XAPP661 (v2.0.2) May 24, 2004

RocketIO Transceiver Bit-Error Rate Tester

Author: Dai Huang and Michael Matera

Summary

This application note describes the implementation of a RocketIO™ transceiver bit-error rate tester (BERT) reference design demonstrating a serial link (1.0 Gb/s to 3.125 Gb/s) between two RocketIO multi-gigabit transceivers (MGT) embedded in a single Virtex-II Pro™ FPGA. To build a system, an IBM CoreConnect™ infrastructure connects the PowerPC™ 405 processor (PPC405) to external memory and other peripherals using the processor local bus (PLB). Use the reference design for this application note, XAPP661.zip, and the Embedded Development Kit[1] (EDK) to build an easily modified or extended PPC405 system. A two-channel Xilinx bit-error rate tester (XBERT) module is used for generating and verifying high-speed serial data transmitted and received by the RocketIO transceivers. The data to be transmitted is constructed using pseudo-random bit sequence (PRBS) patterns. The receiver in XBERT module compares the incoming data with the expected data to analyze for errors. The XBERT supports several different types of user selectable PRBS and clock patterns. Frame counters in the receiver are used to track the total number of data words (frames) received, the total number of data words with bit errors, and the total number of bit errors. The processor reads the status and counter values from the XBERT through the PLB Interface, then sends out the information to the UART. The reference design also supports in-circuit partial reconfiguration of RocketIO transceiver attributes using the Virtex-II Pro internal configuration access port (ICAP). Use this solution to perform partial reconfiguration of the RocketIO transceiver's pre-emphasis and differential swing control attributes. For more information regarding ICAP and in-circuit partial reconfiguration of RocketIO attributes, refer to XAPP662: In-Circuit Partial Reconfiguration of RocketIO Attributes[2].

Hardware Implementation

Overview

Figure 1 provides a high-level view of the hardware contents of the RocketIO transceiver BERT reference design. This design demonstrates a PPC405 system and PLB devices. The PLB consists of four slave devices. A block RAM (BRAM) controller connects 64 KBytes BRAM to the bus serving as the data and instruction memory of the processor. Also attached to the bus are a UART module, a two-channel XBERT module, and an ICAP module each bonded with a Xilinx Intellectual Property Interface (IPIF) providing standardized connections to the bus. The processor has two PLB master connections, one for instruction cache and one for data cache.

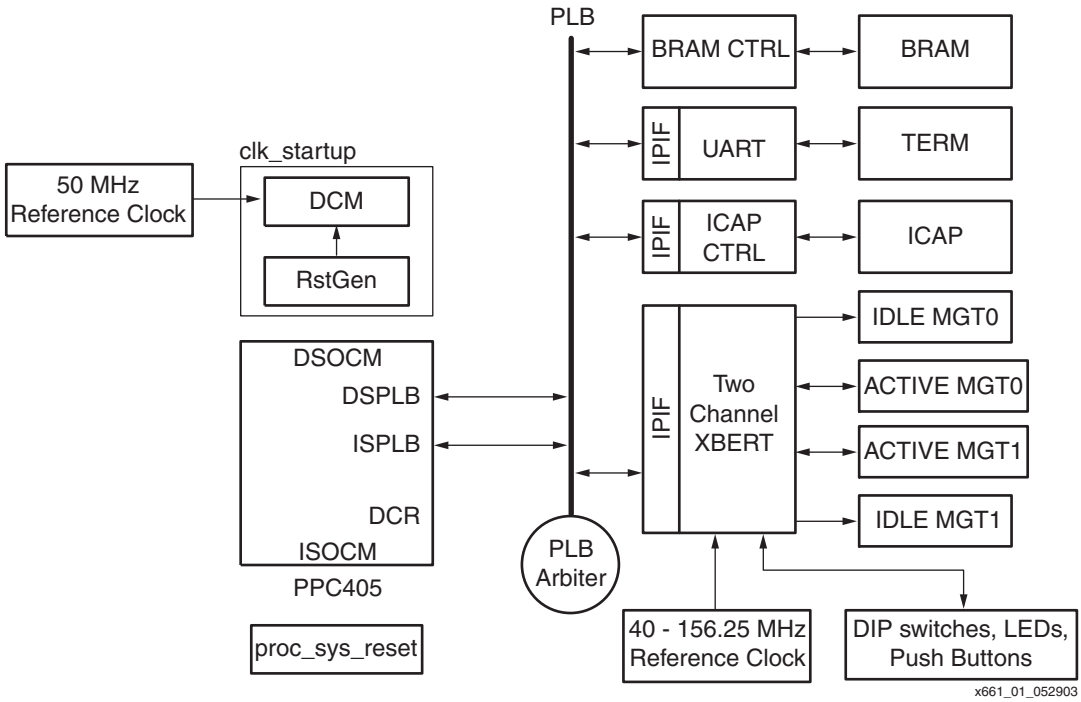


Figure 1: **High-Level Hardware View: RocketIO Transceiver BERT Reference Design**

Processor Local Bus (PLB)

The PLB connects the CPU to high-performance devices such as memory controllers. The PLB protocol supports higher-bandwidth transactions and has a feature set that better supports memory operations from the CPU. Highlights of the PLB protocol include synchronous architecture, independent read/write data paths, and split transaction address/data buses. The reference design includes a 64-bit PLB infrastructure with 64-bit master and slave devices attached.

The UART, the ICAP, and the two-channel XBERT module do not require the high performance available over PLB, but are connected to PLB to demonstrate a simple system without OPB and keep the size of the design small.

The PLB devices in the reference design include:

- PLB Masters
  - CPU Instruction Side PLB Interface (Master ID = 0)
  - CPU Data Side PLB Interface (Master ID = 1)
- PLB Slaves
  - BRAM Controller (Slave ID = 0)
  - 16550 UART (Slave ID = 1)
  - Two-Channel XBERT Module (Slave ID = 2)
  - ICAP Controller (Slave ID = 3)
- PLB Arbiter
  - 8 Master, 64 bit Xilinx PLB Arbiter

Clock/Reset Distribution

Figure 2 illustrates the use of the digital clock manager (DCM) to generate clocks for the CPU and PLB bus in the clk_startup module. A 50 MHz input reference clock is used to generate the main PLB clock. The CPU clock of 200 MHz is a multiple of the PLB clock. Since each clock is generated from the same 50 MHz reference clock input, they are phase-aligned with each other. This synchronous phase alignment is required by the CPU.

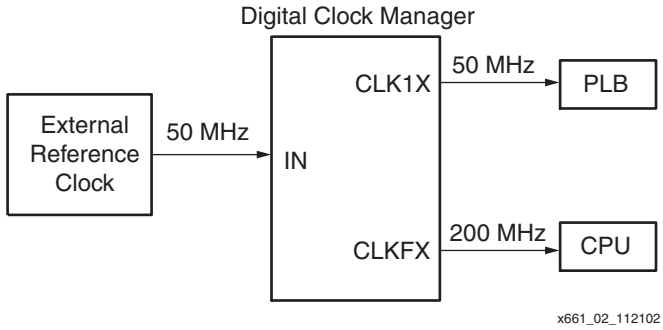


Figure 2: **CPU and PLB Clock Generation**

The startup reset for the PPC405 is generated by the proc_sys_reset module and triggered by an external reset input. The proc_sys_reset module is a standard component provided in the EDK.

