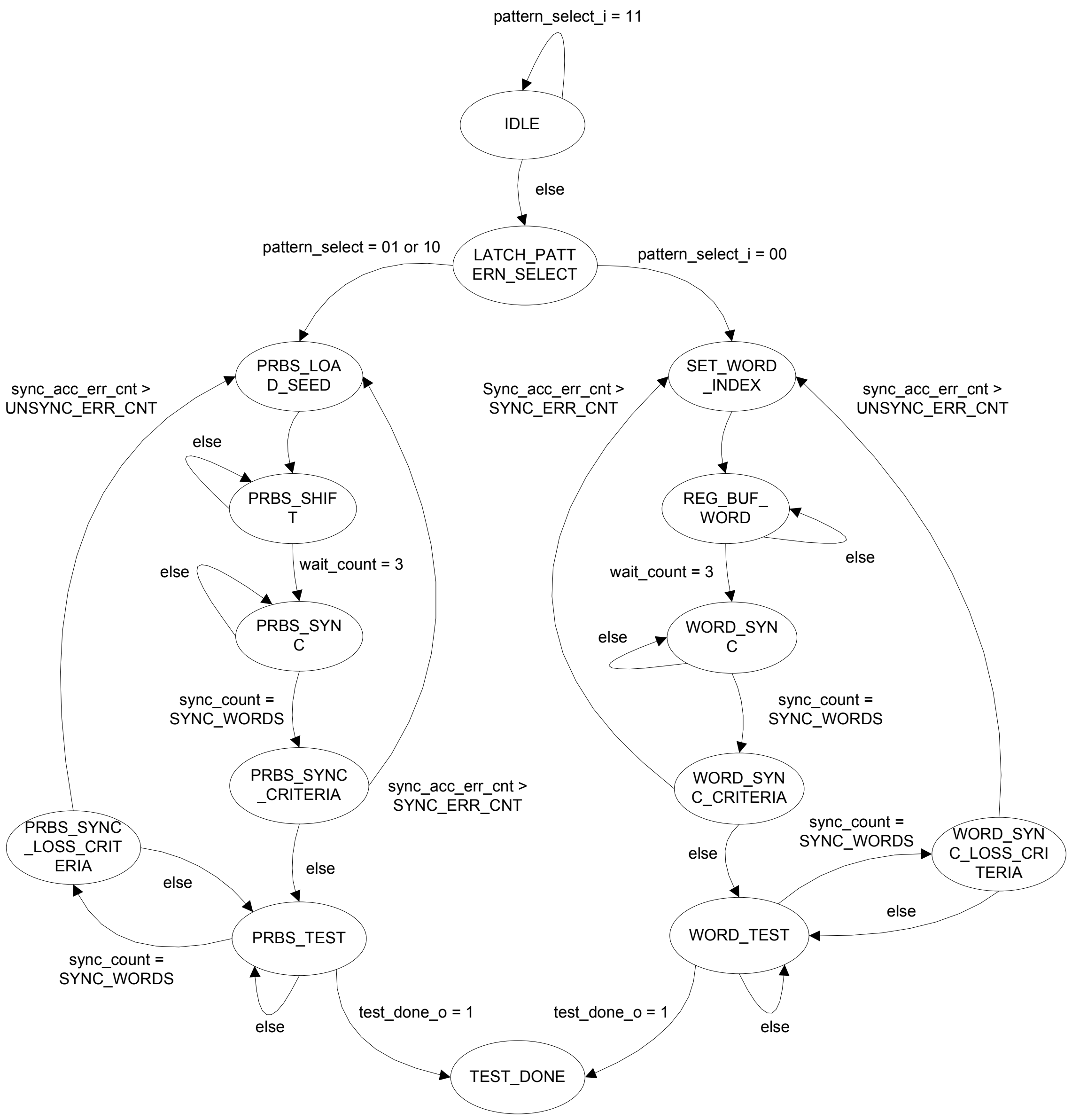
Figure 29 - BERT receiver process flow diagram.

The first stage of the receiver is the input buffer stage. There are two constraints on the input buffer size. For the programmable word pattern, the pattern could start anywhere within an input data word and continue for x+1 bits. In order to search for and be assured of finding the data word in the buffer at least two data words must be in the buffer. For the PRBS patterns, the pattern generators must be seeded with the received data and then shifted x+1 times on a single clock cycle of the word clock, comparing the output data from the LFSRs to the received data. The $2^{31} - 1$ PRBS pattern has 31 shift register stages so it requires 31 bits to seed it. Therefore, the second requirement is that the length of the buffer be at least 31 bits long. Setting the buffer size to 4 data words and limiting the size of a data word to between 8 bits and 64 bits accomplishes both of these requirements. In the design, two additional stages were added to the buffer because it takes one clock cycle to load the LFSR and one clock cycle to perform the first shift of the LFSR. So in order to compare the output of the LFSR with the received data, the data must be saved for two additional clock cycles, thus, the two additional buffer stages. So the buffer in the design contains 6 word stages for a total of 48 registers for an 8-bit data word and 384 registers for a 64-bit data word. For an 8-bit data word there is very little wasted space in the buffer. However, for anything larger than a 31-bit data word, only one buffer stage is required for seeding the LFSRs and in that case three of the buffer stages would be wasted. The number of wasted registers in these cases would be at most 192 for a 64-bit word size. The design could be optimized but, 6 buffer stages are used regardless of the word size for simplicity of implementation.

For the PRBS patterns, the receiver block contains the same pattern generator block as in the BERT pattern generator block described in section 4.3.1. Once synchronization occurs, the pattern generators can be shifted x+1 times per clock cycle of the word clock, to generate the expected receive data for comparison to the received data. For the programmable word pattern, the receiver block contains a register that captures the expected data word on the input signal data_word_i.

The synchronization and control block is a state machine that controls all of the enables and other control signals inside the BERT receiver. For the PRBS patterns, the synchronization process involves seeding the appropriate PRBS pattern generator and then counting bit errors over a fixed interval. If the bit error rate is less than a threshold, then synchronization has occurred and the block allows the error counter and word counter to start counting. For the programmable data word pattern, the synchronization process involves choosing a start index in the input data buffer and comparing the bits in the input data buffer starting at that index and continuing for x+1 bits after that to the expected data word. If the bit error rate is less than a threshold, then synchronization has occurred and the block allows the error counter and word counter to start counting. If the bit error rate is greater than the threshold, then synchronization has not occurred and the process will be repeated with the starting index incremented by 1. The starting index will be incremented over the range of 0 to x and then wrapped back to 0. For both types of patterns, the synchronization and control block continues monitoring the bit error rate of the received data over fixed intervals and if the bit error rate exceeds a certain threshold, then synchronization is lost and the block tries to resynchronize. If synchronization is lost, a counter is incremented to keep track of the number of times synchronization has been lost and it is available as an output on the signal sync_loss_cnt_o. The error counter and word counter are disabled during resynchronization.

The block is designed so that the fixed interval and the bit error rate thresholds for synchronization and losing synchronization are programmable. Normally the bit error rate threshold for synchronization will be set much lower than the bit error rate threshold for loss of synchronization to allow for some hysteresis [43]. The state machine for the synchronization and control block is shown in Figure 30.



All state transitions require enable_i = 1
If enable_i = 0, then the state machine
always remains in the current state, but the
test_en signal is deasserted

All states except for TEST_DONE will
transition immediately back to the IDLE
state if the resync_i input signal is asserted

Figure 30 - Synchronization and BERT control block state machine.

Once synchronization has occurred, the synchronization and control block enables the error counting and word counting. The multiplexer (mux) is selected for the correct error count and an accumulator adds the error count on each clock cycle to the accumulated error count with the new value being captured into the error count register on each clock cycle. The word counter is incremented by one on every clock cycle. The word counter block also has a comparator in it that compares the counter value with an input value specifying the number of words for which the test should continue. Once the word counter reaches that number, the test is complete and the synchronization and control register will remain in the idle state until the block is reset. The block can also be put into an open ended test mode that does not stop after the maximum word count is reached by setting the run_forever_i input signal to a logic high.

5 Implementation and Integration Results

This section describes the hardware design that integrates the bit-error-rate-test (BERT) block and the error correction code (ECC) block together with a multi-gigabit serial transceiver in the Altera Stratix GX device. The implementation is specific to the Altera Stratix GX development board. Some results from testing of the integrated blocks are also presented.

5.1 Implementation Design

The design integrates the ECC and BERT blocks described in sections 3 and 4 with the serial transceiver in the Altera Stratix GX device. In order to compare bit error rates for coded versus uncoded data, two designs were actually implemented. One included a block to perform both ECC and BERT functions and the other only included the BERT block. This section describes the implementation of the version with the ECC and BERT block in detail, but the BERT-only version is very similar. A top level block diagram of the design is shown in Figure 31. The design implements both transmit and receive paths and the intent is that they will be connected in an external loopback configuration.

There are several sub blocks in this design. The major blocks are the BERT + ECC transmit block and the BERT + ECC receiver block and those will be described in more detail. The Altera Stratix GX transceiver block is contained within the Altera Stratix GX FPGA. This design configures the block in x16 data width mode and for a serial data rate of 3.125 Gbps. The reference clock provided to this block is 156.25 MHz on the xtal1_sgx_i signal input to the FPGA from the development board. The block is configured to multiply this clock to the data rate with its internal PLL.

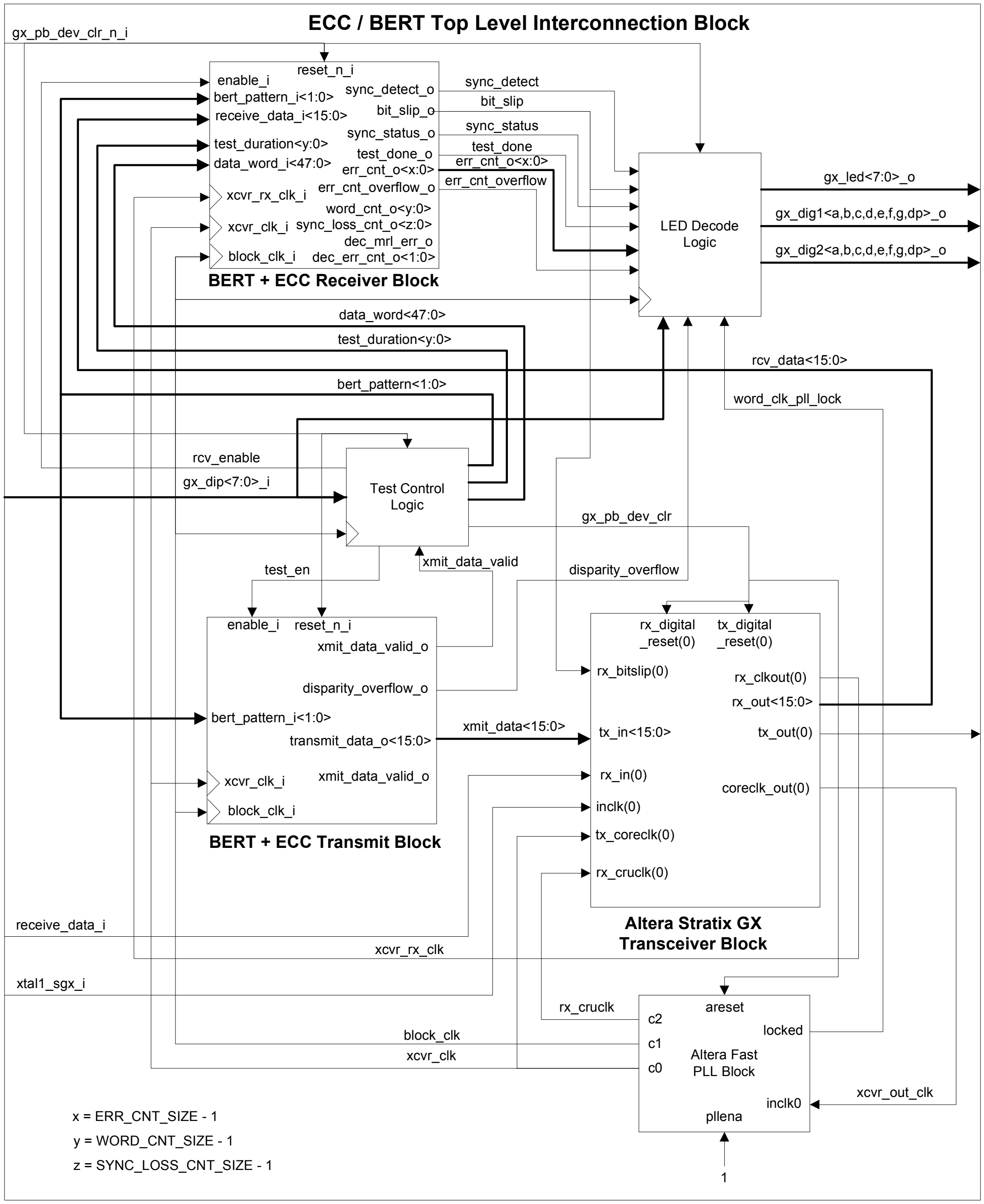


Figure 31 - BERT and ECC integration design top level block diagram.

In addition to the reference clock there are several other clocks used in the design. A divide-by-16 version of the data rate clock inside the transceiver is provided as an output as coreclk_out (195.3125 MHz). The transceiver also provides a divide-by-16 version of the clock recovered from received data stream as an output on rx_clkout (195.3125 MHz). The transceiver is configured such that the 16 bit receive data is phase aligned with this clock.

The coreclk_out is connected to a PLL in the stratix GX device to generate all the other clocks needed in the design. The first clock, rx_cruclk, is divided by 5/4 to match the reference clock frequency (156.25 MHz) and this is connected to the transceiver as the clock used to train the receive PLL. To operate at 3.125 Gbps, the transceiver requires a separate reference clock input, so that is why the xtal1_sgx_i clock couldn't be used. The next two clocks, block_clk and xcvr_clk are phase aligned with each other. The block_clk is a slower parallel clock for the encoder and decoder and for the BERT generator and receiver, and is a divide-by-64 version of the data rate clock (48.828125 MHz). The xcvr_clk matches the parallel data rate coming out of the transceiver and is a divide-by-16 version of the data rate clock (195.3125 MHz). The transceiver clock xcvr_clk is used to clock a multiplexer circuit to go from the x64 data width to the x16 data width in the transmit direction, and a demultiplexer circuit to go from the x16 data width to the x64 data width in the receive direction.

The BERT and ECC transmit block connects the BERT pattern generator to the MRL and ECC encoder and also performs multiplexing from x64 data width to x16 data width for connection to the transceiver. This block also contains some logic to generate a signal alternating between 0 and 1 to use as the pad bit. The transmit block design is shown in Figure 32.

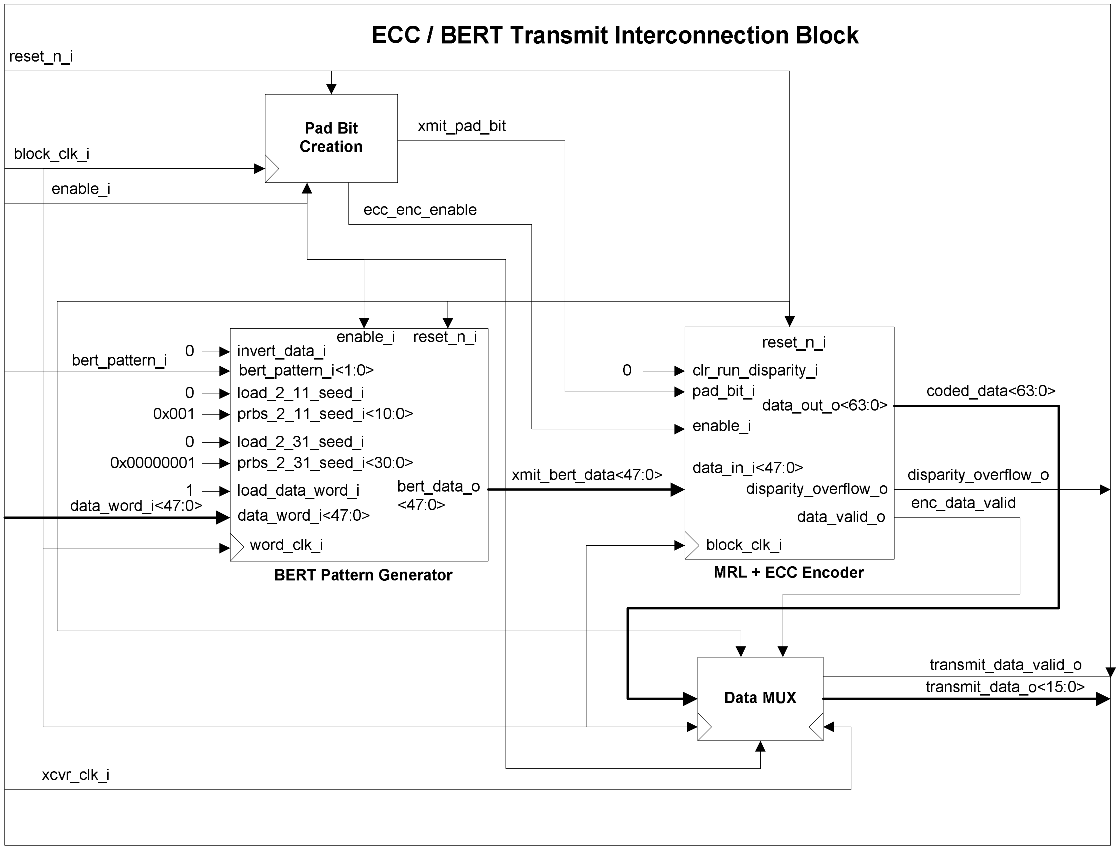


Figure 32 - ECC and BERT transmit block diagram.

The BERT and ECC receive block is a bit more complicated than the transmit block. The incoming data must be demultiplexed and phase aligned with the divide-by-64 block_clk, then decoded, and finally synchronized within the BERT receiver and have errors counted. An ECC block synchronization must also be performed. A block diagram of the BERT and ECC receive block is shown in Figure 33.

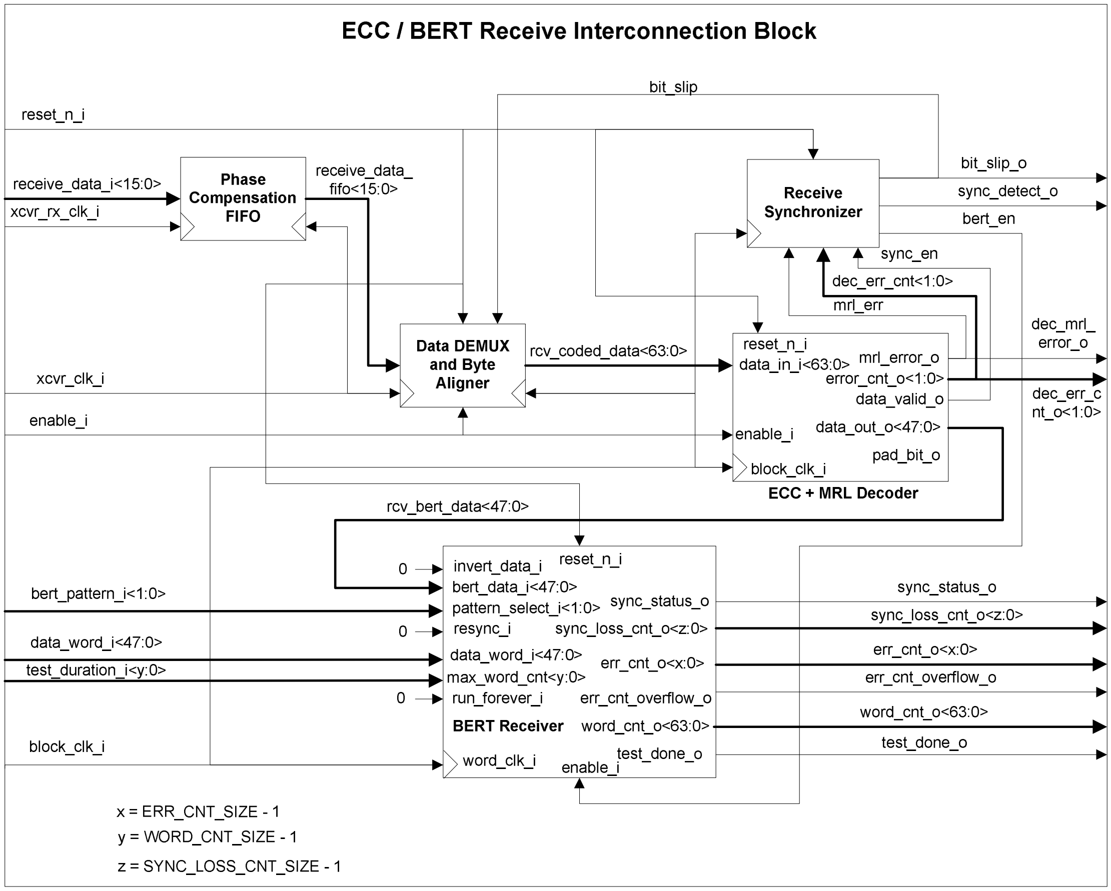


Figure 33 - BERT and ECC receive block diagram.

For the first step in the receive data flow, the incoming receive data that is phase-aligned to the recovered clock, xcvr_rx_clk, must be realigned to the divide-by-16 data rate clock, xcvr_clk, that is in phase with the slower divide-by-64 clock, block_clk, so that the demultiplexer circuit can function. This realignment is accomplished through the use of an Altera megawizard FIFO block configured in x16 data width on both sides. The FIFO is set up so that two clock cycles after data begin to be written in, those data will be read out on every subsequent clock cycle. The FIFO size is 8 words deep, and since the clock frequencies on both sides are equal the FIFO is guaranteed to never overflow or underflow. The two clock cycle delay ensures that the location being read from is never the same as that being written to eliminating possible timing violations between different time domains. The output data from the FIFO is aligned with the xcvr_clk signal.

Next the data is demultiplexed from 16 bits wide to 64 bits wide at the slower block_clk frequency. The data demultiplexer block also contains a buffer that is two 64-bit words deep, allowing for any possible alignment of the 8 bytes in the 64-bit word to be extracted. When the receiver first starts receiving data, there is no guarantee that the alignment of the data coming out of the demultiplexer matches a transmitted ECC code word. To accomplish this alignment, there is a state machine in the receive synchronizer block that monitors that error count and MRL error outputs from the ECC and MRL decoder to determine if alignment has occurred. If there are errors, then the state machine causes the bit_slip_o output to be asserted which is sent to the transceiver and causes it to slip its bit alignment by one bit. The bit alignment slip in the transceiver, however, wraps around on word boundaries, so that after 16 bit slips the bit alignment will be exactly as it was in the beginning. To account for this, the bit_slip signal is also connected to the demultiplexer block where a counter causes the byte alignment to change by one after every 16 bit slips since if 16 bit slips occurred without synchronization being found, the byte alignment must be off. The combination of the state machine and the demultiplexer will continuously cycle through all possible alignments until the right alignment is found and the decoded data are error free. This receive synchronization method is a great advantage to a system that uses error correction coding and was somewhat of an unanticipated benefit of the coding. This synchronization will be referred to as ECC block synchronization from here forward. The criteria for ECC block synchronization was set to the following:

- The ECC decoder error count had to be 0 or 1 for each of 8 consecutive words (it could not be 2 or 3);
- There could be no MRL errors for 8 consecutive words.

The state machine is shown in Figure 34. Once ECC block synchronization occurs, the design assumes that it remains synchronized and will never try to resynchronize.

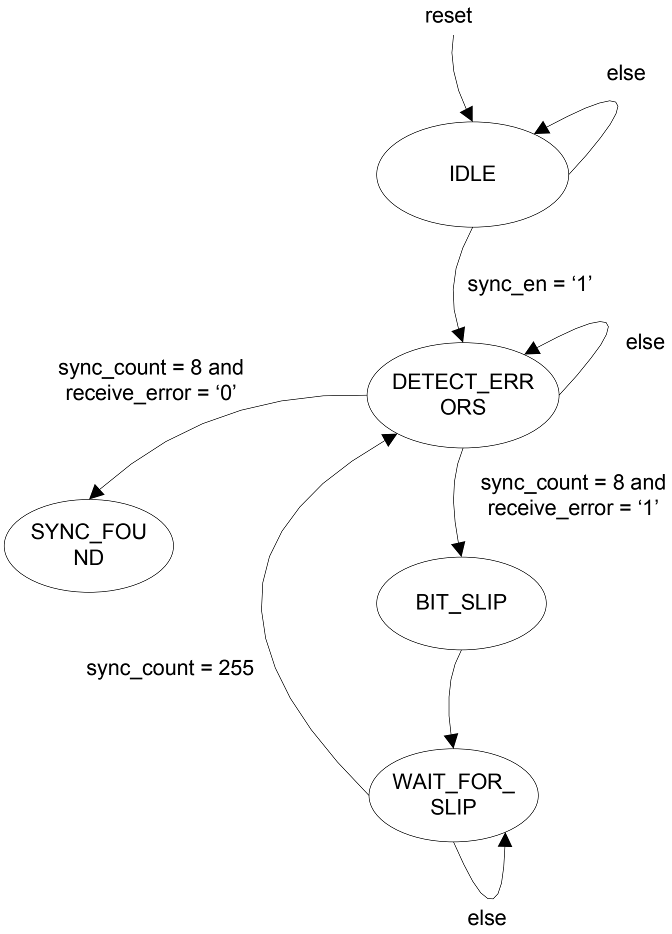


Figure 34 - ECC block synchronization state machine.

The last two parts of the path are the ECC and MRL decoder and the BERT receiver. The design is set up such that the BERT receiver will not try to achieve data pattern synchronization until ECC block synchronization has occurred. The receiver was set up with the data pattern synchronization criteria as no more than 1 bit error over 1024 data words. The BERT receiver was set up so that data pattern synchronization would be lost if more than 50 bit errors occurred over 1024 data words.

One additional complication in the design that was encountered is that the BERT and ECC blocks were both designed to shift data serially most significant bit first. The Altera Stratix GX transceiver shifts data serially least significant bit first and when multiple bytes are used they send data least significant byte first. This does not really matter in the implementation that combines the ECC and BERT blocks since the ECC block synchronization guarantees alignment in the receiver. However, in the uncoded design with only the BERT blocks, there is a possibility with certain bit and byte alignments that the receiver will not synchronize to the data pattern because adjacent bits do not always end up next to each other coming out on the receive side. To fix this, the transmit data going to the transceiver and the receive data coming from the transceiver was bit reversed so that the bits were transmitted most significant bit first. There were many design challenges faced in getting receive data from the transceiver in the correct alignment.

5.2 Stratix GX Development Board Indicators and Settings

The test and control logic shown in Figure 31 uses some dip switches on the Stratix GX development board to control how the testing is conducted. For test setup the switches control the test duration or the number of data words on which the BERT receiver will measure bit error rate and the data pattern to be used in the test. The design was set up to use two different programmable data words in addition to the two PRBS patterns. The first was an alternating 10 pattern and the second was a 0xABCD repeating word. This section describes the meaning and use of some of the indicators and switches used on the Stratix GX development board within the design. A summary of the switch inputs is shown in Table 10. A summary of the LED and numeric display meanings is shown in Table 11.

Table 10 - Stratix GX development board dip switch settings for integration design.

Switch Reference Designator	Switch #	Switch Net Name in Design	Settings
S11	0	gx_dip0_i	SW1 SW0 Bit-Error-Rate-Tester Pattern 0 0 Programmable word 0 1 2^{11}-1 PRBS pattern 1 0 2^{31}-1 PRBS pattern 1 1 Disable pattern generator
	1	gx_dip1_i	
	2	gx_dip2_i	Set programmable word type. 0 = set to an alternating pattern of 1s and 0s 1 = set to preprogrammed data word
	3	gx_dip3_i	SW4 SW3 Test duration 0 0 Short (half a minute) 0 1 Medium (15 minutes) 1 0 Long (several hours) 1 1 Very Long (several days)
	4	gx_dip4_i	
	5	gx_dip5_i	Selects the MSB or LSB of the err_cnt to be displayed. 0 = LSB 1 = MSB
	6	gx_dip6_i	Selects the source for display_cnt<7:0>, the counter value to be displayed on the LED display. 0 = err_cnt (see gx_dip5_i for which bits are selected) 1 = bit_slip_cnt<7:0>
	7	gx_dip7_i	Not used

Table 11 - Stratix GX development board status indicators for integration design.

Indicator Reference Designator	Indicator Description	Output Net Name in Design	Status Net Name in Design	Status Indicated
D10	LED 0	gx_led0_o	sync_detect	Indicates whether the receiver has achieved ECC block synchronization. (On = synchronized)
D11	LED 1	gx_led1_o	sync_loss_status_latch	Indicates whether the BERT receiver has ever lost data pattern synchronization (On = synchronization has been lost)
D12	LED 2	gx_led2_o	err_ovrflw_latch	Indicates whether the error counter has ever overflowed (On = overflow has occurred)
D13	LED 3	gx_led3_o	heartbeat_led	
D14	LED 4	gx_led4_o	sync_status_latch	Indicates whether the BERT receiver has ever achieved data pattern synchronized (On = synchronization has occurred)
D15	LED 5	gx_led5_o	disp_ovrflw_latch	Indicates whether the MRL encoder block has ever indicated that it had a running disparity overflow (On = running disparity overflow occurred)
D9	7 Segment display first digit	gx_dig_1a_o	display_cnt(7:4) or display_cnt(15:12)	Most significant nibble of bit-error-rate-tester error count or count of the number of bit slips in hex
		gx_dig_1b_o		
		gx_dig_1c_o		
		gx_dig_1d_o		
		gx_dig_1e_o		
		gx_dig_1f_o		
		gx_dig_1g_o		
	7 Segment display first period	gx_dig_1dp_o	sync_status	Bit-error-rate-test receiver synchronization status (On = synchronized)
	7 Segment display second digit	gx_dig_2a_o	display_cnt(3:0) or display_cnt(11:8)	Least significant nibble of bit error rate tester error count or count of the number of bit slips in hex
		gx_dig_2b_o		
		gx_dig_2c_o		
		gx_dig_2d_o		
		gx_dig_2e_o		
		gx_dig_2f_o		
		gx_dig_2g_o		
	7 Segment display second period	gx_dig_2dp_o	test_done	Indicates whether the BERT test has reached the maximum word count and is done. (On = test done)

5.3 Test Results

Testing was performed using the Altera Stratix GX development board with the design described in section 5.1. Two versions of this design were created, one exactly as described and another containing only the BERT pattern generator and receiver and not the coding blocks. The reference clock frequency on the development board and the PLL multipliers available in the Stratix GX device allowed for transmission rates of 2.5 Gbps and 3.125 Gbps. The testing was performed using 3.125 Gbps as a coded-data bit rate for the coded data and 2.5 Gbps for the uncoded data. The ratio of these two rates represents a reasonable approximation of the code rate of the ECC used in this project (actual code rate is 0.75 while this ratio is 0.8). Therefore, on both cases the rate of transfer of input data is approximately the same (2.34375 Gbps versus 2.5 Gbps). It would be better to compare coded to uncoded transfers with transmission rates having the 0.75 ratio, and therefore both transferring input data at the same rate, but that was not easily accomplished in the test setup.

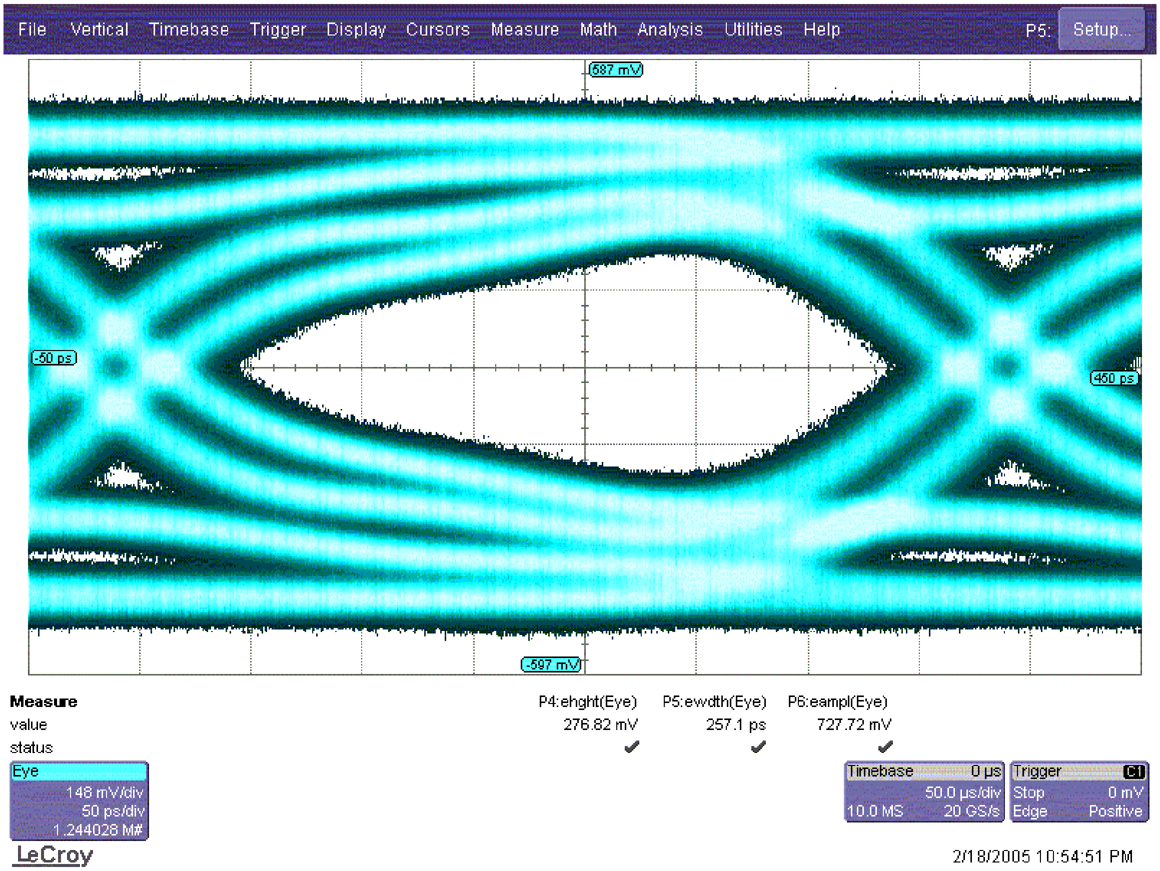
Testing was performed using four different data patterns for both the coded and uncoded designs. Those data patterns were a 16-bit repeating word pattern of 0xABCD, an alternating 1010 pattern, the $2^{11} - 1$ PRBS pattern, and the $2^{31} - 1$ PRBS pattern. A backplane loopback card was plugged into the Stratix GX development board as the transmission channel, with a single transceiver transmitting, so there was minimal crosstalk present during the testing. Two different backplane loopback cards were used, one with 10 in. of trace length and the other with 40 in. of trace length, and bandwidth plots for these were shown in Figure 7 in section 3.1.1. The optimum output voltage setting for the transceiver was determined to be 1 V peak-to-peak differential for the loss in the 40 in. backplane channel and this setting was used for both the 10 in. and the 40 in. backplane tests. Originally the transceiver was set up for no channel equalization, but some experimentation with the equalization settings was done.

A couple of different test durations were used in the BER testing. The first will be referred to as the short test and was set to test $6.87 * 10^{10}$ information bits for the uncoded data (test duration of about 0.46 minutes) and $6.44 * 10^{10}$ information bits for the coded data (test duration of about 0.46 minutes). This test would be expected to count over 60 bit errors for bit error rates around 10^{-9}, providing a good statistical confidence at this bit error rate. The second duration will be referred to as the long test and was set to test $5.28 * 10^{13}$ bits for the uncoded data (test duration of 5.87 hours) and $5.28 * 10^{13}$ bits for the coded data (test duration of 6.26 hours). This test would be expected to result in over 50 bit errors for bit error rates around 10^{-12}, providing a good statistical confidence at this bit error rate. If the long test completed error free, there is 95% confidence that the bit error rate is below $5.68 * 10^{-14}$.

In addition to BER measurements, some signal quality measurements were also made using a Lecroy SDA6020 Serial Data Analyzer. This is a 6 GHz real-time oscilloscope with some software applications for measuring jitter and bathtub curves. The scope stores 32 M samples per channel and can sample at 20 GS/s. A solder-in probe with a bandwidth of 7 GHz was used for the measurements (part number D600ST-S1). The following sections describe some actual test results based on the above test setups and test equipment.