The two-channel XBERT module requires a separate high-quality clock source to drive RocketIO transceivers. The data transfer rate of the RocketIO transceiver will be 20 times this clock rate. Therefore, running the RocketIO transceiver at 3.125 Gb/s requires a 156.25 MHz clock source. This clock source must be at least 40 MHz, with a duty cycle between 45% and 55%, and a 100 ppm or better frequency stability, with as low as possible jitter. For lower jitter, it is necessary to use differential clock inputs. At speeds of 2.5 Gb/s or greater, the REFCLK configuration introduces more than the maximum allowable jitter to the RocketIO transceiver. For these high-speed designs, the BREFCLK configuration is required. The BREFCLK configuration uses dedicated routing resources to reduce jitter. BREFCLK must enter the FPGA through a dedicated clock I/O. There are two groups of dedicated differential clock input pads on the top of the device for the top MGTs, named BREFCLK and BREFCLK2 respectively. This is also true on the bottom. Setting the REF_CLK_V_SEL attribute of the RocketIO transceiver to "1" enables a BREFCLK path. The REFCLKSEL pin can be used to select between BREFCLK and BREFCLK2. Consult the RocketIO Transceiver User Guide[3] for more information.

Figure 3 shows the clock generation for the two-channel XBERT module. Only one instantiation of the RocketIO transceiver is shown. The XBERT module uses a 2-byte data path. Therefore, it is capable of using a simple clock scheme with both USRCLK and USRCLK2 running at the same frequency. The buffered BREFCLK or BREFCLK2 is routed directly to the TXUSRCLK and TXUSRCLK2 on the RocketIO transceiver. RXUSRCLK and RXUSRCLK2 takes the recovered clock (RXRECCLK) from the RocketIO transceiver. The board interface module in the two-channel XBERT requires a slow clock (typically 12.5 MHz) to drive the debounce circuitry and the LED pulse-width modulation (PWM) module. This slow clock is derived from the PLB clock and generated from a separate DCM.

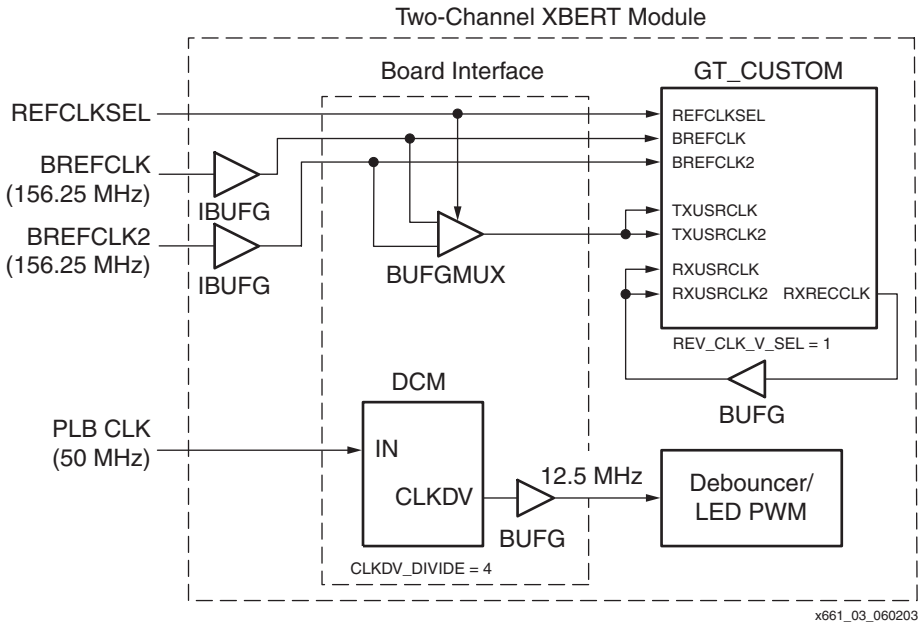


Figure 3: **Clock Generation in Two-Channel XBERT Module**

RocketIO Transceiver

The RocketIO transceiver consists of the physical-media attachment (PMA) and physical-coding sublayer (PCS). The PMA contains the serializer/deserializer (SERDES), TX and RX buffers, clock generator, and clock recovery circuitry. The PCS contains the 8B/10B encoder/decoder and the elastic buffer supporting channel bonding and clock correction. The PCS also handles cyclic redundancy check (CRC). The reference design only implement most necessary features provided by the RocketIO transceiver. Table 1 describes the RocketIO transceiver features deployed in the reference design.

Table 1: **Deployment of the RocketIO Transceiver Features in the Reference Design**

Layer	RocketIO Transceiver Features	Deployment in the Reference Design	Note
PMA	SERDES	Yes	
	SERDES_10B Mode	No	
	Comma Detect and Realign	Yes	PCOMMA only
	Clock Generation and Recovery	Yes	
	External Serial Loopback	Yes	False by default
PCS	Rx Elastic Buffer	Yes	
	Tx FIFO	Yes	
	Internal Parallel Loopback	Yes	False by default
	Clock Correction	No	
	8B/10B Encoder/Decoder	No	
	CRC	No	
	Channel Bonding	No	

The reference design facilitates two programmable loopback features provided by RocketIO transceivers to allow the user quick testing of the system without having the need to apply external cables or pattern generators. Loopback allows the user to send the data that is being transmitted directly to the receiver of the transceiver. External serial loopback places the transceiver into a state where transmit data is directly fed back to the receiver in PMA layer. An important point to note is that the feedback path is at the output pads of the transmitter. This tests the entirety of the transmitter and receiver. Proper termination (typically 50 Ω) on output pads is required to run a RocketIO transceiver in serial loopback mode. The second loopback option is internal parallel loopback. It checks the digital circuitry only by feeding back the data before the SERDES. Termination on output pads is not required in parallel loopback although transmitter outputs still remain active.

The RocketIO transceiver BERT reference design implements 2-byte data path on the RocketIO transceiver parallel interface to the FPGA fabric. With 8B/10B encoding bypassed, the parallel interface on the RocketIO transceiver is 20 bits wide. The reference design supports using different level of TX_DIFF_CTRL and TX_PREEMPHASIS settings on the RocketIO transceiver to improve output waveform shaping for various load conditions.

Xilinx Bit-Error Rate Tester (XBERT)

The Xilinx Bit-Error Rate Tester (XBERT) is a set of HDL modules designed to test the bit-error rate of a RocketIO transceiver in Virtex-II Pro. A simple setup for doing such test is to apply SMA cables between two RocketIO transceivers so that transmitted data on each RocketIO transceiver can be looped back to the receiver of the other RocketIO transceiver over the established link. Note that such cable loopback is different from the two programmable loopback features provided by a RocketIO transceiver because it tests the entire hardware system. The XBERT is a self-contained module that can be implemented in FPGA fabric without CPU interference. The RocketIO transceiver BERT reference design gives an example of XBERT application in a PPC405 based system.

A single channel XBERT contains three primary modules; GigabitBER_TX, GigabitBER_RX, and MGT_BERT_4, as shown in Figure 4. GigabitBER_TX is to generate and transmit the data. GigabitBER_RX is used to check the received data and count errors over a serial link. MGT_BERT_4 provides a placeholder for the RocketIO transceiver instantiation.

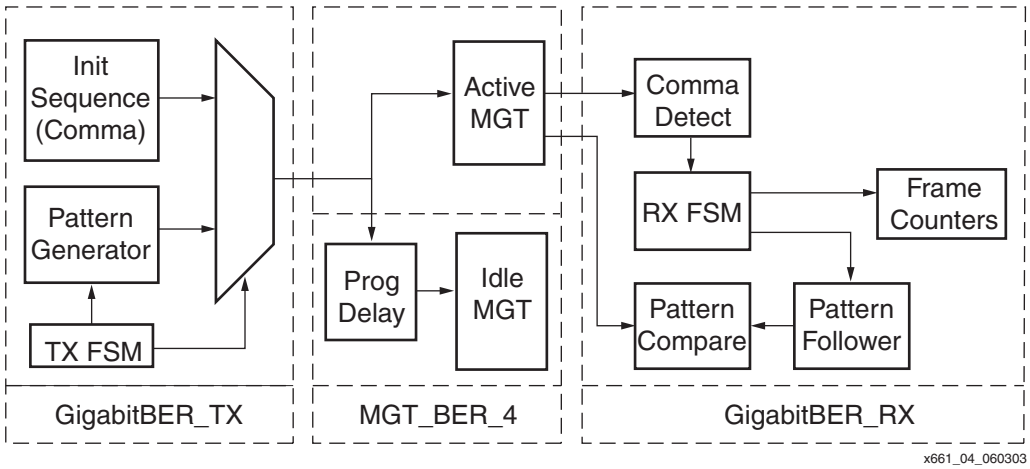


Figure 4: **Single Channel XBERT Block Diagram**

GigabitBER_TX contains a transmission state machine (shown in Figure 5), that controls the pattern generator and the multiplexer between an initialization sequence and outputs from the pattern generator. After detecting a TX reset on the XBERT, an initialization sequence formed by a comma stream is transmitted to the receiver. A comma used in XBERT is a 20-bit word composed of `10'b0011111010` (K28.5) and `10'b0011100001` (D28.7). After

transmitting 524,288 commas, the state machine starts taking inputs from the pattern generator.

GigabitBER_RX contains a receiving state machine split into two parts, comma-detection FSM and pattern-follower FSM (shown in Figure 6). Comma-detection FSM detects the incoming comma stream. After receiving 64 consecutive commas, the comma-detection FSM recognizes a remote reset from the transmitter and enters the COMMA_STATE. A remote reset signal is asserted in COMMA_STATE. Detection on the rising edge of the remote reset signal generates a single pulse of ENPCOMMAALIGN on the RocketIO transceiver. The comma-detection FSM can only step out of this state when the link is up or when it receives 64 non-comma words. Therefore, the state machine tolerates scattered errors in the comma stream and avoids triggering multiple remote resets. Once the link is up, the comma-detection FSM is locked into ARMED_STATE. This prevents recognizing any data pattern as a comma. The pattern-follower FSM controls the pattern follower generating the expected data pattern. During the remote reset, the pattern follower generates and locks its first data. The state machine declares link up when the received data matches the first expected data. Afterwards, the pattern follower advances. However, when the state machine detects two consecutive errors in the received data, it enters the ABORT_STATE and declares a link down. The pattern-follower FSM stays in the ABORT_STATE until it receives a new remote reset from the transmitter.

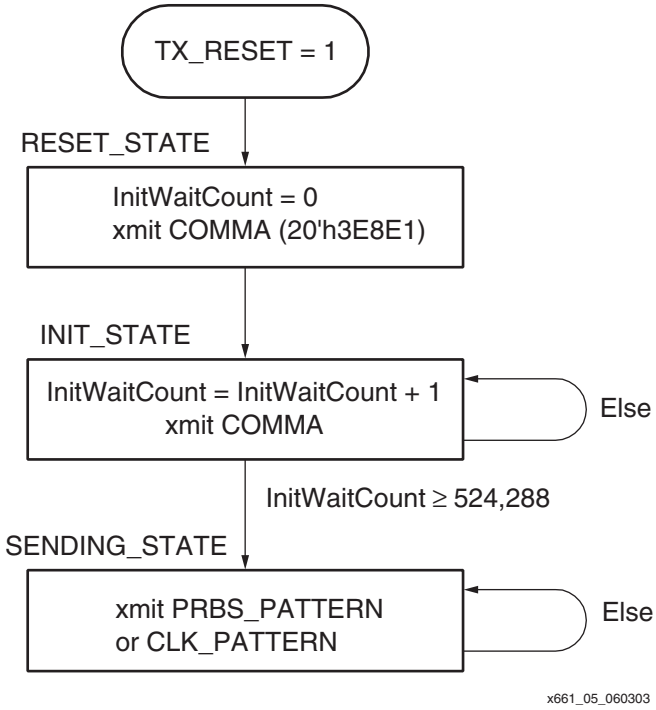


Figure 5: **XBERT Transmission State Machine**

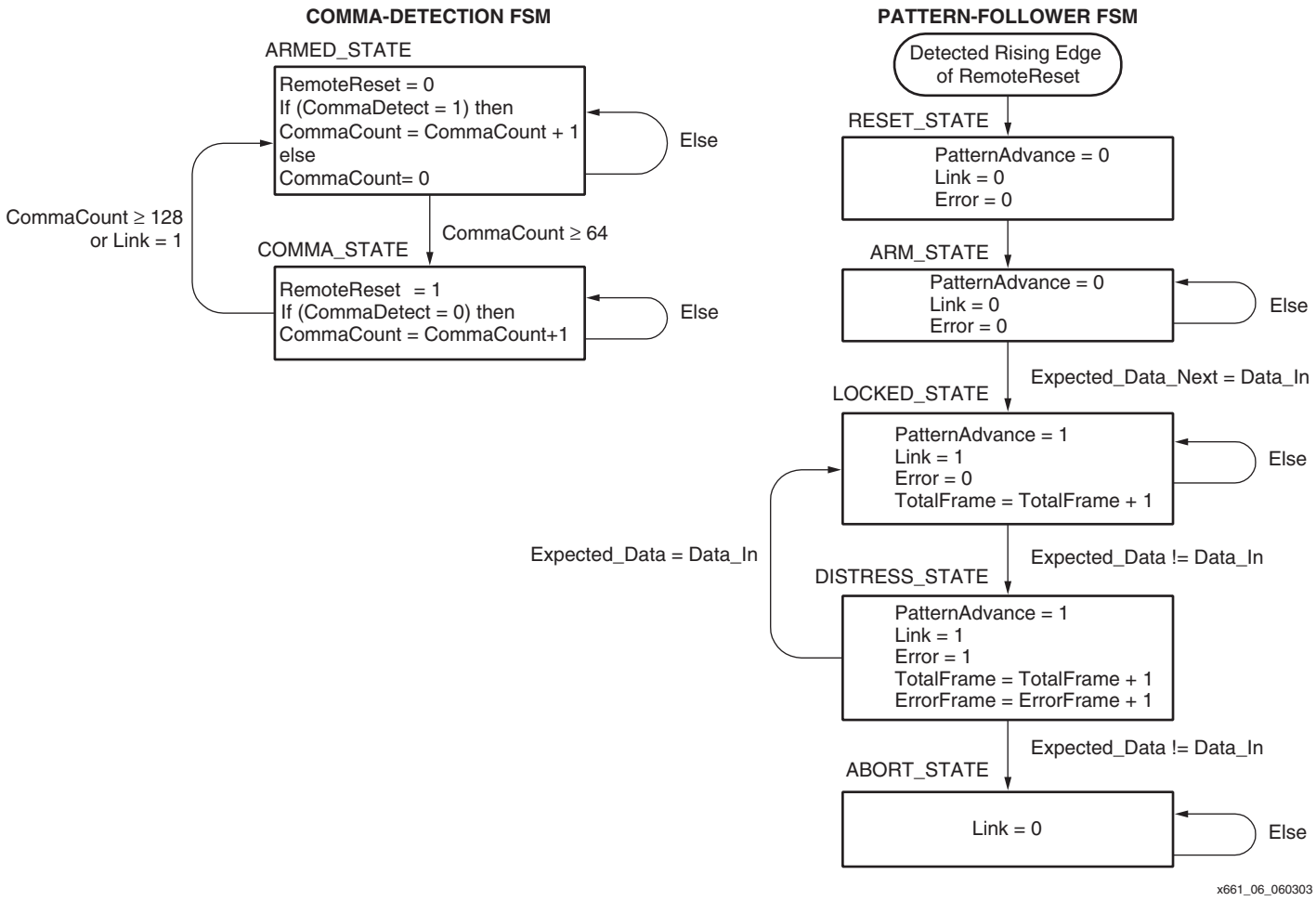


Figure 6: **XBERT Receiving State Machine**

XBERT instantiates two RocketIO transceivers using MGT_BER_4 modules. The active MGT connects to both GigabitBER_TX and GigabitBER_RX. The idle MGT, an option in the design, only connects to GigabitBER_TX through a programmable-delay module. The programmable-delay module can insert 0 to 15 clock cycles of delay on the entire 20-bit frame between the pattern generator and the idle MGT. Therefore, the idle MGT can be used to create adjustable noise interference to the active MGT.