5.3.1 BER for 10 in. Backplane Channel Uncoded Data

Testing was performed with uncoded data being transferred at 2.5 Gbps through the 10 in. backplane. Multiple short tests were performed with each data pattern. Data pattern synchronization always occurred, and there were never any bit errors. A long test was performed using the $2^{11} – 1$ PRBS pattern, and there were no errors, meaning that the bit error rate was below $5.86 * 10^{-14}$ with a 95% confidence. An eye diagram was measured for for the $2^{11} – 1$ PRBS pattern and is shown in Figure 35.



Measure	P4:ehght(Eye)	P5:ewdth(Eye)	P6:eampl(Eye)
value	276.82 mV	257.1 ps	727.72 mV
status	✓	✓	✓

Eye
148 mV/div
50 ps/div
1.244028 M#

Timebase 0 µs
50.0 µs/div
10.0 MS 20 GS/s

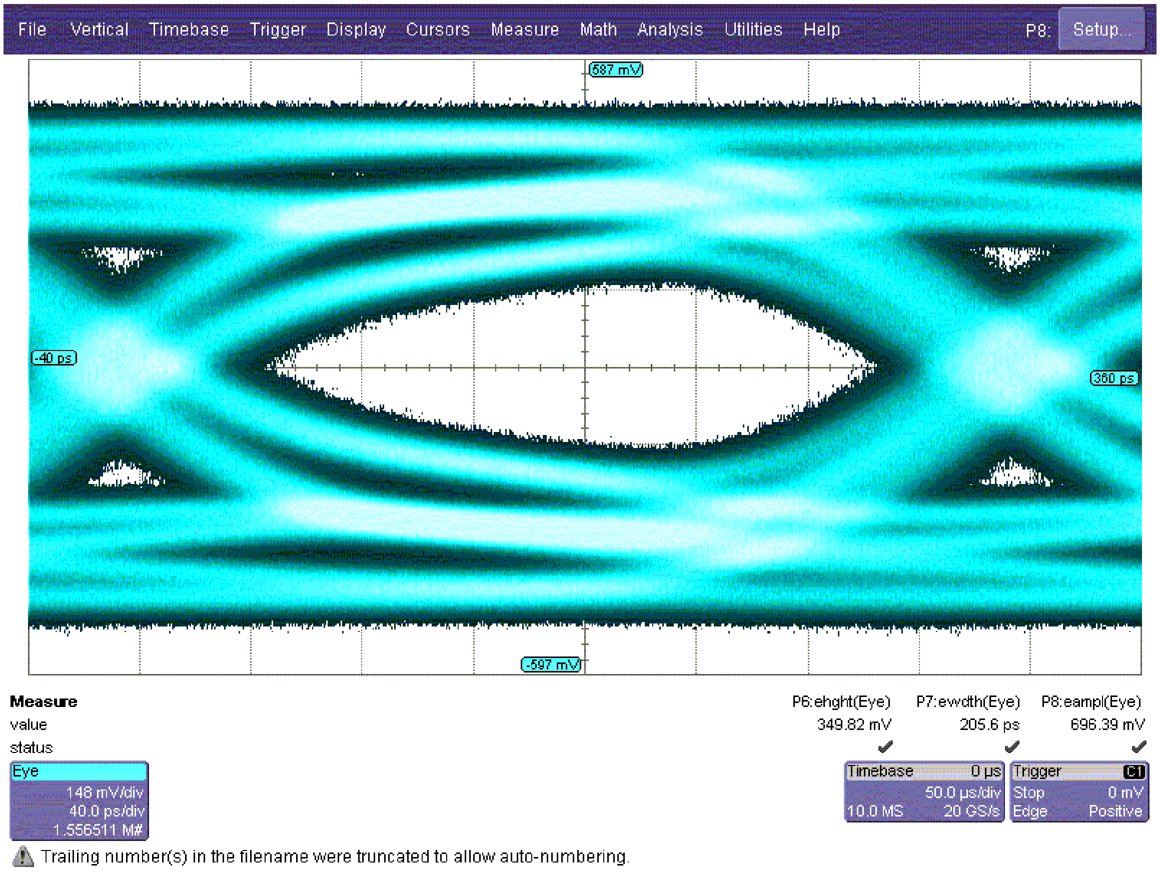
Trigger C1
Stop 0 mV
Edge Positive

2/18/2005 10:54:51 PM

Figure 35 - Uncoded data eye diagram, 2.5 Gbps, 2^{11} - 1 PRBS pattern, 10 in. backplane (oscilloscope screen capture).

5.3.2 BER for 10 in. Backplane Channel Coded Data

Testing was performed with coded data being transferred at 3.125 Gbps (or an information rate of 2.34375 Gbps) through the 10 in. backplane. ECC block synchronization and data pattern synchronization occurred for all four data patterns tested and there were never any bit errors measured in the short test duration. Multiple tests were performed with each pattern. A long test was performed using the $2^{11} - 1$ PRBS pattern, and there were no errors, meaning that the bit error rate was below $5.86 * 10^{-14}$ with a 95% confidence. An eye diagram was measured for the $2^{11} - 1$ PRBS pattern and is shown in Figure 36.



Measure	P6:ehght(Eye)	P7:ewdth(Eye)	P8:eampl(Eye)
value	349.82 mV	205.6 ps	696.39 mV
status	✓	✓	✓

Eye
148 mV/div
40.0 ps/div
1.556511 M#

Timebase 0 µs
50.0 µs/div
10.0 MS 20 GS/s

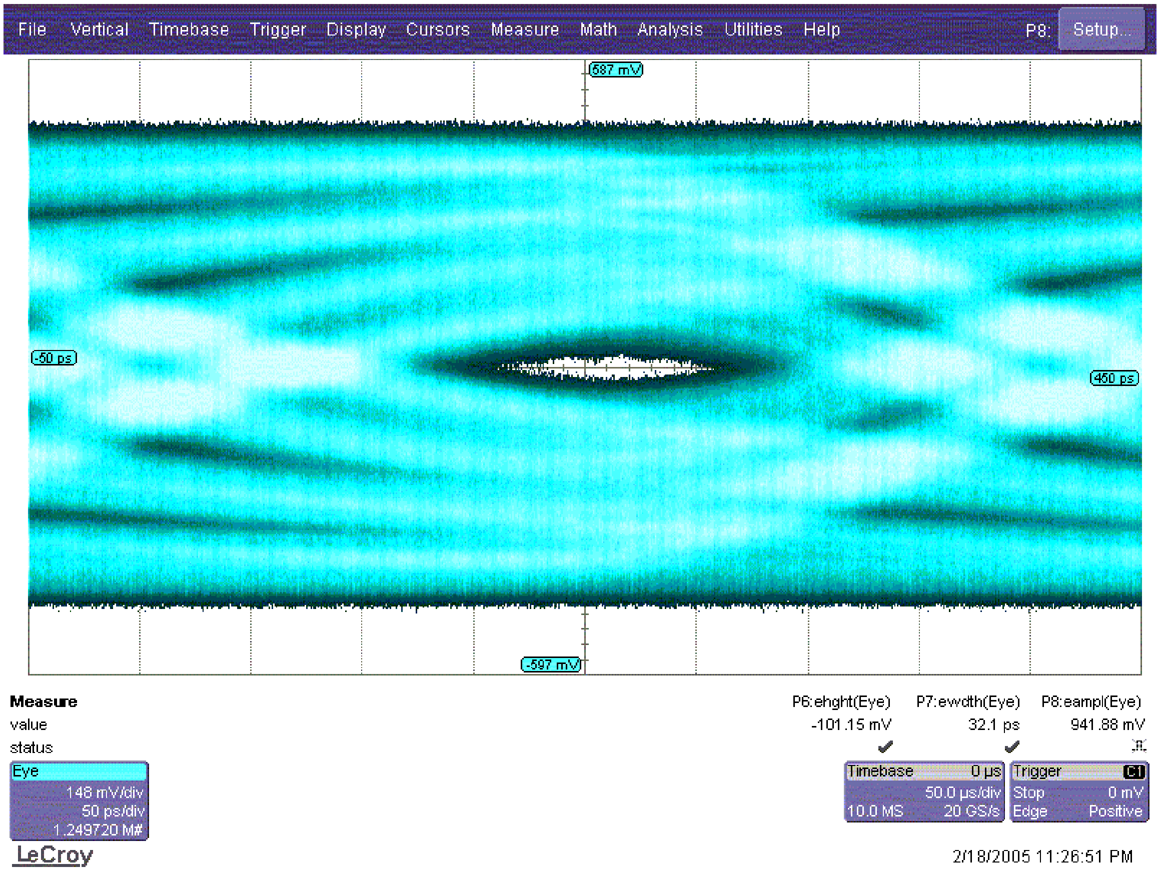
Trigger C1
Stop 0 mV
Edge Positive

⚠ Trailing number(s) in the filename were truncated to allow auto-numbering.

Figure 36 - Coded data eye diagram, 3.125 Gbps, 2^{11} - 1 PRBS pattern, 10 in. backplane (oscilloscope screen capture).

5.3.3 BER for 40 in. Backplane Channel Uncoded Data

Testing was performed with uncoded data being transferred at 2.5 Gbps through the 40 in. backplane. Multiple short tests were performed with each data pattern, and data pattern synchronization always occurred. For the alternating 1010 pattern and the repeating data word 0xABCD pattern, there were never any bit errors. For the $2^{31} – 1$ PRBS pattern the error counter overflowed and the BERT receiver lost synchronization throughout the test. The bit error rate for this pattern had to be better than $1.53 * 10^{-5}$ but it was much larger than the target bit error rate for the short test measurement, equal to 10^{-9}. The behavior for the $2^{31} – 1$ PRBS pattern was very consistent; however, the behavior for the $2^{11} – 1$ PRBS pattern was not. For the latter pattern, synchronization always occurred and was never lost during a test, but the bit error rate varied from test to test. The variance can be classified into three categories: a low bit error rate in which only 1 to 5 errors were counted, a medium bit error rate when many errors were counted, and a very high bit error rate where the error counter overflowed many times. The range of errors measured that can be classified as medium varied a lot as well. The lowest recorded error count was 609 and the highest was 88029 corresponding to error rates of $8.86 * 10^{-9}$ and $1.28 * 10^{-6}$ respectively. To rule out differences in the signal quality from test to test being the cause of the bit error rate difference, eye diagram measurements were made for the cases where medium bit error rate was observed and low bit error rate was observed. The eye diagrams were basically identical. A reasonable explanation for the variation is that the transceiver is synchronizing onto the data and centering its sample point at a different location from test to test. Depending on the location of the sample point, the error rate would vary. It’s not entirely clear why there seems to be three distinct error count classifications and not a more continuous distribution of error counts. Everything points to something inside the transceiver as being the cause of this since (a) nothing like this ever occurs on the 10 in. backplane channel, indicating that the logic is not the cause, and (b) the signal quality is the same, so the channel is not the cause. An eye diagram was measured for for the 2^{11} – 1 PRBS pattern and is shown in Figure 37. This eye diagram clearly illustrates why there are bit errors, since it is nearly closed.



Measure	P6:ehght(Eye)	P7:ewdth(Eye)	P8:eampl(Eye)
value	-101.15 mV	32.1 ps	941.88 mV
status	✓	✓	

Eye
148 mV/div
50 ps/div
1.249720 M#

Timebase 0 µs
50.0 µs/div
10.0 MS 20 GS/s

Trigger C1
Stop 0 mV
Edge Positive

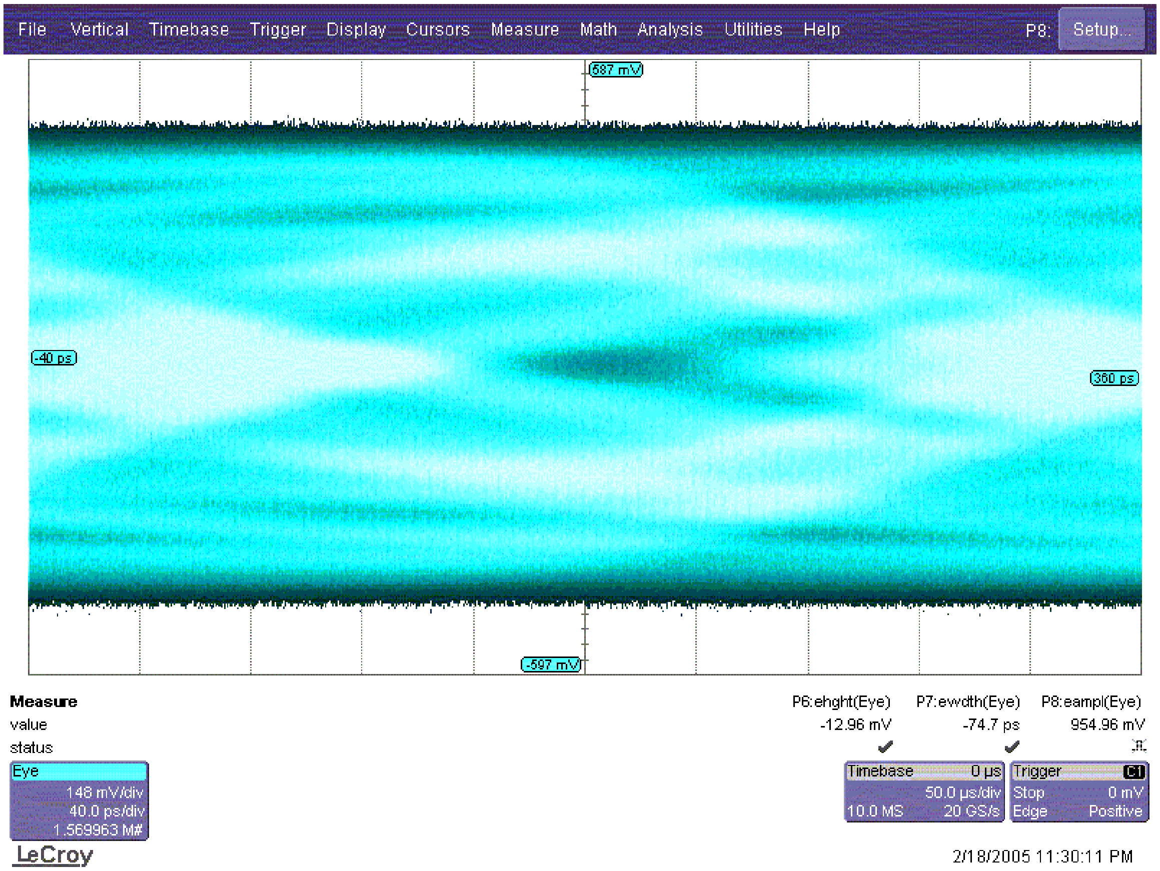
2/18/2005 11:26:51 PM

Figure 37 - Uncoded data eye diagram, 2.5 Gbps, 2^{11} - 1 PRBS pattern, 40 in. backplane (oscilloscope screen capture).

5.3.4 BER for 40 in. Backplane Channel Coded Data

Testing was performed with coded data being transferred at 3.125 Gbps (or an information rate of 2.34375 Gbps) through the 40 in. backplane. For the alternating 1010 data pattern, ECC block synchronization occurs within the first pass, data pattern synchronization occurs and there were never any bit errors measured over multiple short tests. For the repeating 0xABCD data word pattern, ECC synchronization never occurred, so no error count or data pattern synchronization was determined. For the $2^{11} - 1$ PRBS pattern, ECC block synchronization always occurred but never on the first pass. Data pattern synchronization never occurred for this pattern so no bit error count was available. The results for the $2^{31} - 1$ PRBS pattern were exactly the same as for the $2^{11} - 1$ PRBS pattern; ECC block synchronization occurred after multiple passes but data pattern synchronization never occurred. Clearly, the increased transmission rate required to send the code bits makes the bit error rate worse than without coding for the 40 in. backplane channel. An eye diagram was measured for the $2^{11} - 1$ PRBS pattern and is shown in Figure 38. This eye diagram clearly illustrates why data pattern synchronization wasn’t possible, since it is completely closed.

Since it was clear that there was no coding gain at the code rate of the ECC design in this project for the 40 in. backplane channel, an interesting question is, would there be a coding gain at any code rate? To test this, the coded design was changed to operate at a transmitted rate of 2.5 Gbps, the same transmitted rate as was used with the uncoded system. The coded design now has an information rate of 1.875 Gbps. Multiple tests were conducted with each data pattern in this setup. ECC block synchronization always occurred on the first pass and data pattern synchronization also always occurred. There were never any bit errors for the alternating 1010 pattern, the repeating 0xABCD word pattern, and the $2^{11} - 1$ PRBS pattern for the short test. For the $2^{31} - 1$ PRBS pattern there was a varying bit error count from 649 to 21551, corresponding to bit error rates from 1.01 * 10^{-8} to 3.34 * 10^{-7}, respectively. This performance is clearly an improvement over that of the uncoded data at 2.5 Gbps through the 40 in. backplane. The implication is that a code having a code rate higher than that of the tested code (i.e., 0.75), but having similar error correction capabilities and similar MRL characteristics, if it exists, may be effective for the 40 in. backplane channel.



```
Measure                                   P6:ehght(Eye)  P7:ewdth(Eye)  P8:eampl(Eye)
value                                       -12.96 mV       -74.7 ps      954.96 mV
status                                          ✓              ✓
Eye                                       Timebase      0 µs  Trigger          C1
        148 mV/div                                50.0 µs/div  Stop           0 mV
        40.0 ps/div                       10.0 MS      20 GS/s Edge       Positive
        1.569963 M#
                                                     2/18/2005 11:30:11 PM
```

Figure 38 - Coded data eye diagram, 3.125 Gbps, 2^{11} - 1 PRBS pattern, 40 in. backplane (oscilloscope screen capture).

It is also interesting to note that over 25 short tests were performed using the 2^{11} -1 PRBS pattern and there were never any bit errors detected. This is interesting because at the same transmitted rate of 2.5 Gbps, the uncoded data had the three modes of operation, low, medium, and high bit error rate. It seemed reasonable to expect that with this coded system the high bit error rate mode would present itself at least occasionally and result in some bit errors at 2.5 Gbps. The 2^{31} -1 PRBS pattern with the coding also did not exhibit the different modes of operation, all the measurements could be classified as medium bit error rates.