The reference design implements a two-channel XBERT by replicating two XBERTs in the FPGA fabric. The placement of two XBERTs is a choice between a single-bank configuration or split-bank configuration. In single-bank configuration two XBERTs and their transceivers are placed on the same bank of an FPGA and share the same clock inputs. In split-bank configuration two XBERTS are separated into the top and bottom FPGA bank, each taking a dedicated clock input.

A board interface module debounces inputs from the on-board DIP switches and push buttons, and synchronizes them to a slow clock (typically 12.5 MHz) derived from the run-time clock input (i.e., the PLB clock in the reference design). This slow clock also drives a pulse-width modulation (PWM) module generating modulated LED outputs to the LEDs. Figure 7 illustrates a wrapper of all these modules instantiated in a two-channel XBERT module. Table 3 to Table 7 provides port lists of the two-channel XBERT.

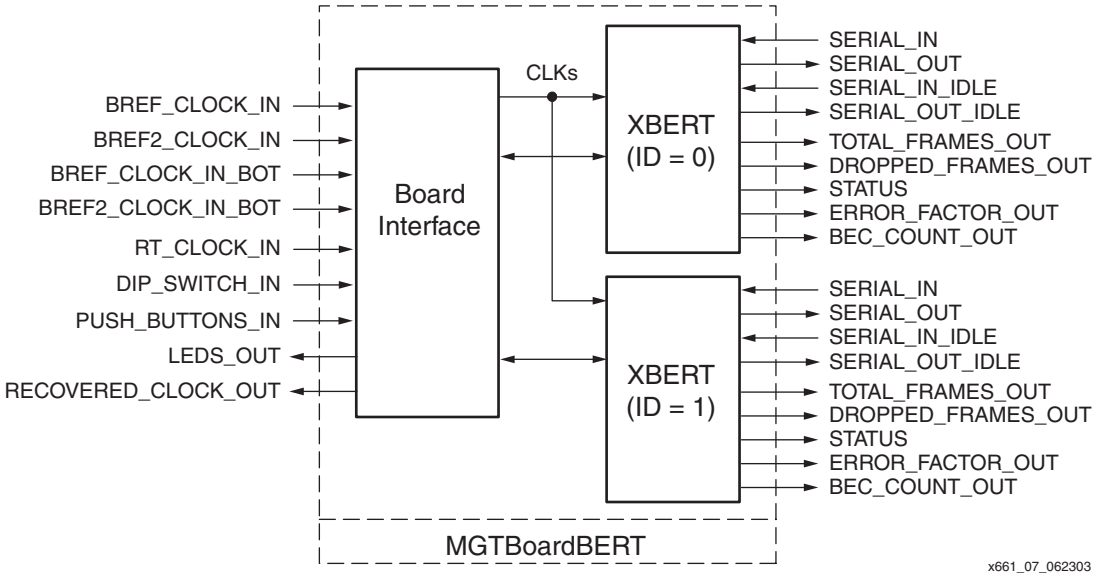


Figure 7: **Two-Channel XBERT Block Diagram**

In order to perform a successful bit-error rate test some key features of the RocketIO transceiver must be properly controlled. Most importantly the serial data stream must exactly match the data stream generated by the pseudo-random bit sequences (PRBS) pattern generator. In most cases this type of pattern violates run length and DC balance rules of the RocketIO transceiver. This violation is intentional in order to produce part stress. In order for the part to be stressed properly, 8B/10B encoding on the RocketIO transceiver must be bypassed. The use of the 8B/10B encoding will not only prevent proper line stress but will disrupt the operation of the XBERT. Other mandatory RocketIO transceiver attribute settings required by the XBERT are listed in Table 7.

Table 2: **List of Two-Channel XBERT Ports**

Port Name	Direction	Width	Clock Domain	Description
bref_clock_p_in bref_clock_n_in	In In	1 1	N/A	High quality RocketIO transceiver differential BREFCLK inputs.
bref2_clock_p_in bref2_clock_n_in	In In	1 1		High quality RocketIO transceiver differential BREFCLK2 inputs as an alternative to BREFCLK.
bref_clock_p_in_bot bref_clock_n_in_bot	In In	1 1		High quality RocketIO transceiver differential BREFCLK inputs used in split-bank configuration.
bref2_clock_p_in_bot bref2_clock_n_in_bot	In In	1 1		High quality RocketIO transceiver differential BREFCLK2 inputs as an alternative to BREFCLK used in split-bank configuration.
rt_clock_in	In	1		Board interface run-time clock input. It is used to generate the slow clock for the debouncer and LED PWM.
DIP_switch_in	In	32	Asynchronous	On-board DIP switch inputs. Refer to Table 3 for more information.
Push_buttons_in	In	4		On-board push button inputs. Refer to Table 4 for more information.

Table 2: **List of Two-Channel XBERT Ports *(Continued)***

Port Name	Direction	Width	Clock Domain	Description
serial_p_in serial_n_in	In In	2 2	Analog	High speed differential inputs of two active RocketIO transceiver ports. Bit 0 is the input to the active transceiver in XBERT 0. Bit 1 is the input to active transceiver in XBERT 1.
serial_p_out serial_n_out	Out Out	2 2		High speed differential outputs of two active RocketIO transceiver ports. Bit 0 is the output from the active transceiver in XBERT 0. Bit 1 is the output from the active transceiver in XBERT 1.
serial_p_in_idle serial_n_in_idle	In In	2 2	Analog	High speed differential inputs of two idle RocketIO transceiver ports. Bit 0 is the input to the idle transceiver in XBERT 0. Bit 1 is the input to idle transceiver in XBERT 1.
serial_p_out_idle serial_n_out_idle	Out Out	2 2		High speed differential outputs of two idle RocketIO transceiver ports. Bit 0 is the output from the idle transceiver in XBERT 0. Bit 1 is the output from the idle transceiver in XBERT 1.
total_frames_out	Out	96	Buffered RocketIO Transceiver Recovered Clock	Number of total frames received since link was established. Lower 48 bits are outputs from XBERT 0. Upper 48 bits are outputs from XBERT 1.
dropped_frames_out	Out	64		Number of frames received with error since link was established. Lower 32 bits are outputs from XBERT 0. Upper 32 bits are outputs from XBERT 1.
bec_count_out	Out	64		Number of bit errors in received data since link was established. Lower 32 bits are outputs from XBERT 0. Upper 32 bits are outputs from XBERT 1.
error_factor_out	Out	80		Shortest gap between two consecutive errors expressed in frames. This number is used to calculate the precision and confidence numbers of the bit-error rate. The lower 40 bits are outputs from XBERT0. The upper 40 bits are outputs from XBERT1.
status	Out	29		XBERT status outputs. Refer to Table 5 for more information.
LEDs_out	Out	16	Internal slow clock running at a quarter frequency of the rt_clock_in	On-board LED outputs. Refer to Table 6 for more information.
recovered_clock_out	Out	3	N/A	Clock outputs. Bit 0 and 1 are the transceiver recovered clocks from the XBERT 0 and XBERT 1, respectively. Bit 2 is the buffered transceiver reference clock output.

Table 3: **DIP_switch_in Port on Two-Channel XBERT**

Bit	Target XBERT	Function	Description	Default Value
0	XBERT0	Active MGT Tx Inhibit	If a logic High, inhibits the active RocketIO transceiver transmitter in the XBERT.	0
8	XBERT1			
1	XBERT0	Active MGT Power Down	If a logic High, shuts down both the receiver and the transmitter of the active RocketIO transceiver in the XBERT.	0
9	XBERT1			
[3:2]	XBERT0	Active MGT Loopback	Selects one of the two programmable loopback modes of the active RocketIO transceiver in the XBERT. The higher bit is for serial loopback and the lower bit is for internal parallel loopback.	0
[11:10]	XBERT1			
[7:4]	XBERT0	Pattern ID	Selects one of the PRBS/Clock patterns to load in the XBERT.	0
[19:16]	XBERT1			
13	XBERT0	Active MGT Error Insert	If a logic High, toggles all the bits in the current outgoing frame in the XBERT to result in 20-bit errors at the receiver.	0
14	XBERT1			
15	Both XBERTs	Clock Select	Selects BREFCLK or BREFCLK2 as the RocketIO transceiver reference clock. If a logic Low, uses BREFCLK. If a logic High, uses BREFCLK2.	0
20	XBERT0	Idle MGT Power Down	If a logic High, shuts down both the receiver and the transmitter of the idle RocketIO transceiver in XBERT.	0
21	XBERT1			
22	XBERT0	Idle MGT Tx Inhibit	If a logic High, inhibits the idle RocketIO transceiver transmitter in XBERT.	0
23	XBERT1			
[25:24]	XBERT0	Idle MGT Loopback	Selects one of the two programmable loopback modes of the idle RocketIO transceiver in XBERT. The higher bit is for serial loopback and the lower bit is for internal parallel loopback.	0
[27:26]	XBERT1			
[31:28]	Both XBERTs	Idle MGT Programmable Delay	Sets the programmable delay of 0 to 15 clock cycles between the pattern generator and the idle RocketIO transceivers in both XBERTs.	0
12	Reserved			

Table 4: **Push_buttons_in Port on Two-Channel XBERT**

Bit	Function	Description	Default Value
0	Master Reset	If asserted, reset all RocketIO transceivers in two-channel XBERT and reset the DCM generating the internal slow clock for the board interface.	0
1	XBERT0 Tx Reset	If asserted, reset the transmitter in XBERT0 and send an initialization sequence to the receiver. The detection of this initialization sequence will cause the receiver to reset as well.	0
2	XBERT1 Tx Reset	If asserted, reset the transmitter in XBERT1 and send an initialization sequence to the receiver. The detection of this initialization sequence will cause the receiver to reset as well.	0
3	Rx Reset	If asserted, reset all frame counters in both XBERTs.	0

Table 5: **Status Port on Two-Channel XBERT**

Bit	Target XBERT	Function	Description
0	XBERT0	Data Detect	Asserted when comma data is detected in the data stream. It is normal to see commas during a PRBS sequence.
8	XBERT1		
1	XBERT0	Tx Reset Detect	Asserted when a Tx reset is in progress. It indicates the transmitter is sending the initialization sequence to the receiver.
9	XBERT1		
2	XBERT0	Abort	Asserted when ongoing bit-error rate test has been aborted due to a high bit-error rate detected. A high bit-error rate is simply classified by detecting two consecutive errors in the data stream. The frame counters are still operating when Abort occurs. However the bit-error rate measurement based on these frame counters is not likely to be accurate because of a possible out-of-sync over the link under such high error rate.
10	XBERT1		
3	XBERT0	Link	Asserted when link between the transmitter and the receiver is established.
11	XBERT1		
[5:4]	XBERT0	Active MGT Loopback	Indicates if the active RocketIO transceiver is running in a loopback mode. The higher bit is for serial loopback and the lower bit is for internal parallel loopback.
[13,12]	XBERT1		
6	XBERT0	Active MGT Power Down	Asserted when the active RocketIO transceiver is shut down.
14	XBERT1		
7	XBERT0	Active MGT Tx Inhibit	Asserted when the active RocketIO transceiver transmitter is inhibited.
15	XBERT1		
16	XBERT0	Overflow	Asserted when the total bit error counter (bec_count_out) overflows.
17	XBERT1		
[20:18]	Both XBERTs	Configuration Mode	The software application uses these preset bits to identify the configuration of the RocketIO transceiver placement implemented in the design.
[22:21]	XBERT0	Idle MGT Loopback	Indicates if the idle RocketIO transceiver is running in a loopback mode. The higher bit is for serial loopback and the lower bit is for internal parallel loopback.
[26:25]	XBERT1		
23	XBERT0	Idle MGT Power Down	Asserted when the idle RocketIO transceiver is shut down.
27	XBERT1		
24	XBERT0	Idle MGT Tx Inhibit	Asserted when the idle RocketIO transceiver transmitter is inhibited.
28	XBERT1		

Table 6: **LEDs_out port on Two-Channel XBERT**

Bit	Target XBERT	Function	Description
0	XBERT0	Link	"ON" – Link is up.
8	XBERT1		"OFF" – Link is down.
1	XBERT0	Tx Reset Detect	"ON" – Indicates the XBERT transmitter is sending the initialization sequence.
9	XBERT1		"OFF" – XBERT transmitter is in normal operation.
2	XBERT0	Data Detect	"Blink" – Indicates a comma data is detected in the data stream.
10	XBERT1		

Table 6: **LEDs_out port on Two-Channel XBERT *(Continued)***

Bit	Target XBERT	Function	Description
3	XBERT0	Error Detect	"Blink" – Indicates a bit error was detected in a frame.
11	XBERT1		
4	XBERT0	Abort	"ON" – Indicates that bit-error rate test has been aborted due to a high bit-error rate.
12	XBERT1		"OFF" – No consecutive error is detected. Frame counters are trustworthy.
5	XBERT0	Frame Dropped	"ON" – One or more frames has been dropped due to bit errors.
13	XBERT1		"OFF" – No dropped frame so far.
6	XBERT0	Tx Inhibit	"ON" – RocketIO transceiver Tx inhibit input is a logic High.
14	XBERT1		"OFF" – RocketIO transceiver Tx inhibit input is a logic Low.
7	XBERT0	Power Down	"ON" – RocketIO transceiver PowerDown input is a logic High.
15	XBERT1		"OFF" – RocketIO transceiver PowerDown input is a logic Low.

Table 7: **XBERT Settings on RocketIO Transceiver Attributes**

Attribute	Value	Description
ALIGN_COMMA_MSB	True	Controls the alignment of detected commas within the transceivers 2-byte wide data path. XBERT assumes aligning comma with 20-bit alignment range.
RX_DECODE_USE	False	This determines if the 8B/10B decoding is bypassed. XBERT bypasses the 8B/10B.
RX_BUFFER_USE	True	Use the Rx buffer.
TX_BUFFER_USE	True	Use the Tx buffer.
RX_DATA_WIDTH TX_DATA_WIDTH	2	Use the 2-byte data path. XBERT is configured only to work in two- byte mode.
CLK_CORRECT_USE	False	Do not use clock correction logic. XBERT does not support clock correction.
CHAN_BOND_MODE	OFF	Do not use channel bonding. XBERT does not support channel bonding.
TX_CRC_USE RX_CRC_USE	False	XBERT do not use CRCs because these options will insert spurious data into the PRBS causing errors.
PCOMMA_DETECT	True	Enable detection of the PCOMMA (plus-comma). This comma is contained in the initialization sequence in link initialization. Without this option the XBERT will not link.
MCOMMA_DETECT	False	Do not detect the MCOMMA (minus-comma).
SERDES_10B	False	Denotes whether the reference clock runs at 1/20 (full rate) or 1/10 (half rate) the serial bit rate. XBERT does not support the half rate mode.
RX_CLK_V_SEL	1	Select BREFCLKs for 2.5 Gb/s or greater serial speeds. XBERT assumes RocketIO transceiver running at 2.5 Gb/s or higher and connects the clock to BREFCLK ports on all transceivers.