5.3.5 Equalization with Coded and Uncoded Data in the 40 in. Backplane

Some additional testing was done on the 40 in. backplane channel with the 2.5 Gbps uncoded data and the 3.125 Gbps coded data. The receiver within the transceiver has an equalizer built in that can be set to a range of equalization levels from 0 (off) to 4 (maximum equalization). Experimentation revealed that a setting of 2 for the coded data at 3.125 Gbps was sufficient to make the errors go away. In multiple tests with each of the 4 data patterns for the coded data at 3.125 Gbps, ECC block synchronization always occurred, data pattern synchronization always occurred, and there were never any bit errors. A long test was also performed with the $2^{11} - 1$ PRBS pattern and it completed error free indicating a 95% confidence that the bit error rate is better than $5.86 * 10^{-14}$. The same equalization setting of 2 was used on the 2.5 Gbps uncoded data with the 40 in. backplane with exactly the same results; data pattern synchronization always occurred, there were never any bit errors on the short test, and the long test with the $2^{11} - 1$ PRBS pattern completed error free. It is not clear from the results of this test if the bit error rate was improved with the use of coding and equalization.

5.3.6 BER Bathtub Curves and Coding Gain for 10 in. Backplane Channel

The bathtub curve model of bit error rate versus sample position described in section 3.4.4 can be used to evaluate the improvement due to the addition of error-correction coding, for an ECC design. This is being referred to as an improvement due to the addition of error-correction coding, instead of a coding gain, because it is being determined for a system for which coding is added without increasing the transmission rate in the channel. In such a case the rate at which input data is transferred will be reduced. For the situation where the coded system has a transmission rate in the channel that is increased to keep the transfer rate of input data at a constant, an analyzed improvement is referred to as a coding gain. The improvement due to the addition of error-correction coding can still be a useful measure to evaluate whether a given code rate code will be effective if the change in jitter from one transmission rate to another can be estimated. The uncoded bit error rate can be converted into a coded bit error rate using (36) [35], where

$$BER_{coded} \leq \sum_{i=t+1}^{n} \binom{n}{i} p^{i} (1-p)^{n-i} \ . \qquad (\ 36\ )$$

In (36) n is the code word size (63 bits for the ECC design in this project), t is the number of bit errors that can be corrected per code word (2 for the ECC design in this project), and p is the transition probability of a binary symmetric channel (BSC) or the uncoded probability of a bit error. Equation (36) was derived from an upper bound on code word error probability [35], and assumes that for every code word error that occurs, all 48 data bits in the code word will be in error. This assumption is overly pessimistic and therefore weakens the upper bound on the amount of improvement due to the addition of ECC. In this design, where an ECC code is used with an MRL code, if uncorrected bit errors occur in the MRL code bit positions, the MRL decoder could incorrectly invert or not invert the data bits causing most of the bits to then be in error. A better upper bound could be placed on the improvement due to the addition of the ECC in this design by calculating the likelihood that a code word error would result in errors in the MRL bit positions and using that to determine the average number of bit errors per code word error. Equation (36) could then be modified by including an additional factor equal to the average fraction of data bits in error in each code word that has errors. Another equation giving an approximation of the improvement in bit error rate due to the addition of an ECC is shown in (37) [22], where

$$BER_{coded} \approx \left[\sum_{i=t+1}^{n} i \cdot \binom{n}{i} p^{i} (1-p)^{n-i} \right] \times \frac{1}{n}. \qquad (\ 37\ )$$

Equation (37) does not assume any increase in bit errors by subsequent MRL decoding after the ECC decoder, so it is a bit optimistic in estimating the improvement in bit error rate due to the addition of an ECC. Equation (36) has been used for analysis in this project, but a comparison was made between (36) and (37) to show how much difference there is. The true improvement in bit error rate due to the addition of the ECC in this design is expected to be between these two.

Using (36) with the XAUI bathtub curve shown in Figure 15 in section 3.4.4 and the ECC design in this project results in a plot of the bit error rate versus receiver sample time for a system with the addition of coding. Plots of the XAUI bathtub curve (that is, the bathtub curve derived from the XAUI standard) and the XAUI bathtub curve with an improvement due to the addition of ECC, based on (36), are shown in Figure 39.

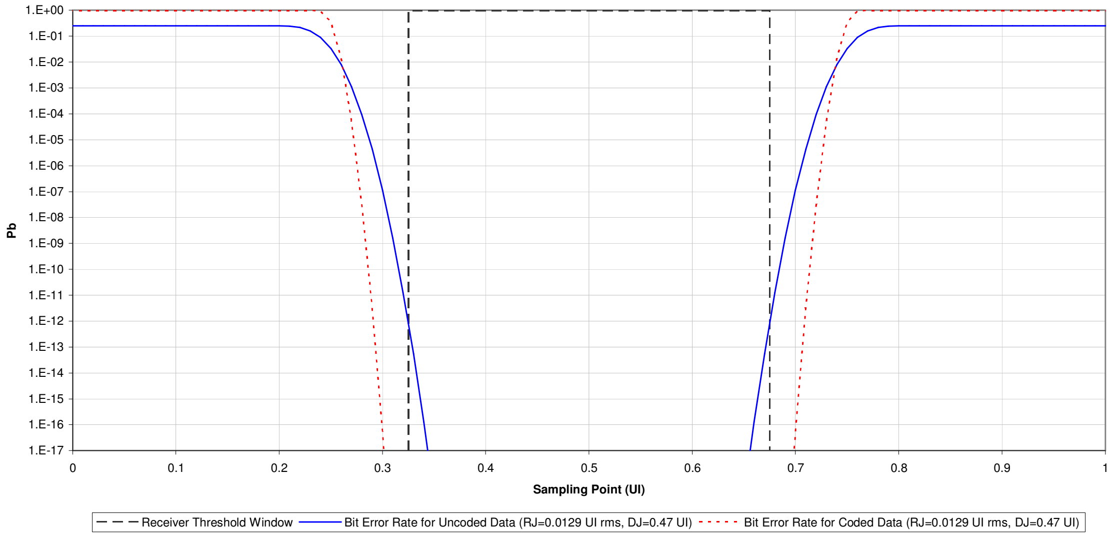


Figure 39 - Coding gain bathtub curves for system with XAUI jitter parameters.

The improvement due to the addition of coding, shown in Figure 39, at a bit error rate of 10^{-12}, is about 0.0335 UI of jitter per side of the bathtub or a total of 0.0671 UI of jitter. For a system operating at the same bit error rate of 10^{-12} but with all random jitter and no deterministic jitter (random jitter of 0.0464 UI rms) the improvement due to the addition of coding is 0.119 UI of jitter per side of the bathtub or a total of 0.237 UI of jitter. So the actual theoretical improvement due to the addition of coding for this system should be somewhere in the range between 0.0671 UI and 0.237 UI of jitter depending on the characteristics of the jitter in the system. The improvement due to the addition of coding using (37) was analyzed for comparison purposes and was found to be 0.0731 for the XAUI jitter case, and 0.258 for the all random jitter case. Equation (37) is more optimistic than (36) by 0.006 UI for the XAUI jitter case and 0.021 UI for the all random jitter case. It is interesting to note that the code is more effective on systems with random noise than systems with both random and deterministic noise.

This bathtub curve methodology was used to evaluate the effectiveness of the coding in the 10 in. backplane system since the BER measurement results were not conclusive. The Lecroy SDA6020 was used to measure the bathtub curve for the transmitted data at 3.125 Gbps (the rate used in the channel with coding) and for the transmitted data at 2.5 Gbps (the rate used in the uncoded channel), in both cases using the $2^{11} - 1$ PRBS data pattern. The scope results are best when many repeats of a pattern can be captured in the sample memory for analysis and this is not possible with the longer $2^{31} - 1$ PRBS pattern. Equation (36) was then applied to the 3.125 Gbps bathtub curve to obtain a bathtub curve applicable to the information or input data for the system with ECC, and the three curves were compared, as shown in Figure 40.

Based on Figure 40, the improvement due to the addition of ECC at a channel transmission rate of 3.125 Gbps and at a bit error rate of 10^{-12} is 0.096 UI on the left side and 0.072 UI on the right side for a total improvement of 0.168 UI. This is just about in the middle of the theoretical range of improvements between 0.0671 UI and 0.237 UI. Also, based on Figure 40, the coding gain between the coded data at a transmission rate of 3.125 Gbps (or an information rate of 2.34375 Gbps) and the uncoded data at 2.5 Gbps is 0.035 UI on the left side and 0.037 UI on the right side for a total coding gain of 0.072 UI. This indicates that for the 10 in. channel, the error correction coding is likely to be effective at lowering the bit error rate. The channel is already operating well below the target bit error rate of 10^{-12} without the coding however.

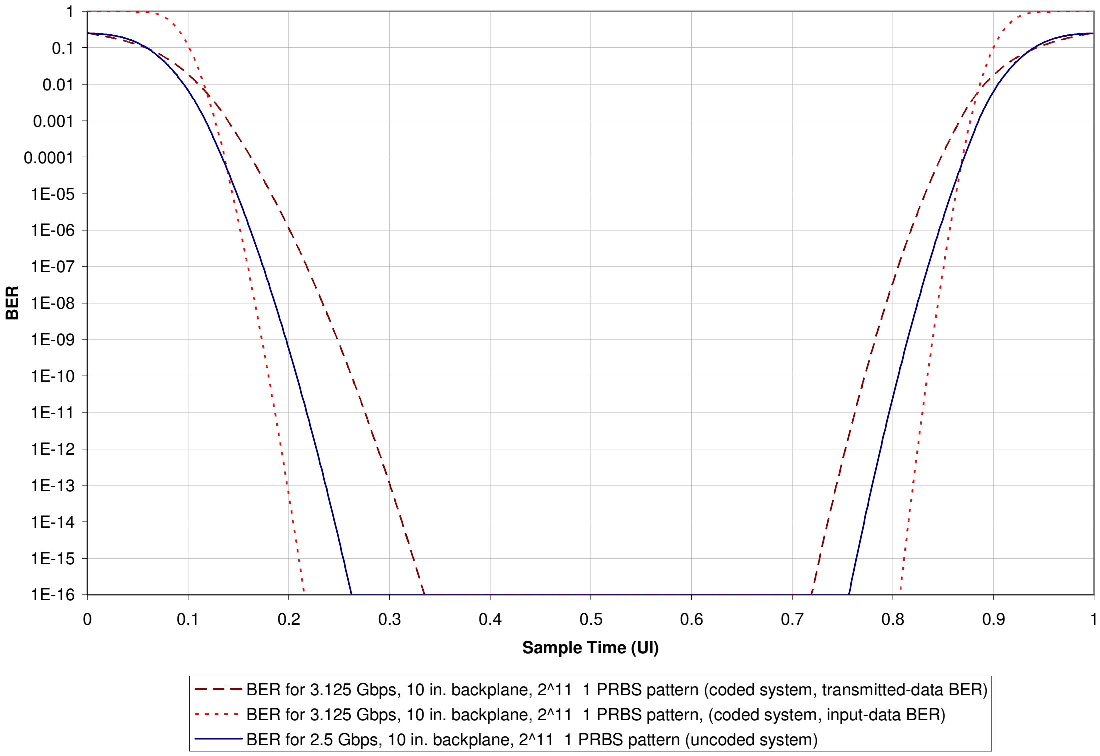


Figure 40 - Coding gain for 10 in. backplane channel.

6 Conclusions and Future Work

An ECC block and a BERT block have been successfully designed, implemented and tested. The ECC block has a code rate of 0.75, meets the MRL requirements of the transceivers and transmission channel, and has a decoding latency of 4 parallel clock cycles or 81.92 ns at 3.125 Gbps. The implemented logic was able to function with a data rate up to 4.34 Gbps, well beyond the capabilities of the transceivers in the Altera Stratix GX. An unexpected benefit of the error correction code was the ease with which ECC block synchronization could be achieved by simply moving the bit alignment until the ECC decode block indicated that there were no errors. This algorithm is no more complicated than the synchronization algorithms currently used in 8B/10B coding. The chosen ECC code is a BCH code that can be implemented with reasonable complexity, and has acceptable decoding latency for applications with multi-gigabit data rates. The BERT block supports three data patterns and is easily configurable and reusable. The performance of the BERT block is sufficient to support 3.125 Gbps data rates in the Altera Stratix GX device.

The coding gain of the ECC block was not sufficient for a 40 in. backplane channel dominated by intersymbol interference. Some additional analysis with a high speed oscilloscope showed that the ECC design was effective on a 10 in. backplane channel. At data rates below 3.125 Gbps, it is not clear if the ECC design would be useful for any real channels. A 10 in. channel does not require error correction to achieve low enough bit error rates. It is clear that a higher code rate is needed for coding to be effective for a 40 in. backplane channel. Determining what code rate is necessary could be done on the current test setup by using a function generator to provide a reference clock to the Altera Stratix GX device. By adjusting the frequency of the reference clock, different data rates could be quickly tried to determine the transmitted data rate at which the current ECC design becomes ineffective, between 2.5 Gbps and 3.125 Gbps. Then 2.5 Gbps, divided by this data rate would be a plausible code rate to further investigate. A new ECC could be designed with the higher code rate and evaluated. Another area for investigation would be different backplane channel lengths between 10 and 40 in. to determine the maximum channel length for which the coding is effective. Perhaps this investigation would also reveal the coding to be more useful for some backplane channel lengths, than it is for the 10 in. backplane.

The strange behavior of the system at 2.5 Gbps without coding in the 40 in. backplane for the $2^{11} - 1$ PRBS pattern is something that could use further investigation. The three different modes of bit error rate performance of low, medium, and high were not seen during the coded data transmission at the same transmission rate. An ECC corrected error counter could be added to the design to count the number of bit errors that have been corrected. This could give some insight into whether the bit errors are still present but are being corrected, or if for some reason due to the data coding alone (MRL coding or additional transitions due to the parity bits) the performance is improved.

The analysis and results indicated that the ECC design is most effective for channels with random noise and is less effective for deterministic noise. As data rates continue to increase, random noise in the form of random jitter will become a larger part of the timing budget. Channel equalization will probably continue to be the first option to combat intersymbol interference, but error correction coding such as that used in this design is also potentially very useful in achieving the target bit error rates.

One thing that was discovered in the project testing was that there is a very strong relationship between data pattern and bit error rate. The longer patterns in most cases seemed to have higher bit error rates. This is definitely an area for more investigation. It indicates that when an error correction code is designed for serial digital multi-gigabit communication systems, the type of data must be taken into account when the code is designed. A specific area of investigation that relates to the pattern length is the MRL coding part of the design. The performance of the MRL code design should be more thoroughly characterized and possibly improved.

The goal of learning about digital communication systems and how they apply to serial digital multi-gigabit communication has been achieved. The communication channel and transceivers have been characterized to determine the factors that are important in code design. Also, the relationship between bit error rate and jitter has been investigated and applied in the project. Some effort has also been spent determining statistical methods to measure bit error rates. A lot of knowledge about error correction codes was also gained by implementing the BCH code, including knowledge about Galois fields, generator polynomials, and error search algorithms.

Another goal of the project was to become familiar with the Altera Stratix GX transceivers. Many obstacles were encountered throughout the implementation phase of the project in working with the transceivers. The most difficult aspects of working with serial data were in making the transformation from serial to wider parallel clock domains and maintaining the proper bit and byte ordering on the receive side. There were several bugs in the logic design that had to be uncovered that related to this, and some were due to misunderstandings about how the Altera Stratix GX transceiver operated. The best example is how the bit slipping operation works within the transceiver. It is actually not a true bit slipping function, but it wraps around on word boundaries.

There are several other areas where some additional investigation could be performed as future projects. One is to evaluate the effects of crosstalk on the bit error rate and see how effective the code design is at dealing with this type of noise. The test setup in this project would be a good starting point since there are four transceivers in the Stratix GX device that connect to the backplane connector. Three aggressor channels could be turned on transmitting BERT data to determine the effects. Another area for investigation would be a more detailed comparison of the performance of the coding design in this project with the 8B/10B coding commonly used in serial standards with multi-gigabit data rates. Some additional work can be done in comparing the error correction capabilities of the code design in this project to the CRC codes commonly used in the serial standards. Enhancements to the code design may be possible that incorporate some additional error detection capabilities to eliminate the need for using CRC codes in conjunction with this coding scheme. Finally, some investigation into how a block code such as that designed in this project can fit into a higher layer protocol would be useful.

7 References

[1] Eiliya, Herman. July / August, 2003. "Grasp the Ins and Outs of High-Speed I/O" *Communications Systems Design* Vol 9 (7), pp. 38-41.

[2] IEEE Computer Society. August 30, 2002. *IEEE Std 802.3ae - 2002 (Amendment to IEEE Std 802.3 - 2002)*. New York, NY: The Institute of Electrical and Electronics Engineers.

[3] Infiniband Trade Association. November 6, 2002. *Infiniband Architecture Specification Volume 1 and 2 Release 1.1*. Portland, OR: Infiniband Trade Association.

[4] PCI-SIG. April 15, 2003. *PCI Express Base Specification, Rev 1.0a*. Portland, OR: PCI Special Interest Group.

[5] RapidIO Trade Association. June, 2002. *RapidIO Interconnect Specification Part VI: Physical Layer 1x/4x LP-Serial Specification*. Austin, TX: RapidIO Trade Association.

[6] Altera Corporation. February 2004. "Altera Stratix GX FPGA Family Datasheet version 2.1." [Internet, WWW, PDF]. *Available*: Available in .PDF format; Address: http://www.altera.com/literature/ds/ds_sgx.pdf. [Accessed: 22 April 2004].

[7] Xilinx Corporation. March 9, 2004. "Xilinx Virtex-II Pro Platform FPGAs: Complete Datasheet version 3.1.1." [Internet, WWW, PDF]. *Available*: Available in .PDF format; Address http://direct.xilinx.com/bvdocs/publications/ds083.pdf. [Accessed: 22 April 2004].

[8] Johnson, Howard and Martin Graham. 2003. *High-Speed Signal Propagation Advanced Black Magic.* New Jersey: Prentice Hall Professional Technical Reference.

[9] Widmer, A. X., P. A. Franaszek. September 1983. "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code" *IBM Journal of Research and Development* Vol. 27 (5), pp. 440-451.

[10] National Committee for Information Technology Standards. November 19, 2002. *Information technology - Fibre Channel - Physical Interfaces (FC-PI).* ANSI INCITS 352-2002.

[11] National Committee for Information Technology Standards. October 22, 2003. *Information Technology - Fibre Channel - Framing and Signaling (FC-FS).* ANSI INCITS 373-2003.

[12] Altera Corporation. 2003. *Altera Stratix GX Development Board Data Sheet version 1.1*. Order Number: DS-STGXDVBD-1.1. San Jose, California: Altera Corporation.

[13] Gray, J. S. 1972. "One Gigabit/Second Signal Processing and Data Handling." *Microwave Symposium Digest, GMTT International*. Vol. 72 (1) pp 192-194.

[14] Bosch, Berthold G. March 1979. "GIGABIT ELECTRONICS - A REVIEW." *Proceedings of the IEEE.* Vol. 67 (3) pp 340-379.

[15] Tektronix. 26 September 2001. "SONET Telecommunications Standard Primer" [Internet, WWW, PDF]. *Available:* Available in .PDF format; *Address:* http://www.tektronix.com/Measurement/App_Notes/SONET/2RW_11407_2.pdf. [Accessed: 24 September 2003]. A copy of this document is available from the author.

[16] "Vitesse's GaAs Multiplexer Handles Telecom Data at 1.25 Gigabit/s" March 31, 1998. *Electronics* Vol. 61 (7), pp. 26.

[17] Walker, Richard C., Thomas Hornak, Chu-Sun Yen, Joseph Doernberg, and Kent H. Springer. June 1991. "A 1.5 Gb/s Link Interface Chipset for Computer Data Transmission" IEEE Journal on Selected Areas in Communications Vol. 9 (5), pp. 698-703.

[18] Menasce, Victor. October 17, 2001. "Breaking Performance Bottlenecks with Rapid I/O" Electronic Engineernig Vol. 73 (897), pp 45-48.

[19] Holden, Brian. "HyperTransport I/O Link Adds Networking Extensions for Next Generation Communications Design." In DesignCon 2003 Conference. 27-30 January 2003. *DesignCon 2003 System-on-Chip and ASIC Design Conference.* International Engineering Consortium.

[20] Strassberg, Dan. November 28 2002. "Testing Gigabit Serial Buses. First Get Physical." *EDN* Vol. 47 (26), pp 53-63.

[21] Shannon, Claude E. July and October 1948. "A Mathematical Theory of Communication." *Bell System Technical Journal* Vol. 27, pp 379-473 and 623-656.

[22] Sklar, Bernard. 2001. *Digital Communications Fundamentals and Applications Second Edition*. New Jersey: Prentice Hall PTR.

[23] Merix Corporation. 2003. *Material Property Comparison*. [Internet, WWW, PDF]. Available: Available in .PDF format; Address: http://www.merix.com/technology.php?section=add&page=secured/pdf/mtl_property_comp.pdf . [Accessed: 25 March 2004]. A copy of this document is available from the author.

[24] Razavi, Behzad. 2001. *Design of Analog CMOS Integrated Circuits*. New York, NY: The McGraw-Hill Companies, Inc.

[25] Franco, Sergio. 2002. *Design with Operational Amplifiers and Analog Integrated Circuits Third Edition*. New York, NY: The McGraw-Hill Companies, Inc.

[26] National Committee for Information Technology Standards. September 1, 1999. *Information Technology - Fibre Channel - Methodologies for Jitter Specification.* INCITS TR-25-1999.

[27] Tyco Electronics. May 4, 2001. "AMP Z-PACK HM-Zd Performance at Gigabit Speeds, Report # 20GC014, Rev. B." [Internet, WWW, PDF]. *Available*: Available in .PDF format; Address: http://hmzd.tycoelectronics.com/documents/electrical_performance.pdf. [Accessed: 16 March 2004].

[28] Ahmad, Bilal. "Performance Specification of Interconnects." In DesignCon 2003 Conference. February 2003. *DesignCon 2003 High-Performance System Design Conference*. International Engineering Consortium.

[29] PCI-SIG. July 24, 2000. *PCI-X Addendum to the PCI Local Bus Specification, Rev 1.0a*. Portland, OR: PCI Special Interest Group.

[30] Ahmad, Bilal and Jeff Cain. "Performance Evaluation of High Speed Serial Links." In DesignCon 2001 Conference. January 29 – February 1, 2001. *DesignCon 2001 Conference Proceedings*. International Engineering Consortium.

[31] Shannon, Claude E. 1949. "Communication in the Presence of Noise." *Proceedings of Institute of Radio Engineers* Vol. 37, pp 10-21.

[32] van Wijngaarden, Adriaan J., and Kees A. Schouhamer. April 2001. "Maximum Runlength-Limited Codes with Error Control Capabilities." *IEEE Journal on Selected Areas in Communications* Vol. 19 (4), pp. 602-611.

[33] Costello, D.J. , J. Hagenauer, H. Imai, and S.B. Wicker. October 1998. “Applications of Error-Control Coding.” IEEE Transactions on Information Theory Vol. 44 (6), pp 2531-2560.

[34] Williams, Dave. February 3, 2000. “Turbo-Product Codes Advance ECC Technology.” *EDN* Vol. 45 (3), pp 77-82.

[35] Lin, Shu and Daniel J. Costello. 2004. *Error Control Coding: Fundamentals and Applications Second Edition*. New Jersey: Prentice Hall.

[36] MacWilliams, F. J. and N. J. A. Sloane. 1977. *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: Elsevier Science B. V.

[37] Coles, Alistair and David Cunningham. October 1998. “Low Overhead Block Coding for Multi-Gb/s Links.” *HP Labs Technical Reports* Extended Enterprise Laboratory HPL-98-168.

[38] Wicker, Stephen B. 1995. *Error Control Systems for Digital Communication and Storage*. New Jersey: Prentice Hall Inc.

[39] Synthesys Research. 2004. “BERTScope 7500A and 12500A Bit Error Rate Analyzer Datasheet.” [Internet, WWW, PDF]. *Available*: Available in .PDF format; Address: http://www.synthesysresearch.com/pdf/BERTScope.pdf. [Accessed: 22 July 2004].

[40] DeCoursey, W. J. 2003. *Statistics and Probability for Engineering Applications with Microsoft Excel*. Massachusetts: Elsevier Science.

[41] Wolaver, Dan H. May 30, 1995. “Measure Error Rates Quickly and Accurately.” *Electronic Design* Vol. 43 (11), pp 89-98.

[42] International Telecommunication Union Telecommunication Standardization Sector. May 1996. *General Requirements for Instrumentation for Performance Measurements on Digital Transmission Equipment*. ITU-T Recommendation O.150.

[43] Coombs, Clyde F. 1999. *Electronic Instrument Handbook Third Edition*. New York: McGraw-Hill.

8 Bibliography

Altera Corporation. 2003. *High-Speed Development Kit, Stratix GX Edition User Guide version 1.0*. Order Number: UG-STRATIXGX-1.0. San Jose, California: Altera Corporation.

d'Abreu, Manuel. 2002. "Noise – Its Sources, and Impact on Design and Test of Mixed Signal Circuits" *Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications.* IEEE Computer Society.

Imai, H. 1990. *Essentials of Error-Control Coding Techniques*. San Diego, CA: Academic.

9 Appendix A – ECC Block Datasheet

The Product Realization Company

Serial Digital Multi Gigabit Communications Block Code – (51,48) MRL and (63,51) BCH

Document Number: 9132.011
EC/Revision: Rev 1.0
Revision Date: March 7, 2005
Author: David Carney

Features:

- Two error correcting capability
- Some three or more error detecting capability
- 48 bit logic side data interface
- 16 bit transceiver side data interface
- Run length limited to 64 bits
- Provides a mechanism for DC balanced data (no guarantee)
- more features

Description:

The core consists of two separate blocks, the encoder and the decoder. The encoder performs a maximum run length limiting on the data followed by parity generation using a (63,51) BCH code. The decoder performs the inverse operation of this. The core is written in VHDL.

Required VHDL Files:

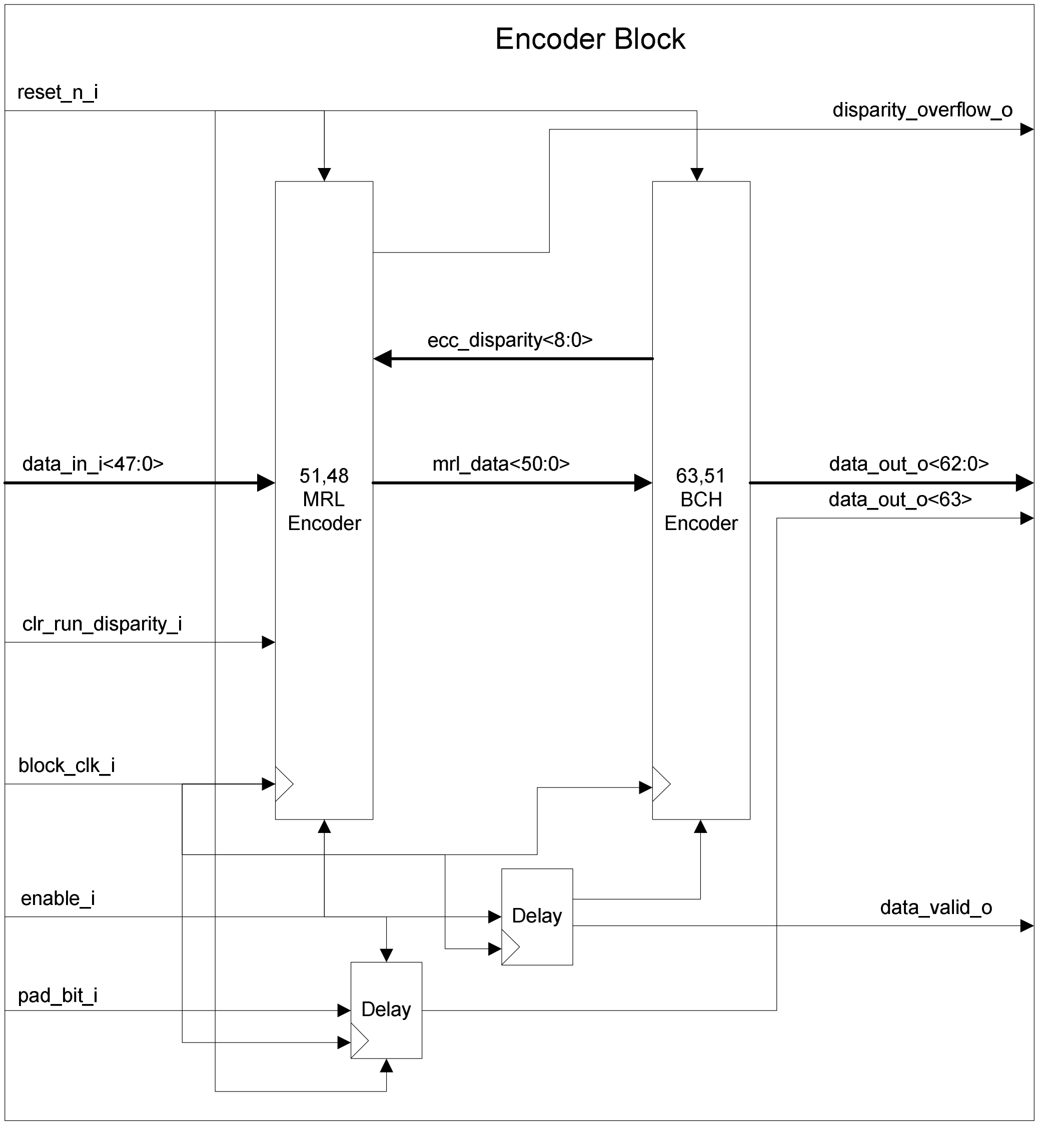
mrl_bch_encoder_64_48.vhd
mrl_51_48_encoder.vhd
bch_63_51_encoder.vhd
bch_mrl_decoder_64_48.vhd
bch_63_51_syndrome.vhd
bch_63_61_error_poly.vhd
gf_2_6_mult.vhd
gf_2_6_inverter.vhd
bch_63_51_error_search.vhd
mrl_51_48_decoder.vhd

Required Libraries:

ieee.std_logic_arith.all
ieee.std_logic_unsigned.all
ieee.std_logic_signed.all
ieee.std_logic_1164.all
gf_2_6_field_pkg.all (gf_2_6_field_pkg.vhd)

Encoder Description

Encoder Functional Diagram:



Encoder Timing:

Block Parameters	Chip	Speed Grade	Package	Area	Speed (MHz)
DISPARITY_REG_SIZE = 9	Altera Stratix GX (EP1SGX25FF1020)	-5	1020 pin BGA	450 LCs 181 LC Registers	67.79

Encoder Parameters:

Generic	Default Value	Description
DISPARITY_REG_SIZE	9	Number of bits that the signed running disparity register contains.

Encoder Input/Output Descriptions:

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
block_clk_i	std_logic	NA	NA	I	Clock used by the block. A new 48 bit data input symbol is valid on every rising edge of this clock. The encoded data rate of the block is 64x this clock.
reset_n_i	std_logic	NA	NA	I	Reset signal for the core (0=reset state, 1=out of reset). This input is asynchronous.
data_in_i	std_logic_vector	47	0	I	Contains the data input symbol. It is synchronous to and valid on the rising edge of the block_clk_i. This comes from the internal core logic source of the data.
pad_bit_i	std_logic	NA	NA	I	Contains a pad bit that is added to the coded bit in data_out_o position 63 for every block output from the encoder. This input must be synchronous to and valid on the rising edge of block_clk_i. The output will be delayed will be delayed by three clock cycles so that it matches with the same data_in_i word that it was input with.
data_out_o	std_logic_vector	63	0	O	Contains the data output symbol. It is synchronous to and output on the block_clk_i. This goes to the transceiver transmit block.
disparity_overflow_o	std_logic	NA	NA	O	A flag to indicate if a running disparity counter overflow has occurred. This output is synchronous to and output on the rising edge of block_clk_i.

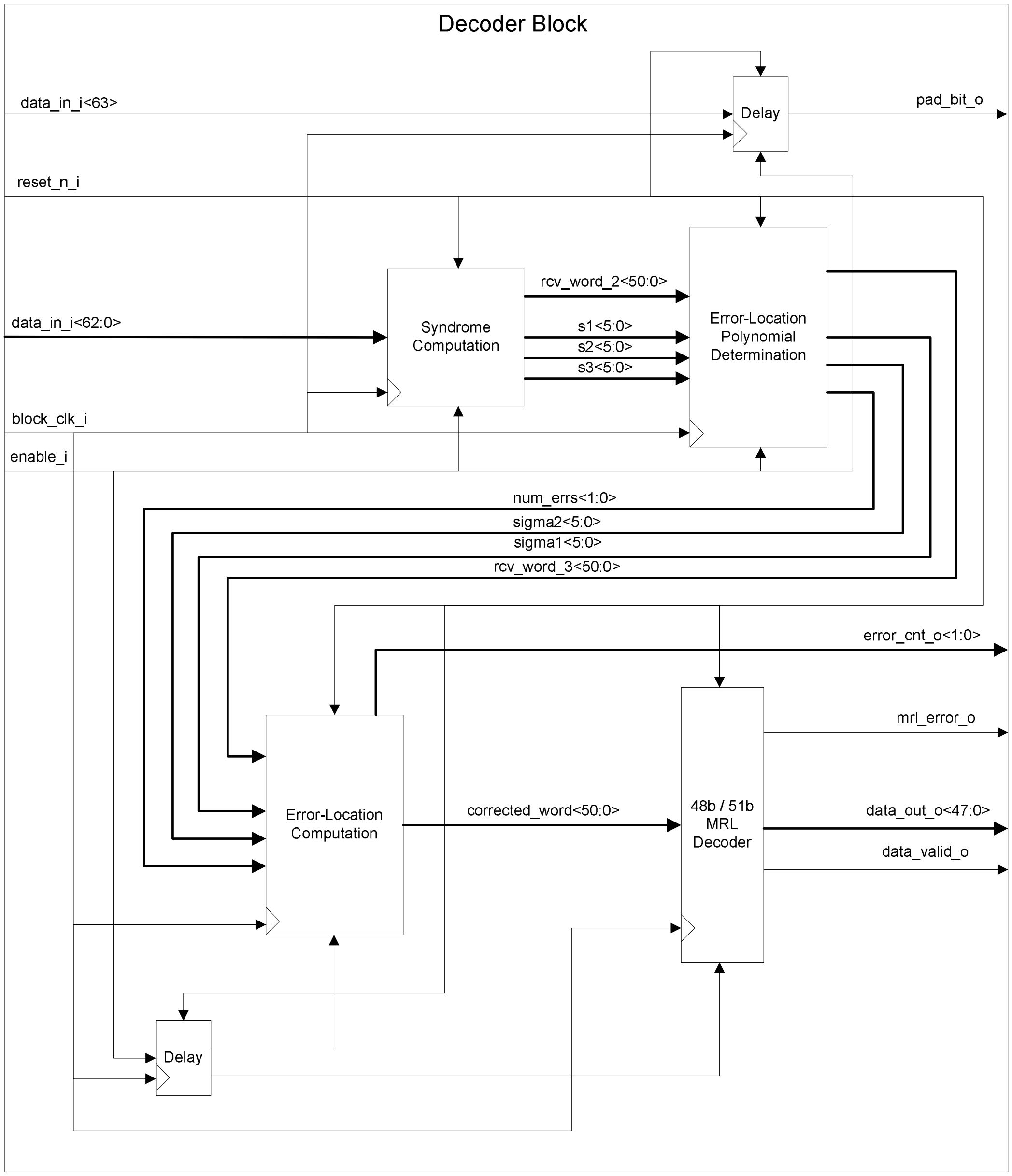
Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
enable_i	std_logic	NA	NA	I	Causes the encoder to be enabled or disabled (1=enabled, 0=disabled). This input is synchronous to and valid on the risinge edge of block_clk_i. In the disabled state the block will be completely inactive and output data will not change. When enable_i transitions from 0 to 1 the block immediately begins processing and outputting data.
clr_run_disparity_i	std_logic	NA	NA	I	Causes the MRL encoder to clear the running disparity register to 0. This input is synchronous to and valid on the rising edge of block_clk_i. If this signal is a 1, the running disparity register will be set to 0 on the next block_clk_i cycle.