Notes:

1. Attributes not listed in this table are set to the default values defined in the GT_CUSTOM cell model.

PRBS Pattern Generation

Bit-error measurements are important means to assess the performance of digital transmission. It is necessary to specify reproducible test sequences that simulate real traffic as closely as possible. Reproducible test sequences are also a prerequisite to perform end-to-end measurement. Pseudo-random bit sequences (PRBS) with a length of $2^n - 1$ bits are the most common solution to this problem.

PRBS pattern is produced using a linear-feedback shift register (LFSR) with appropriate feedback. If the LFSR has n stages, the maximum sequence length will be $2^n - 1$ bits. If the digital signal is directly taken from the output of the LFSR (non-inverted signal) the longest string of consecutive ZEROs will be equal n – 1. If the signal is inverted, n consecutive ZEROs and n – 1 consecutive ONEs will be produced. In addition to strings of consecutive ZEROs and ONEs, PRBS pattern will contain any possible combination of ZEROs and ONEs within a string length depending on n.

There are two types of LFSR implementation for a certain polynomial that yields the equivalent result. *Fibonacci* LFSR or Type-I LFSR uses exclusive OR (XOR) gates outside the shift register loop. *Galois* LFSR, or Type-II LFSR uses exclusive OR gates inside the shift register chain. An implementation of multiple stage LFSR to produce PRBS pattern can be interpreted by a polynomial in a math perspective. For example, a polynomial $x^{15} + x^{14} + 1$ can represent *Fibonacci* LFSR implementation of a fifteen-stage shift register whose 14th and 15th stage outputs are added in a modulo-two addition stage, and the result is fed back to the input of the first stage. A polynomial $1 + x^{14} + x^{15}$ can represent equivalent *Galois* LFSR implementation from the previous example. For LFSRs with only a few taps, the *Fibonacci* implementation will generally achieve a faster clock speed than its *Galois* counterpart. Although faster for a small number of taps, the *Fibonacci* implementation's performance degrades as the number of taps increases. The *Galois* implementation, however, sees hardly any performance loss with an increase in the number of taps. PRBS pattern generator designed in XBERT deploys *Galois* LFSR implementation.

The PRBS pattern generator designed in XBERT implements eight different polynomials specified in ITU-T Recommendation O.150[4]. The ITU (International Telecommunication Union) is an international organization within the United Nations System where governments and the private sector coordinate global telecom networks and services. The ITU-T Recommendation O.150 contains general requirements applicable to instrumentation for performance measurements on digital transmission equipment. When setting the 4-bit Pattern ID on DIP_switch_in port on the two-channel XBERT module, the PRBS pattern generator will activate the corresponding LFSR. Such LFSR will generate PRBS pattern for the specific polynomial. By alternating PRBS patterns XBERT can produce different level of line stress on the RocketIO transceivers. Among eight PRBS patterns there are four of them using inverted patterns. ITU-T considers the inverted patterns as being more stressful than non-inverted patterns when testing the clock recovery circuit in the network terminating devices.

In addition to ITU-T PRBS patterns, XBERT implements several other PRBS and clock patterns. It also supports a user-defined pattern. Table 8 summarizes all the patterns and their respective polynomials implemented in XBERT.

Table 8: **PRBS Patterns and Polynomials**

ID [3:0]	Pattern or Polynomial	Length of Sequence (bits)	Consecutive Zeros	Citation and Notes
0	10101010...	2	0	This pattern can be used as a clock pattern to generate up to 1.5625 GHz differential clock on the transceiver serial outputs. This pattern can also be used as a high-frequency test pattern defined in IEEE STD 802.3 - 2002[5].
1	5 ones 5 zeros	10	5	This pattern can be used as a clock pattern to generate up to 312.5 MHz differential clock on the transceiver serial outputs. This pattern can also be used as a low-frequency test pattern defined in IEEE Std. 802.3 - 2002.
2	10 ones 10 zeros	20	10	This pattern can be used as a clock pattern to generate up to 156.25 MHz differential clock on the transceiver serial outputs. This clock output is also capable of triggering an external oscilloscope.
3	$1+x^6+x^7$ (non-inverted signal)	2^7-1	7	N/A
4	$1+x^5+x^9$ (non-inverted signal)	2^9-1	8	ITU-T Recommendation O.150 section 5.1
5	$1+x^9+ x^{11}$ (non-inverted signal)	2^11-1	10	ITU-T Recommendation O.150 section 5.2
6	$1+x^{14}+ x^{15}$ (inverted signal)	2^15-1	15	ITU-T Recommendation O.150 section 5.3. It is one of the recommended test patterns in SONET specification
7	$1+x^3+ x^{20}$ (non-inverted signal)	2^20-1	19	ITU-T Recommendation O.150 section 5.4. It is one of the recommended test patterns in SONET specification
8	$1+x^{17}+ x^{20}$ (non-inverted signal)	2^20-1	14	ITU-T Recommendation O.150 section 5.5. ITU-T recommends forcing the output bit of this polynomial to be a ONE whenever the next 14 bits are all ZEROs so that the number of consecutive ZEROs will decrease to 14.
9	$1+x^{18}+ x^{23}$ (inverted signal)	2^23-1	23	ITU-T Recommendation O.150 section 5.6. It is one of recommended test patterns in SONET specification
10	$1+x^{27}+ x^{29}$ (inverted signal)	2^29-1	29	ITU-T Recommendation O.150 section 5.7.
11	$1+x^{28}+ x^{31}$ (inverted signal)	2^31-1	31	ITU-T Recommendation O.150 section 5.8. This is a recommended PRBS test pattern for 10 Gigabit Ethernet. See IEEE Std. 802.3ae - 2002[6].
12	$1+x^{10}+x^{30}+x^{31}+x^{32}$ (non-inverted signal)	2^32-1	32	N/A
13	User defined pattern	1 to 20	0 to 20	This pattern consists of two 10-bit values, which can be picked from the 8B/10B table, or any value defined by the user.
14,15	Reserved			

PLB XBERT

The PLB XBERT is designed by connecting a two-channel XBERT module to a Xilinx PLB IPIF slave SRAM module, as illustrated in Figure 8. The PLB IPIF provides standardized PLB bus connections on one side and SRAM-like connections on the other side. The PLB IPIF makes it very easy to attach the two-channel XBERT module to the PLB CoreConnect Bus and uses very little glue logic. The PLB IPIF used in PLB XBERT provides a slave SRAM protocol. The version of the PLB IPIF used by the PLB XBERT only supports PLB non-burst memory transactions (PLB_size[0:3] = 0000, PLB_type[0:2] = 000) of one to four bytes. It ignores all other PLB transaction sizes and types.