Encoder Input Truth Table:

n_reset_i	enable_i	Clr_run_disparity_i	Core State
0	X	X	The encoder is in reset, data_out_o is all zeros. The running disparity counter is reset and all internal data pipeline registers are set to zeros.
1	0	0	Disabled – Core disabled meaning no data is latched in from data_in_i but all other internal registers retain their state and the output data remains in the last valid state.
1	1	0	Enabled – data_out_o will be the encoded version of data_in_i.
1	X	1	The current enabled or disabled state applies to all registers except the running disparity register which will be cleared to 0 on the next clock cycle.

Decoder Description

Decoder Functional Diagram:



Decoder Timing:

Block Parameters	Chip	Speed Grade	Package	Area	Speed (MHz)
NA	Altera Stratix GX (EP1SGX25FF1020)	-5	1020 pin BGA	989 LCs 306 LC Registers	105.33

Decoder Input/Output Descriptions:

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
block_clk_i	std_logic	NA	NA	I	Clock used by the block. A new 48 bit data output symbol is valid on every rising edge of this clock. The decoded data rate of the block is 64x this clock.
reset_n_i	std_logic	NA	NA	I	Reset signal for the core (0=reset state, 1=out of reset). This input is asynchronous.
data_in_i	std_logic_vector	63	0	I	Contains the input data symbol from the transceiver. It is synchronous to and valid on the rising edge of xcvr_clk_i. The data block to be decoded consists of four consecutive data_in_i symbols.
data_out_o	std_logic_vector	47	0	O	Contains the data output symbol. This goes to the internal core logic. This signal is clocked out on the rising edge of block_clk_i.
enable_i	std_logic	NA	NA	I	Causes the decoder to be enabled or disabled (1=enabled, 0=disabled). This input is synchronous to and valid on the risinge edge of block_clk_i. In the disabled state the block will be completely inactive and output data will not change. When enable_i transitions from 0 to 1 the block immediately begins processing and outputting data.
pad_bit_o	std_logic	NA	NA	O	Contains the pad bit stripped off of the input data word before decoding. The output is delayed so that it matches up with the same clock cycle that the corresponding decoded data is output on. This signal is output on the rising edge of block_clk_i.

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
error_cnt_o	std_logic_vector	1	0	O	Contains a count of the number of errors the decoder detected / corrected as ollows: 00 – No errors detected 01 – 1 Error detected and corrected 10 – 2 Errors detected and corrected 11 – 3 or more errors detected This signal is output on the rising edge of block_clk_i.
mrl_error_o	std_logic	NA	NA	O	Contains an indication of whether or not the MRL decoder detected an error pattern in the MRL bits. (1=error, 0=no error). This signal is output on the rising edge of block_clk_i.

Decoder Input Truth Table:

n_reset_in	enable_in	Core State
0	X	The decoder is in reset, data_out_o is all zeros. All internal data pipeline registers are set to zeros.
1	0	Disabled – Core disabled meaning no data is latched in from data_in_i but all other internal registers retain their state and the output data remains in the last valid state.
1	1	Enabled – data_out_o will be the decoded version of data_in_i.

Background

For detailed background information on the use of ECC in serial digital multi gigabit communication systems, refer to section 3.6.

Functional Description

The design consists of an error correct coding scheme coupled with a maximum run length limiting scheme. There are two blocks, the encoder and the decoder. The data word size is 48 bits and there are 15 code bits and 1 pad bit for a total coded word size of 64 bits. The code rate is 0.75. The error correction coding is a 2 error correcting primitive BCH code with a minimum distance of 5. It can detect and correct all 1 and 2 bit error patterns in the 63 code bits (the pad bit is not protected). It can also detect some 3 or more bit error patterns. The code will actually detect 99.976% of all error patterns, however since the minimum distance is 5 and error correction is used, many error patterns beyond 2 bits will actually result in the code making an invalid correction. The intended application for this error correction coding scheme is in wired serial digital multi-gigabit communication systems where the distribution or errors is random.

Encoding Data Flow

Data comes in to the encoder in 48 bit words synchronous to and valid on the rising edge of block_clk_i. There are two steps to the encoder, MRL encoding and BCH encoding. The operation of these blocks occurs normally unless they are held in reset or they are disabled. Each block completes in 1 block_clk_i clock cycle, so the encoder takes 2 block_clk_i clock cycles to encode data. After the initial 2 clock latency, data is continuously coming out of the encoder.

(51,48) MRL Encoder

After the first clock cycle the data will be encoded with the MRL block and three code bits will be added to the data and the data will either be inverted or not inverted to minimize the running disparity of 1s and 0s. The rules for the encoding are as follows. The MRL bits are labeled as MRL<2:0>.

- If the running disparity is greater than or equal to 0
    - If the disparity of the current 48 bit input data word is greater than or equal to 0, then invert the data bits and set MRL<0,2> to 0
    - Else leave the 48 bit input data word alone and set MRL<0,2> to 1
- Else if the running disparity is less than 0
    - If the disparity of the current 48 bit input data word is greater than or equal to 0, then leave the data bits alone and set MRL<0,2> to 1
    - Else invert the data bits and set MRL<0,2> to 0
- Set MRL<1> to the inverse of MRL<2>

A running disparity register is maintained in the MRL block that is updated on every clock cycle. A positive disparity indicates that more 1s than 0s have occurred and a negative disparity indicates that more 0s than 1s have occurred. The running disparity is based on both the MRL coded data and on the disparity of the BCH parity bits that have been added to the transmitted data after this block. The disparity of the BCH parity bits is fed back to this block.

- At reset Running Disparity is set to 0
- If the data has not been inverted
    - Current Disparity = (1s Count of Input Data) – (48 – (1s Count of Input Data))
    - Running Disparity = (Running Disparity) + (Current Disparity) + 1 + (ECC Disparity)
- If the data has been inverted
    - Current Disparity = (48 – (1s Count of Input Data)) – (1s count of Input Data)
    - Running Disparity = (Running Disparity) + (Current Disparity) -1 + (ECC Disparity)
- If the Running Disparity has overflowed the size of its register in the positive direction, then set it to the maximum register size and set the disparity_overflow_o output bit to 1.
- Else if the Running Disparity has overflowed the size of its register in the negative direction, then set it to the minimum register size and set the disparity_overflow_o output bit to 1.
- Else set the disparity_overflow_o output bit to 0.

This MRL code provides a mechanism to limit the maximum run length of the output data to no more than the encoded block size of 64 bits. It also provides a mechanism for controlling the amount of disparity between 0s and 1s in transmitted data to limit the DC spectra content of the data. There is no guaranteed limit on the running disparity though since the code adds an extra transition bit and downstream the BCH coder adds 12 more unconstrained bits. The running disparity register in the block is bounded to 8 bits in magnitude in either the positive or negative direction so if the running disparity exceeds 255 or -255 the disparity_overflow_o output will be asserted to alert user logic to the problem.

(63,51) BCH Encoder

This block implements the cyclic coding algorithm for a primitive binary (63,51) BCH code on the 51 bit data from the MRL encoder. The block computes the 12 parity bits for the encoded data within a single block_clk_i clock cycle. It also computes the disparity between 1s and 0s in those 12 parity bits and passes that number back to the MRL block for use in maintaining the running disparity within that clock cycle.

Data Format

The output data is valid when the data_valid_o signal is asserted. The format of the 64 bit output data word is shown in the following table.

Bit Position	63	62	61 59	58 35	34 32	31	30 28	27 4	3 1	0
Description	Pad Bit	MRL<2>	C<11:9>	Data<47:24>	C<8:6>	MRL<1>	C<5:3>	Data<23:0>	C<2:0>	MRL<0>

Decoding Data Flow

Data comes in to the decoder in 64 bit words synchronous to and valid on the rising edge of block_clk_i. There are two steps to the decoder, BCH decoding and MRL decoding. The operation of these blocks occurs normally unless they are held in reset or they are disabled. The MRL block completes in 1 block_clk_i clock cycle. The BCH decoder takes 3 block_clk_i clock cycles. The decoder thus takes 4 block_clk_i clock cycles to decode data. After the initial 4 clock latency, data is continuously coming out of the decoder.

Data Format

The input data must be valid on the rising edge of block_clk_i and it must be in the following format.

Bit Position	63	62	61 59	58 35	34 32	31	30 28	27 4	3 1	0
Description	Pad Bit	MRL<2>	C<11:9>	Data<47:24>	C<8:6>	MRL<1>	C<5:3>	Data<23:0>	C<2:0>	MRL<0>

(63,51) Decoder

This step takes 3 block_clk_i clock cycles to complete. The operation is pipelined by the decoding steps which are syndrome computation, error location polynomial calculation, and error search. The output after these three decoding stages is the 51 bit corrected data word and an indication of the number of errors detected or corrected in the data. The error_cnt_o<1:0> value is delayed by one clock cycle so that it matches up with the actual output data word from the decoder which is delayed by one cycle for MRL decoding.

The (63,51) BCH code has a minimum distance of 5, so all error patterns up to 4 bits in length will be detected. However since the code will attempt to correct all 1 and 2 error patterns, there is some chance that some 3 or more input error patterns could be recognized as 2 input error patterns and corrected wrongly. There is no way of knowing if this occurred or not. Some 3 or more input error patterns will be detected correctly. If this code is used in an application where error detection is important, then assume that if error_cnt_o<1:0> is 10 or 11 (2 or 3 or more errors respectively) that there are errors in the received word and do not use it as valid data.

(51,48) MRL Decoder

This stage is very simple. It completes in a single clock cycle and the 48 bit data is output on the rising edge of block_clk_i. The following rules define how this block operates.

- If MRL<2:0> = 010 then invert the data bits to decode them
- Else if MRL<2:0> = 101 then leave the data bits alone to decode them
- Else set the mrl_error_o signal to a 1 to indicate a decoding error

The MRL decoder provides one additional error correction check on the data. If the BCH decoder incorrectly decoded a word but one of the MRL bits was made to be incorrect, the MRL decoder could detect that and mark the entire data word as in error with the mrl_error_o signal.

Synchronization

In order to operate correctly in a communication system, the encoder and decoder must be synchronized. There is no internal synchronization provided in the ECC block and this must be handled with additional external logic. One possible means of synchronizing is waiting a fixed amount of time, checking for normal data reception. If it doesn't occur, stop the decoder for one symbol time and then start it again. Repeat this process until normal data reception occurs. The decoder can be stopped by using the enable_in input signal. Another means would be to use an external framer or synchronizer and only performing the encoding / decoding on the data payload.

Functional Timing

All the registers in the encoder and decoder block are clocked by the rising edge of block_clk_i. All inputs except the asynchronous reset signal (reset_n_i) to the blocks must be valid for the rising edge and all outputs from the blocks are output on the rising edge of this clock. The following table shows how the various control signals for the encoder and decoder should be asserted and when valid output data will be present from each

.

Signals	**Clock Cycle #**								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Common Signals									
reset_n_i	0	1	1	1	1	1	1	1	1
Encoder Signals									
enable_i	0	0	1	1	1	1	1	1	1
data_in_i	X	V	V	V	V	V	V	V	V
pad_bit_i	X	V	V	V	V	V	V	V	V
data_out_o	X	X	X	X	V	V	V	V	V
Decoder Signals									
enable_i	0	0	1	1	1	1	1	1	1
data_in_i	X	V	V	V	V	V	V	V	V
pad_bit_i	X	V	V	V	V	V	V	V	V
data_out_o	X	X	X	X	X	X	V	V	V
pad_bit_o	X	X	X	X	X	X	V	V	V
X = Don't care or invalid data V = Valid data or signal									

The worse case timing path for the encoder is in the MRL encoding block. There is a lot of combinational logic in this block to calculate the running disparity and then make a decision based on the current running disparity and the accumulated running disparity as to which way to encode the data. This process could be fairly easily pipelined to improve performance at the cost of extra latency. The worse case timing path for the decoder is in the error search block. This is a very wide combinational logic block that must test every single error position in the error polynomial and test for 0s by performing multiplication and addition. The block must also then count the number of errors detected. This block could likely be pipelined between the actual error search and error counting fairly easily at the cost of an additional stage of latency. For current generation serial applications the performance of both blocks exceeds the serial data rate that the transceivers support. The encoder can function with a serial data rate of up to 4.34 Gbps and the decoder can function with a serial data rate of up to 6.74 Gbps. It may be possible to remove some of the pipelining in the decoder design since there is so much slack beyond the maximum serial data rate to reduce the latency below 4 clock cycles.

Test Benches

There are block level test benches for the blocks and a top level test bench that thoroughly tests the error correcting capabilities through the use of a bit error rate tester block.

Encoder Block Level Test Benches

There is one block level test bench for the encoder to simulate the functionality of the maximum run length block. This test bench is contained in the file \encoder_block_sims\mrl_51_48_encoder_tb.vhd. This test bench simply provides the clock and control signals, instantiates the MRL block of the encoder and sends a data pattern through. The resulting MRL encoding must be manually checked. A Mathcad program exists that implements the MRL encoding functionality for use in manual checking and is located at \encoder_block_sims\mrl_48_51.mcd.

Decoder Block Level Test Benches

There are two simple decoder block level simulations located at \decoder_block_sims\gf_2_6_inverter_tb.vhd and \decoder_block_sims\gf_2_6_mult_tb.vhd. The first one simply goes through all the possible 6 bit GF(2^6) patterns to force the block to display the inverse which must be checked manually from a Galois Field table. The second one simply tries several different 6 bit GF(2^6) input values to the multiplier to force the block to display the result and then automatically checks that it is correct.

Encoder and Decoder Block Level Simulation

The rest of the block level simulations are contained in a single file located at \encoder_decoder_block_sim\bch_63_51_tb.vhd. This test bench creates the clock and control signals and connects together all of the individual functional blocks of the encoder and decoder. It does not use the top level vhdl files for either the encoder or decoder, but instead replicates that connectivity. The test bench then inserts data into the encoder and also inserts errors into the data between the encoder and decoder. The following things are automatically checked throughout the data path.

- The encoder parity bit generation
- The decoder syndrome computation
- The decoder error polynomial coefficient calculation
- The decoder output data
- The decoder output error count

The expected values hard coded in the test bench for all of these tests were manually calculated using a Mathcad program model of the encoder and decoder.

High Level Bit Error Rate Test Simulation

The top level encoder and decoder blocks are instantiated and connected together in a single test bench along with a bit error rate test pattern generator and bit error rate test receiver. This test bench is contained in \encoder_decoder_bert_sim\encoder_decoder_bert_tb.vhd. The BERT pattern generator and receiver blocks are other Plexus building blocks that have the ability to send and receive and count bit errors for three different patterns including a $2^{11} - 1$ PRBS, a $2^{31} - 1$ PRBS, and a programmable data word pattern. The purpose of this test bench is to give an overall stress to the encoder and decoder blocks and to gather some information about its performance. The test bench connects the BERT pattern generator to the encoder, then connects the encoder output data with or without errors inserted to the decoder, and then connects the decoder output to the BERT receiver. The test bench also creates the necessary clock and control signals. The following tests and self checks are performed.

- All single and double bit error patterns are inserted into the receive data
- After single and double bit error insertion, the test bench automatically checks to be sure that the BERT receiver did not detect any errors (they were all corrected)
- All three bit error patterns are inserted into the receive data
- The number of three bit error patterns that are inserted is counted (err_3_pattern_cnt)
- The number of three bit error patterns that are identified as three or more bit error patterns is counted (err_3_detected_cnt)
- The receive pad bit is checked versus what is sent into the block
- The fact that bit errors are counted is checked after the three bit error pattern insertion is done (just that there were errors, not the actual number is checked)
- The disparity_overflow_o flag is checked to make sure that it does not get asserted

All self checks in the test bench are done with assertions and ERROR or WARNING messages will be printed in the simulation transcript window if any errors occur.

At the beginning of this test bench file is a package that contains constant definitions that control how some of the tests work. The following table describes those constants.

Constant	**Description**	**Default Value**	**Stresses**
BLOCK_CLK_PERIOD	The clock period for the word clock (not important for functional sims)	20 ns	
ERR_CNT_SIZE	These constants map directly to generics in the BERT pattern generator and pattern receiver	10	
SYNC_LOSS_CNT_SIZE		8	
WORD_CNT_SIZE		64	
SYNC_ERR_CNT		12	
UNSYNC_ERR_CNT		1228	
SYNC_WORDS		256	
INVERT_DATA	Specifies the state of the invert_data_i signal to the BERT pattern generator and receiver (1=invert data pattern, 0=don’t invert)	0	
DATA_WORD	The constant data word to use in the word pattern test	A5BFC8E14729	Try patterns that will cause the MRL running disparity register to overflow.
PRBS_2_11_SEED	The value to be connected to prbs_2_11_seed_i of the BERT pattern generator	00000000001	

Constant	**Description**	**Default Value**	**Stresses**
PRBS_2_31_SEED	The value to be connected to prbs_2_31_seed_i of the BERT pattern generator	00000000000000000000000000000001	
TEST_DURATION	The value to be connected to the max_word_cnt_i signal on the BERT receiver	0x270000	
TIME_BETWEEN_ERRORS	The number of clock cycles to wait between error insertions	10	
BERT_PATTERN	Selects which pattern to the BERT blocks should use	00 for Word, 01 for PRBS $2^{11} – 1$, or 10 for $2^{31} – 1$	
DISPARITY_REG_SIZE	Sets the size of the running disparity register in the MRL encoder.	9	Set this to a smaller value such as 5 so that disparity overflows will occur.

10 Appendix B – BERT Block Datasheet

Bit Error Rate Test Block Datasheet

PLEXUS®
The Product Realization Company

Document Number: 9132.041
EC/Revision: Rev 1.1
Revision Date: March 7, 2005
Author: David Carney

Features:

- Supports variable length data word sizes from 8 bits to 64 bits
- Supports $2^{11} – 1$ and $2^{31} -1$ PRBS data patterns
- Supports constant data word data pattern
- Receive block automatically synchronizes to the data pattern and then counts bit errors

Description:

The core consists of two separate blocks, the BERT pattern generator and the BERT receiver. The pattern generator provides a variable width output data word based on the selected data pattern. The receiver synchronizes to the selected data pattern and once synchronized counts bit errors between the received data and the expected data. The core is written in VHDL.

Required VHDL Files Hierarchy:

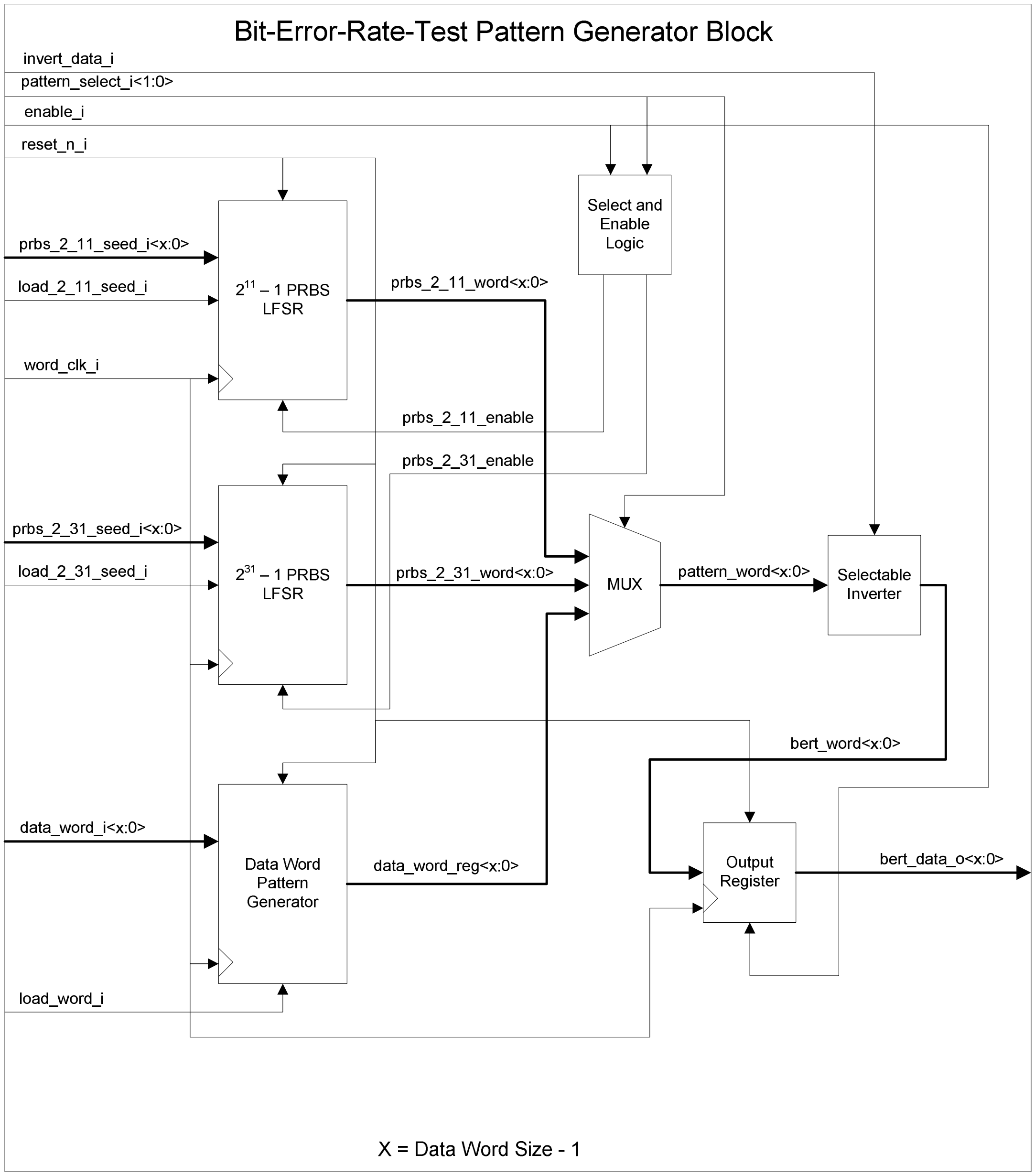
- bert_pattern_gen.vhd
  - prbs_generator.vhd
- bert_receiver.vhd
  - bert_receiver_control.vhd
  - prbs_generator.vhd
  - word_compare.vhd

Required Libraries:

use ieee.std_logic_arith.all; (standard VHDL library)
use ieee.std_logic_unsigned.all; (standard VHDL library)
use ieee.std_logic_1164.all; (standard VHDL library)

Pattern Generator Description

Pattern Generator Functional Diagram:



Pattern Generator Timing:

Block Parameters	Chip	Speed Grade	Package	Area	Speed (MHz)
WORD_SIZE = 48	Altera Stratix GX (EP1SGX25FF1020)	5	1020 pin BGA	321 LCs 282 LC Registers	422.12
WORD_SIZE = 8	Altera Stratix GX (EP1SGX25FF1020)	5	1020 pin BGA	87 LCs 82 LC Registers	422.12
WORD_SIZE = 64	Altera Stratix GX (EP1SGX25FF1020)	5	1020 pin BGA	428 LCs 362 LC Registers	422.12

Pattern Generator Parameters:

Generic	Default Value	Description
WORD_SIZE	48	Integer that determines how many bits are in the output data block. Valid range for this parameter is between 8 and 64.

Pattern Generator Input/Output Descriptions:

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
word_clk_i	std_logic	NA	NA	I	Clock used by the block. A new output data word is output on every rising edge of this clock.
reset_n_i	std_logic	NA	NA	I	Reset signal for the core (0=reset state, 1=out of reset). This input is asynchronous.
invert_data_i	std_logic	NA	NA	I	Selects whether or not to invert the pattern data to be output (1=inverted, 0=not inverted).
enable_i	std_logic				This input enables the PRBS generators and the output register (1=enabled, 0=disabled)
pattern_select_i	std_logic_vector	1	0	I	Selects the pattern to be captured into the output data register. 00=Data word 01=PRBS $2^{11} - 1$ 10=PRBS $2^{31} - 1$ 11=Data word This input is sampled every clock cycle and will select the corresponding pattern output.
prbs_2_11_seed_i	std_logic_vector	10	0	I	The value to seed the 11 LFSR stages with of the PRBS $2^{11} - 1$ pattern generator. By default out of reset, the generator is seeding with: 000 0000 0001
load_2_11_seed_i	std_logic	NA	NA	I	When this signal is active high, the PRBS $2^{11} - 1$ pattern generator LFSR is loaded with the bits on the prbs_2_11_seed_i signal. If the pattern was being generated when this happens, it will start over at the new seed.
prbs_2_31_seed_i	std_logic_vector	30	0	I	The value to seed the 31 LFSR stages with for the PRBS $2^{31} - 1$ pattern generator. By default out of reset, the generator is seeding with: 0x00000001
load_2_31_seed_i	std_logic	NA	NA	I	When this signal is active high, the PRBS $2^{31} - 1$ pattern generator LFSR is loaded with the bits on the prbs_2_31_seed_i signal. If the pattern was being generated when this happens, it will start over at the new seed.

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
data_word_i	std_logic_vector	X	0	I	The value for the constant data output word pattern.
load_word_i	std_logic	NA	NA	I	When this signal is active high, the value on data_word_i is latched into the data word register on the rising edge of word_clk_i.
bert_data_o	std_logic_vector	X	0	O	The output data word from the pattern generator. The data is in order from most significant bit (MSb) to least significant bit (LSb). In a serial application the MSb should be transmitted first.

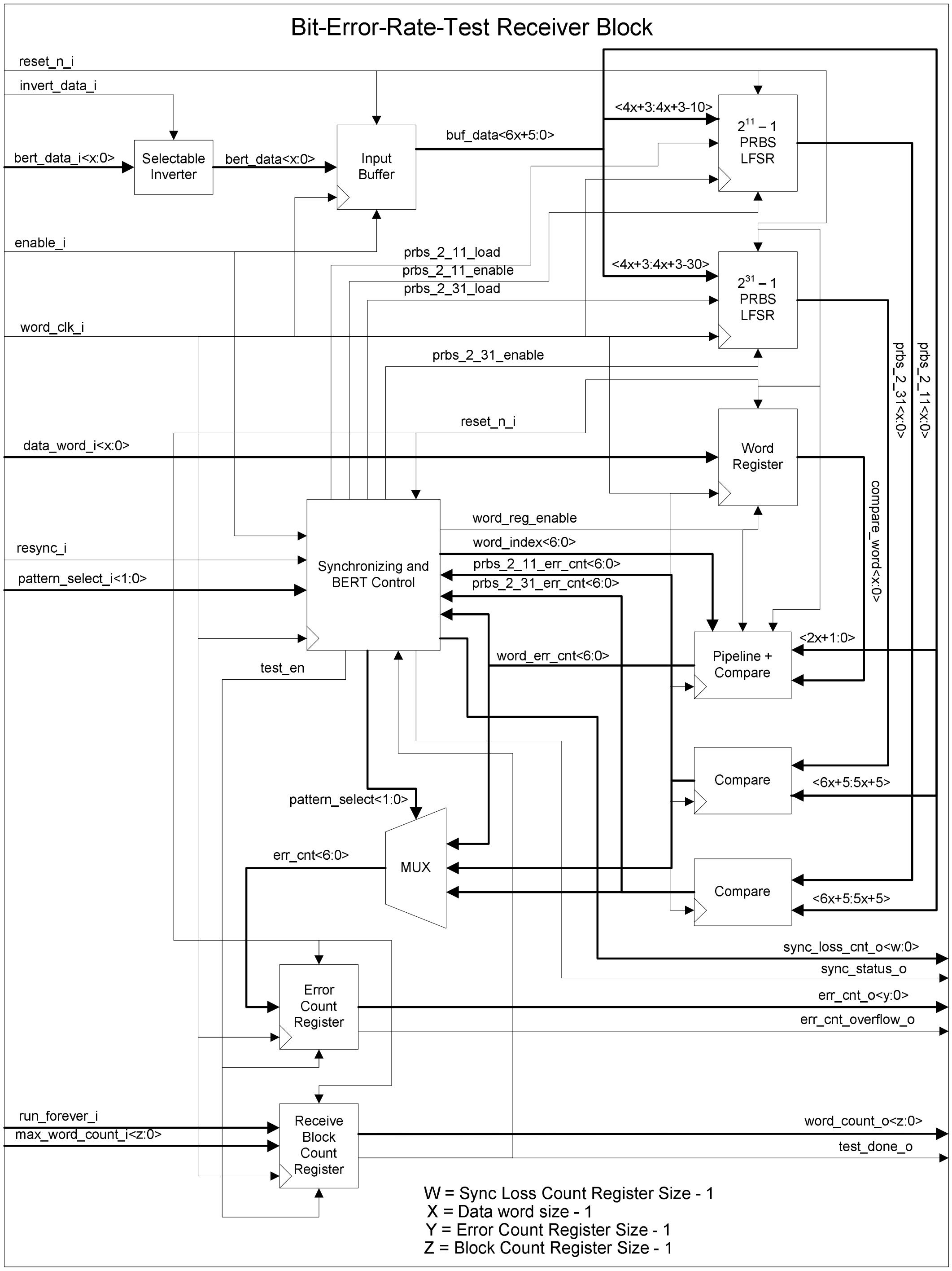
X = WORD_SIZE - 1

Pattern Generator Input Truth Table:

n_reset_i	enable_i	pattern_select_i <1:0>	Core State
0	0	XX	The pattern generator is in reset, and bert_data_o is all zeros. All internal registers are also set to all zeros.
1	0	XX	Output disabled – Core disabled meaning no data is latched in to the output stage register and the LRSRs are not shifted.
1	1	01	The PRBS $2^{11} - 1$ pattern is selected and is being generated. On each subsequent rising edge of word_clk_i the next pattern word will be output.
1	1	10	The PRBS $2^{31} - 1$ pattern is selected and is being generated. On each subsequent rising edge of word_clk_i the next pattern word will be output.
1	1	00 or 11	The data word pattern is selected and will be output on each rising edge of word_ckl_i.

Bit-Error-Rate-Test Receiver Description

Bit-Error-Rate-Test Receiver Functional Diagram:



W = Sync Loss Count Register Size - 1
X = Data word size - 1
Y = Error Count Register Size - 1
Z = Block Count Register Size - 1

Bit-Error-Rate-Test Receiver Timing:

Block Parameters	Chip	Speed Grade	Package	Area	Speed (MHz)
WORD_SIZE = 48 OTHERS = default	Altera Stratix GX (EP1SGX25FF1020)	5	1020 pin BGA	1679 LCs 727 LC Registers	103.73
WORD_SIZE = 8 OTHERS = default	Altera Stratix GX (EP1SGX25FF1020)	5	1020 pin BGA	908 LCs 279 LC Registers	126.25
WORD_SIZE = 64 OTHERS = default	Altera Stratix GX (EP1SGX25FF1020)	5	1020 pin BGA	2091 LCs 907 LC Registers	109.39
WORD_SIZE=16, USE_PRBS_2_31 = FALSE, USE_PROG_WORD = FALSE, OTHERS = default	Altera Stratix GX (EP1SGX25FF1020)	7	1020 pin BGA	526 LCs 272 LC Registers	107.48

Bit-Error-Rate-Test Receiver Parameters:

Generic	Default Value	Description
WORD_SIZE	48	Integer that determines how many bits are in the input data word. Valid range for this parameter is 8 to 64 bits.
ERR_CNT_SIZE	8	Integer that determines how many bits are in the error count counter. Valid range for this parameter is 8 to 31 bits.
SYNC_LOSS_CNT_SIZE	8	Integer that determines how many bits are in the synchronization loss counter. Only synthesis restriction on this parameter.
WORD_CNT_SIZE	64	Integer that determines how many bits are in the received word counter. Only synthesis restriction on this parameter.
SYNC_ERR_CNT	12	Integer number for the maximum number of errors that can occur over a specified number of words (SYNC_WORDS) for synchronization to occur. If more errors than this occur, the block will continue trying to synchronize. Only synthesis restriction on this parameter.
UNSYNC_ERR_CNT	1228	Integer number for the maximum number of errors that can occur over a specified number of words (SYNC_WORDS) for synchronization to be maintained. If more errors than this occur, the block will lose synchronization and will then try to resynchronize. Only synthesis restriction on this parameter.
SYNC_WORDS	256	Integer number for the number of words to count errors over to determine synchronization state. Only synthesis restriction on this parameter.

Generic	Default Value	Description
USE_PRBS_2_11	TRUE	Boolean value that specifies whether logic for the PRBS $2^{11} – 1$ pattern will be instantiated with a conditional compile. The purpose of this is to allow the block to be smaller for cases where only certain patterns are needed. It is important that if this is set to FALSE that the PRBS $2^{11} – 1$ pattern not be selected, because if it is the block will not function correctly. The synthesis tool will not flag an error.
USE_PRBS_2_31	TRUE	Boolean value that specifies whether logic for the PRBS $2^{31} – 1$ pattern will be instantiated with a conditional compile. The purpose of this is to allow the block to be smaller for cases where only certain patterns are needed. It is important that if this is set to FALSE that the PRBS $2^{31} – 1$ pattern not be selected, because if it is the block will not function correctly. The synthesis tool will not flag an error.
USE_PROG_WORD	TRUE	Boolean value that specifies whether logic for the programmable word pattern will be instantiated with a conditional compile. The purpose of this is to allow the block to be smaller for cases where only certain patterns are needed. It is important that if this is set to FALSE that the programmable word pattern not be selected, because if it is the block will not function correctly. The synthesis tool will not flag an error.

Bit-Error-Rate-Test Receiver Input/Output Descriptions:

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
word_clk_i	std_logic	NA	NA	I	Clock used by the block. A new input data word is captured by the block on every rising edge of this clock.
reset_n_i	std_logic	NA	NA	I	Reset signal for the core (0=reset state, 1=out of reset). This input is asynchronous.
enable_i	std_logic	NA	NA	I	When asserted high, causes the input data word to be captured on the rising edge of word_clk_i and allows all other logic in the block to function (1=enabled, 0=disabled).
invert_data_i	std_logic	NA	NA	I	Selects whether or not to invert the input pattern data before processing (1=inverted, 0=not inverted).

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
pattern_select_i	std_logic_vector	1	0	I	Selects the pattern to be synchronized to and bit errors counted for after synchronization. 00=Data word 01=PRBS $2^{11} - 1$ 10=PRBS $2^{31} - 1$ 11=Data word
resync_i	std_logic	NA	NA	I	When asserted active high causes the block to resynchronize to the data. This will not reset the word count and error count registers though and once synchronization occurs, they will continue counting from their current state.
data_word_i	std_logic_vector	X	0	I	The value for the constant data input word pattern.
bert_data_i	std_logic_vector	X	0	I	The word input data to be error checked. This data must be valid on the rising edge of word_clk_i. The data is assumed to be in most significant bit (MSb) to least significant bit (LSb) transmission order to match the BERT pattern generator.
run_forever_i	std_logic			I	Specifies whether the test will be conducted for a finite word count or will be conducted open-ended forever. 0 = run to max_word_count_i 1 = run open-ended
max_word_count_i	std_logic_vector	Z	0	I	Specifies the word count value that when reached will stop the test if run_forever_i is set to 0.
sync_loss_cnt_o	std_logic_vector	W	0	O	Output to indicate how many times the block has lost synchronization during a test. This output is updated on the rising edge of word_clk_i.
err_cnt_o	std_logic_vector	Y	0	O	Counter output that indicates how many bit errors have been counted during the test. This output is updated on the rising edge of word_clk_i. If the error counter overflows, this value will wrap around, but the err_cnt_overflow_o signal will be asserted.
err_cnt_overflow_o	std_logic	NA	NA	O	If the error counter output overflows, this signal will be set to a 1 on the word_clk_i cycle that the overflow occurred. This output is updated on the rising edge of word_clk_i.

Signal Name	Signal Type	Vector Range		IO	Description
		Vector high (MSB)	Vector low (LSB)		
word_cnt_o	std_logic_vector	Z	0	O	Counter output that specifies how many words have been tested for bit errors. The word counter will start at 0 after synchronization occurs and will increment by 1 on each word_clk_i rising edge that the block is enabled until the counter reaches max_word_count_i.
test_done_o	std_logic	NA	NA	O	Output that indicates that the word_cnt_o counter has reached max_word_count_i (1=counter reached, 0=counter not yet reached). This signal is updated on the rising edge of word_clk_i. After this signal is asserted, the test will stop, and the block must be reset to conduct another test.