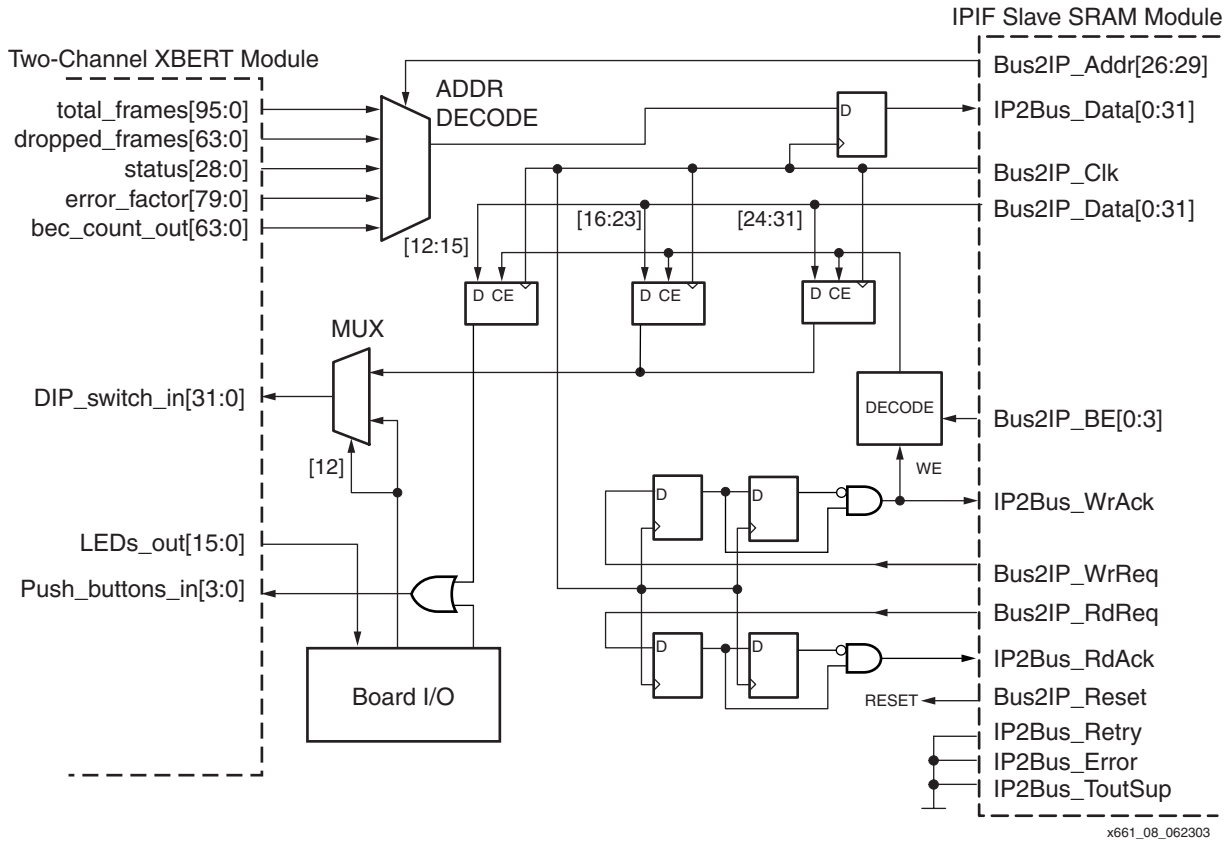


Figure 8: **Two-Channel XBERT PLB Interface**

Clock Domain

The interface is synchronized on Bus2IP_Clk, the PLB clock.

Read/Write Acknowledgement

The IPIF module only supports single read or write operation on the PLB. Proper read and write acknowledgement are required from the attached device. The two-channel XBERT PLB interface uses a simple technique to generate the acknowledgement responses. The acknowledgement of read and write cycles is handled by generating a single clock delayed envelope for the respective request signal. This envelope is also used to qualify the write transaction on the two-channel XBERT module. Any slave device attached on the PLB is assigned a base address so that read or write request can be distinguished from one slave device to another.

Read Operation

The IPIF module compares the read address with the base address of the two-channel XBERT module before it issues the read request to the XBERT module. The data outputs of the XBERT module consist of one 96-bit total frame counter value, one 80-bit error factor value, one 64-bit dropped frame counter value, one 64-bit total bit error counter value, and one 29-bit status value. The Bus2IP_Addr bus is decoded to select multiple 32-bit vectors from the XBERT module outputs. Because the IPIF module uses a 32-bit data path and it always provides byte addresses, the lower 2 address lines on Bus2IP_Addr are left unconnected, leaving the next four address lines (Bus2IP_Addr[26:29]) to provide up to sixteen 32-bit words. The output from the data multiplexer is registered at Bus2IP_Clk and then fed into IP2Bus_Data[0:31]. Even though the XBERT refreshes the frame counter outputs at the fast BREFCLK clock speed (e.g., 156.25 MHz), the CPU can only sample the data at PLB clock speed (i.e., 50 MHz).

Write Operation

The IPIF module compares the write address with the base address of the two-channel XBERT module before it issues the write request to the XBERT module. Because there is only a single 32-bit word to write to the XBERT, multiplexing on the Bus2IP_Addr in each write transaction is not necessary. However, the reference design requires the ability to write individual bytes of data to the XBERT module. The IPIF module provides the byte enables by the way of the Bus2IP_BE signals. To register individual bytes from the 32-bit data bus, each bit of the Bus2IP_BE signal is ANDed with the write enable (WE) to create an enable signal on each register. The data inputs of the XBERT module consist of a 32-bit DIP switch input and a 4-bit push button input. The CPU can write data to these ports to override the inputs from the board I/O (i.e., DIP switches and push buttons) once the board I/O is disabled. Inputs from the board I/O or the CPU are selected by bit 12 of the external DIP switch inputs (DIPS [12]).

Other Devices on PLB

The PPC405 processor, PLB bus infrastructure and arbiter, PLB 16550 UART, PLB BRAM interface controller, and BRAM block used in this reference design are standard components provided in the Embedded Development Kit (EDK)[1]. The reference design does not have an interrupt controller. The UART interrupt output is directly fed into the CPU's non-critical interrupt input. The CPU's critical interrupt input is tied Low.

With the PLB 16550 UART controller, an evaluation core supplied in the EDK, the user can perform both functional and timing simulation, and download and configure the design into hardware. After a bitstream is generated and downloaded, the IP is fully functional for six to eight hours. If a "time out" occurs, re-download and configure the FPGA again. A copy of the EDK and instructions for generating a full license for the UART 16550 are included in the purchase of a PLB 16550 UART controller. For more information on Xilinx Processor Peripheral IP LogiCOREs, please visit the Embedded Development Kit product page at: http://www.xilinx.com/edk

Software Implementation

Overview

The RocketIO transceiver BERT reference design provides a software application called MGT_DEMO to work with the hardware platform. Figure 9 illustrates the flow diagram of the MGT_DEMO software application.

On power-up or after a full reconfiguration of the FPGA, the software performs a variety of setup operations before the start-up screen is displayed on the terminal window. After entering the main menu, set the RocketIO attributes TX_PREEMPHASIS and TX_DIFF_CTRL, then run the BERT test, or query some of the Virtex-II Pro configuration registers. XAPP662[2] gives details on using software to set RocketIO Attributes.

After entering the RocketIO BERT test from the main menu, the MGT_DEMO software first executes XBERT reset sequence, then cycles through a routine to read the two-channel

XBERT statistics registers, and print the register values (including the frame counter and status values) to the UART. Although the XBERT updates the frame counter and status outputs at the fast BREFCLK (typically 156.25 MHz) clock, XBERT statistics registers are only updated at the PLB clock (typically 50 MHz). Moreover the MGT_DEMO software only samples the statistics registers approximately every four seconds. In addition the software application reads the UART input then executes the corresponding command that is issued by the user on the PC terminal.