W = SYNC_LOSS_CNT_SIZE - 1
X = WORD_SIZE – 1
Y = ERR_CNT_SIZE – 1
Z = WORD_CNT_SIZE – 1

Bit-Error-Rate-Test Receiver Input Truth Table:

n_reset _in	enable_ in	pattern_ select_i< 1:0>	Core State
0	X	XX	The BERT receiver is in reset. err_cnt_o is set to 0, sync_loss_cnt_o is set to 0, word_cnt_o is set to 0, and all output status signals are set to the inactive state. All internal registers are also set to their default states.
1	0	XX	Disabled - Core disabled meaning no data is latched in from the bert_data_i signal, the internal pattern generators are not updated, and no error checking or word counting occurs. When the core is once again enabled, it will continue operating where it left off. If it was synchronized before being disabled it may need to resynchronize.
1	1	00 or 11	The data word pattern is selected for synchronization and error detection.
1	1	01	PRBS 2^{11} – 1 pattern selected for synchronization and error detection.
1	1	10	PRBS 2^{31} – 1 pattern selected for synchronization and error detection.

Background

For detailed background information on performing BERT in serial digital multi gigabit communication systems, refer to section 4.

Functional Description

The functional description of the pattern generator and the receiver of the bit-error-rate-test block is described in this section. A description of the patterns that are generated and tested in this BERT block are given in the following table.

Pattern	**Length (bits)**	**Length in Time for 3.125 Gbps**	**Generator Polynomial for LFSR**	**Max 0s Run (bits)**	**Max 1s Run (bits)**
Programmable Word	8 – 64 bits	2.56 ns – 20.48 ns	NA	64	64
$2^{11} - 1$ PRBS	2047	655.04 ns	$X^{11} + X^{9} + 1$	10	11
$2^{31} - 1$ PRBS	2,147,483,647	687.2 ms	$X^{31} + X^{28} + 1$	30	31

Pattern Generator Data Flow

The pattern generator block contains three separate pattern generators which are the $2^{11} - 1$ PRBS generator, the $2^{31} - 1$ PRBS generator and the data word generator. The two PRBS generators are very similar and are implemented using LFSRs as shown in the following figures.

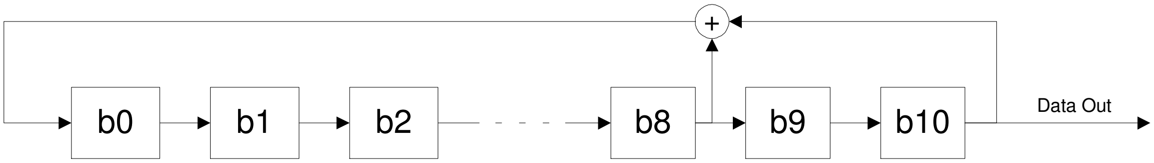


Figure - LFSR Implementation for 2^{11} - 1 PRBS Generator

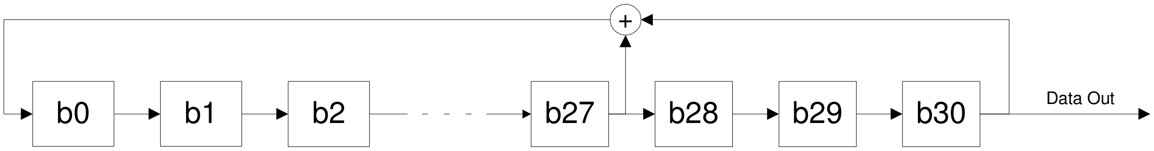


Figure - LFSR Implementation for 2^{31} - 1 PRBS Generator

The LFSRs are implemented using combinational logic to shift them WORD_SIZE times per word_clk_i clock cycle. These LFSRs are based on the polynomial for each pattern shown in the following table. The LFSRs produce repeating bit patterns and they can be started at any point in the repeating pattern by loading a seed value into the LFSR register bits. This is done using the seed inputs to the block and asserting the corresponding seed load signals.

The programmable data word is simply generated by a register in the block that captures an input data word when the load_word_i signal is asserted. It is possible to have the pattern generator generate any desired pattern by providing a new word on each word_clk_i cycle.

The next stage of the pattern generator is the selection of the pattern to be transmitted through the mux using the pattern_select_i signal.

After the mux is a selectable inverter that allows the pattern to be inverted or not using the invert_data_i signal.

The final stage is the selected pattern is captured into an output register.

If only a single PRBS pattern is needed, some logic overhead can be eliminated by simply instantiating the PRBS generator block (prbs_generator.vhd). The inputs and outputs for this block are described in the comments in the source code.

Receiver Data Flow

The first stage of the receiver data flow is the selectable inverter which inverts the data coming in if the invert_data_i signal is asserted.

The next stage of the receiver data flow is the input buffer. The buffer is 6 words long to allow for enough bits to be captured to seed the $2^{31} – 1$ PRBS generator within the receiver that is used for generating the expected data. The first 4 words contain at least 32 bits since the minimum data word size is 8 bits. The next 2 words in the buffer are used to save the data for long enough to make the comparison with the output from the PRBS generator. A block diagram of the buffer is shown in the following figure.

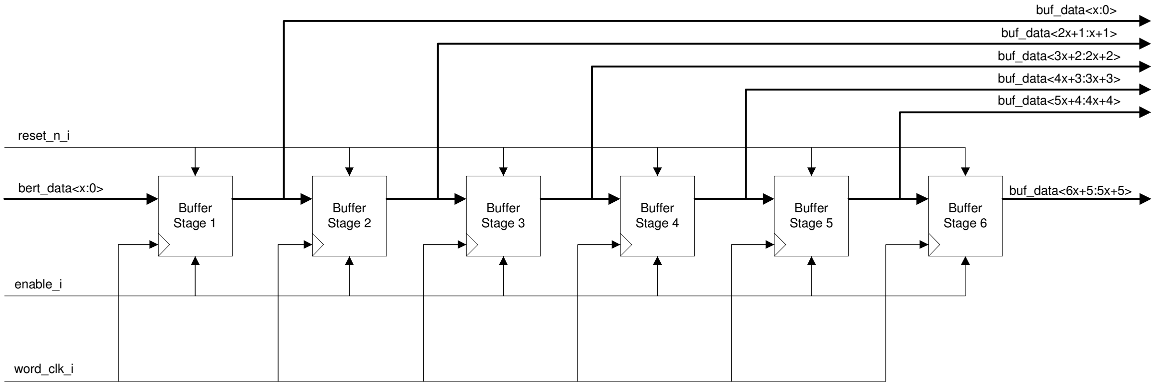


Figure - Pattern Receiver Input Buffer Block Diagram

The last stage of the receiver data flow after synchronization has taken place is the comparison of the received data with the expected data and the counting of the bit errors. The number of bit errors for the selected pattern is then added to the current value in the error counter and the result is captured in the output register. Also after each comparison, the word count register is incremented by one. The comparison is made directly with the data word in the third stage of the buffer for the PRBS patterns. In the synchronization of the programmable word pattern, the starting index in the buffer is determined for where the word is lined up in the buffer. The comparison is made bit by bit starting at the starting index and continuing for all of the bits in the word.

The synchronization and the error counting stages are controlled by the Synchronization and BERT control block. This block contains a state machine that controls the appropriate enables and outputs. A high level flow for how the synchronization block works after coming out of reset is given in the following table.

Step	Description for PRBS Patterns	Description for Programmable Word Pattern
1	Wait two clock cycles for the data buffer to be filled up	
2	If the selected pattern is a PRBS pattern, then assert the appropriate load signal and enable signal for the selected PRBS pattern generator. The received data is loaded into the appropriate PRBS generator as a seed value.	If the selected pattern is the programmable word pattern, then increment the word buffer index by one starting from 0 and continuing until X (wrap back to 0)
3	Add the error count value for the selected pattern to an accumulator starting at 0.	
4	Goto step 3 (repeat SYNC_WORDS number of times)	
5	If the error count accumulator is less than SYNC_ERR_CNT then go to step 6 else go back to step 2.	
6	Assert the synchronization status output, turn on the enables for the error counter and the word counter, and set the select lines for the error count mux to select the correct error count value.	
7	Add the error count value for the selected pattern to an accumulator starting at 0.	
8	Goto step 7 (repeat SYNC_WORDS number of times)	
9	If the error count accumulator is greater than UNSYNC_ERR_CNT then deassert the synchronization status output, disable the error counter and word counter and go to step 2. Otherwise go back to step 7.	

The synchronization algorithm will search for synchronization at startup by ensuring that the bit error rate is below a certain threshold defined by the following equation.

$$BER = \frac{SYNC\_ERR\_CNT}{BLOCK\_SIZE \times SYNC\_BLOCKS}$$

It will then continuously monitor the synchronization status and if the bit error rate increases above the threshold defined by the following equation, assumes that synchronization has been lost and then again tries to resynchronize.

$$BER = \frac{UNSYNC\_ERR\_CNT}{BLOCK\_SIZE \times SYNC\_BLOCKS}$$

If the value of SYNC_ERR_CNT is set lower than UNSYNC_ERR_CNT there is hysteresis built into the synchronization so the block will not jump in and out of synchronization when the channel gets noisy.

If the test continues until the word_cnt_o value reaches the max_word_cnt_i value, then the test will stop and the test_done_o bit will be asserted. The BERT receiver block will not do anything more until it is reset.

Functional Timing

All input signals in the design are captured on the rising edge of word_clk_i and all output signals in the design clocked out on the rising edge of word_clk_i. All actions in response to input signals will occur on the first rising edge that the signal is valid for except the reset_n_i signal which is asynchronous.

There are no real restrictions on the relative timings of input control signals to the block. A reasonable order of operations for a design including both the pattern generator and the bert receiver is given in the following table.

Signals	**Clock Cycle #**								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Common Signals									
reset_n_i	0	1	1	1	1	1	1	1	1
invert_data_i	X	X	I	I	I	I	I	I	I
Pattern Generator Signals									
pattern_select_i	11	11	11	PQ	PQ	PQ	PQ	PQ	PQ
prbs_2_11_seed_i	X	X	V	X	X	X	X	X	X
load_2_11_seed_i	0	0	1	0	0	0	0	0	0
prbs_2_31_seed_i	X	X	V	X	X	X	X	X	X
load_2_31_seed_i	0	0	1	0	0	0	0	0	0
data_word_i	X	X	V	X	X	X	X	X	X
load_word_i	0	0	1	0	0	0	0	0	0
bert_data_o	0s	0s	0s	0s	V	V	V	V	V
Receiver Signals									
enable_i	0	0	0	1	1	1	1	1	1
pattern_select_i	11	11	11	PQ	PQ	PQ	PQ	PQ	PQ
max_word_count_i	X	X	X	V	V	V	V	V	V
data_word_i	X	X	V	V	V	V	V	V	V
X = Don’t care V = Valid data or signal I = 0 or 1 PQ = 00 for Programmable word, 01 for PRBS $2^{11} – 1$, or 10 for $2^{31} – 1$									

The pattern generator can function at very high rates for any valid data word size between 8 and 64 bits. For the Altera Stratix GX device, the pattern generator can function at the highest allowable internal clock frequency. The bert receiver is much more limited in performance however. As the word size increases the maximum operating frequency decreases, but not as fast as the effective data rate increases. For multi-gigabit data rates, a word size of 32 bits or higher may be required.

The worst case timing path in the pattern receiver is for the data word pattern. To improve the timing of the design, an architecture change would need to be made to the synchronization design for the word pattern. Instead of searching through two input data word buffer stages for the start of the programmable word the receiver could be designed to just look in the first input data word buffer stage and have some external logic synchronize the incoming data so that it lines up in that stage. The receiver could output a “bit_slip” signal that could trigger the external logic to slip the data by 1 bit using a barrel shifter. When using this receiver in a multi-gigabit serial application, the Xilinx and Atlera transceivers support this bit slip functionality.

Test Benches

The test bench for the bert pattern generator and bert receiver is contained in the file bert_tb.vhd. This test bench instantiates the bert pattern generator block and the bert receiver block and generates the word_clk_i signal and all of the necessary control signals. The test bench then monitors the output signals and provides self checking with assertions. Any simulation errors will appear in the simulation transcript window. There are two main parts to the bert_tb.vhd file, a package section to define constants, and the main test bench entity and architecture. There are three separate packages defined, one for each of the three data patterns. The packages are defined first in the file and then the package for the desired test is selected with a "use" statement. This must be manually edited to switch between tests. To conduct a test, initiate the corresponding .do script from within Modelsim for each type of pattern (bert_tb_word.do, bert_tb_prbs_2_11.do, or bert_tb_prbs_2_31.do). The script will set up the simulation, set up the waveform window, and conduct the simulation for an appropriate amount of time for each type of test.

Package Constants

The following table lists all of the constants defined in the packages and what they are for. Also listed are values for the constants that can be used to stress the design.

Constant	**Description**	**Default Value**	**Stresses**
WORD_CLK_PERIOD	The clock period for the word clock (not important for functional sims)	20 ns	
WORD_SIZE	These constants map directly to generics in the pattern generator and pattern receiver	48	
ERR_CNT_SIZE		10	Use a smaller value such as 8 to cause a small enough register size to cause overflows in the errors counted
SYNC_LOSS_CNT_SIZE		8	
WORD_CNT_SIZE		64	
SYNC_ERR_CNT		Word = 2, PRBS = 12	
UNSYNC_ERR_CNT		Word = 20, PRBS = 1228	
SYNC_WORDS		Word = 10, PRBS = 256	
INVERT_DATA	Specifies the state of the invert_data_i signal to each block	0	
DATA_WORD	The constant data word to use in the word pattern test	48 bits = AA55FF0055AA 8 bits = A5 64 BITS = 00A0000F00005040	Try patterns that have symmetry to cause the receiver to synchronize at the wrong place.

Constant	Description	Default Value	Stresses
PRBS_2_11_SEED	The value to be connected to prbs_2_11_seed_i of pattern generator	00000000001	
PRBS_2_31_SEED	The value to be connected to prbs_2_31_seed_i of pattern generator	0000000000000000000000000000001	
TEST_DURATION	The value to be connected to the max_word_cnt_i signal on the bert receiver	Word pattern = FFF PRBS patterns = 30000	
ERRORS_TO_INSERT	Number of bit errors to insert into the data. The errors are inserted one per block until this number of errors is reached.	400	Choose value that results in sync loss and verify that sync loss occurs. This may need to be changed in conjunction with the sync parameters.
ERROR_BURST	If set to a 1, the test will insert a single error burst to cause a loss of sync and automatically check for that loss of sync	1	Set to 0 to test the error counting functionality in the bert receiver.
TIME_BETWEEN_ERRORS	If errors are to be inserted into the data, they are inserted 1 per word and spaced apart by this many words.	250	
DELAY_TO_BURST_ERROR	The amount of clock cycles to delay after synchronization occurs before the error burst is sent.	800	Adjusting this value can cause the error burst to line up with the sync check in the bert receiver such that not enough errors are present for a loss of sync.
TEST_PATTERN	Selects which pattern to test	00 for Word, 01 for PRBS $2^{11} - 1$, or 10 for $2^{31} - 1$	
TRANSMIT_ENABLE_DELAY	Number of clock cycles to delay from reset to enabling the transmitter	10	
RECEIVE_ENABLE_DELAY	Number of clock cycles to delay from reset to enabling the receiver	20	
DISABLE_RECEIVER_DELAY	The number of clock cycles to disable the receiver for during the test. If set to 0, the receiver will not be disabled.	0	Set to a positive number and verify that the receiver behaves as expected while being disabled.
LOAD_WORD_DELAY	The number of clock cycles to wait before loading the programmable word in the	30	

Constant	Description	Default Value	Stresses
	pattern generator		
TRANSMIT_DELAY	There is a buffer delay on the data transmitted from the bert pattern generator, to the data received at the bert receiver. This number specifies the number of clock cycles for that delay.	10 (Valid range is 0 to WORD_SIZE – 1)	Change this value to watch the programmable word pattern be synchronized to at different indexes.
TEST_TIMEOUT	Maximum number of clock cycles to wait after the test bench counter reaches TEST_DURATION while checking for a test_done_o assertion by the bert receiver. If this counter expires without seeing a test_done_o assertion, an error is reported.	10000	
SYNC_TIMEOUT	The amount of time to wait before checking to see if initial synchronization occurred	Defined by other constants, do not change.	

Self Checking

The test bench contains self checking functionality. The following is a list of things that are checked.

- Initial synchronization occurs within a reasonable amount of time
- The number of errors counted matches the number inserted (when no burst error insertion is present)
- The number of errors counted is larger than the number inserted (when burst error insertion is present)
- That the test stops at the maximum word count
- That the synchronization is lost after initial synchronization when a burst error insertion is present
- That the error count overflow output bit is asserted when the error counter overflows and that it is deasserted when there is no overflow

The following table contains all of the error messages and their possible causes.

ERROR Message	Reason	Manual Checks
ERROR - initial synchronization did not occur as expected	The initial synchronization did not occur within the specified time frame.	Verify whether synchronization ever occurred and if not determine why. If it did determine if the expected time frame for it to occur was accurate
ERROR - errors counted did not match errors inserted	When a burst error was not present, the number of errors counted did not match the number inserted.	If the number of errors inserted was enough to cause a loss of synchronization, this error may occur in which case a manual check would be required. Another possible cause is an error count register size that is too small to count the number of errors inserted causing the counter to overflow.
ERROR - too few errors counted with burst error present	When a burst error was present, the number of errors counted was not larger than the number of single errors inserted.	Check to make sure the error counter did not overflow.
ERROR - test did not stop at max word count	The test did not complete in the expected amount of time	Check where the word_cnt signal is it and if it is incrementing. If so, then it may just be that with the setup parameters, the test is taking longer than expected. Try increasing the value of TEST_TIMEOUT.
ERROR - Expected loss of sync, but it did not occur	The error burst did not cause a loss of sync.	It is possible that the error burst lined up with the synchronization checking state machine such that not enough errors were detected to cause a loss of synchronization. Try changing DELAY_TO_BURST_ERROR constant.
ERROR - Did not get expected err_cnt_overflow indication	The error counter overflowed but the overflow output bit was not asserted	There's no valid reason to see this error so a design error is implied
ERROR - Got unexpected err_cnt_overflow indication	The overflow output bit was asserted but the error counter did not overflow	There no valid reason to see this error so a design error is implied

Engineering Project Report Approval Form

MASTER OF SCIENCE - ENGINEERING Milwaukee School of Engineering

This report, for the project titled ___CODING AND BIT-ERROR-RATE-TEST BLOCKS FOR A SERIAL DIGITAL MULTI-GIGABIT COMMUNICATION SYSTEM____,

submitted by the student ___David T. Carney___________,

has been approved by the following committee:

Faculty Advisor: ________________________________ Date: ________________

Faculty Member: ________________________________ Date: ________________

Faculty Member: ________________________________ Date: ________________