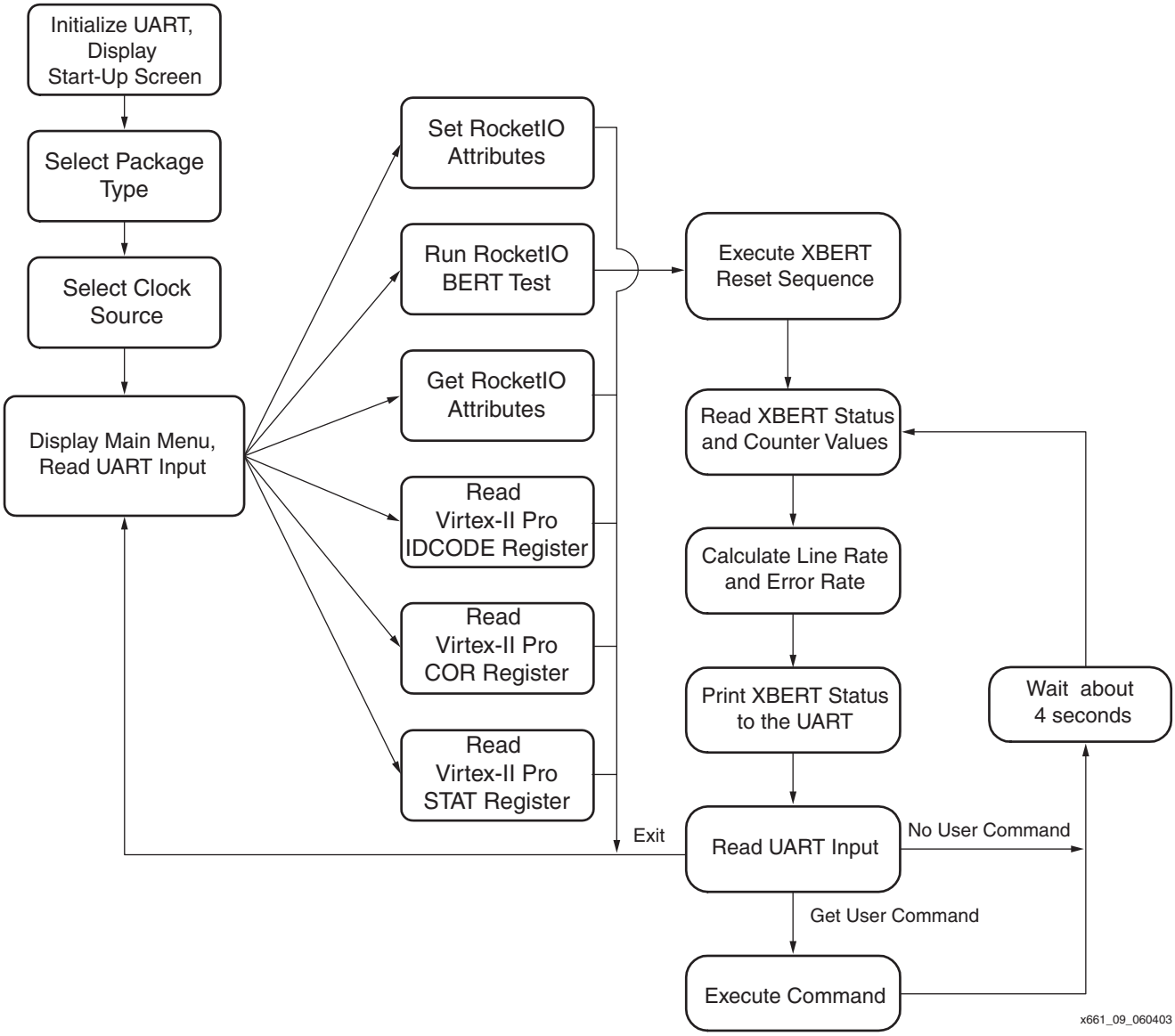


Figure 9: **Flow Diagram of MGT_DEMO Software Application**

The application is linked with the board support package (BSP) supplied in the Embedded Development Kit[1]. The BSP offers the interface to peripheral devices and to low-level PowerPC405 core functions. The BSP also provides functionality that allows the C library to access the hardware. For more information consult the Embedded-System Tools Guide: Stand-Alone Board Support Package[1].

Software instruction is loaded into the BRAM on the PLB. UART, ICAP, and the two-channel XBERT are memory mapped to the PLB. Table 9 lists the software memory map and base addresses of these PLB devices.

Table 9: **PLB Device Memory Map**

Device	Address Boundaries		Size
	Upper	Lower (Base Address)	
PLB Slave 0 – BRAM	FFFFFFFF	FFFF0000	64 KBytes
PLB Slave 1 – UART	A0003FFF	A0002000	8 KBytes
PLB Slave 2 – Two-Channel XBERT	A00100FF	A0010000	256 Bytes
PLB Slave 3 – ICAP	A01000FF	A0100000	256 Bytes

UART Function on the BERT Test Menu

The application uses basic UART functions provided in the BSP library as well as some supplemental functions (e.g., uart_printf) provided in the MGT_DEMO application to communicate through the UART. The software initializes the UART with the baud rate, the number of data bits, the desired parity, and the number of stop bits. It supports I/O on the UART allowing user navigation around different menus. Once the user enters the RocketIO BERT test menu, the software prints a page showing the frame counter numbers and status of the two-channel XBERT to the UART. It also reads characters from the UART and calls the subroutine for the specific command. Table 10 lists all the supported characters and corresponding commands on the BERT test menu. In this implementation the software continuously polls on the UART for about four seconds between each print. Typing ahead more than one character during the polling results in multiple executions of corresponding commands. This can alter result (e.g., switch the clock source twice).

Table 10: **Supported Characters and Software Commands**

Character	Target XBERT	Software Command
"1"	XBERT0	Toggle active RocketIO transceiver Tx inhibit. Type once to turn on Tx inhibit. Type again to turn it off.
"4"	XBERT1	
"2"	XBERT0	Toggle active RocketIO transceiver power down. Type once to power down. Type again to power up.
"5"	XBERT1	
"3"	XBERT0	Toggle active RocketIO transceiver serial loopback. Type once to turn on serial loopback. Type again to turn it off.
"6"	XBERT1	
"i" or "I"	XBERT0	Toggle idle RocketIO transceiver Tx inhibit. Type once to turn on Tx inhibit. Type again to turn it off.
"j" or "J"	XBERT1	
"o" or "O"	XBERT0	Toggle idle RocketIO transceiver power down. Type once to power down. Type again to power up.
"k" or "K"	XBERT1	
"p" or "P"	XBERT0	Toggle idle RocketIO transceiver serial loopback. Type once to turn on serial loopback. Type again to turn it off.
"l" or "L"	XBERT1	
"7"	XBERT0	Toggle active RocketIO transceiver error insertion. Type once to turn on error insertion. Type again to turn it off.
"8"	XBERT1	
"9"	Both XBERTs	Switch clock source. Type once to use BREFCLK2 for the RocketIO transceivers. Type again to switch back to BREFCLK.

Table 10: **Supported Characters and Software Commands *(Continued)***

Character	Target XBERT	Software Command
"0"	Both XBERTs	Reset two-channel XBERT. This command will execute XBERT reset sequence.
"+" or "="	Both XBERTs	Switch to the next PRBS/clock pattern on both XBERTs. If it reaches the last pattern, it will roll back to the first one.
"–" or "_"	Both XBERTs	Switch to the previous PRBS/clock pattern on both XBERTs. If it reaches the first pattern, it will move to the last pattern.
"d" or "D"	XBERT0	Switch to the next PRBS/clock pattern on associated XBERT only. If it reaches the last pattern, it will roll back to the first one.
"c" or "C"	XBERT1	
"a" or "A"	XBERT0	Switch to the previous PRBS/clock pattern on associated XBERT only. If it reaches the first pattern, it will move to the last one.
"z" or "Z"	XBERT1	
"." or ">"	Both XBERTs	Increase programmable delay on the idle RocketIO transceiver. It wraps around when it overflows.
"," or "<"	Both XBERTs	Decrease programmable delay on the idle RocketIO transceiver. It wraps around when it underflows.
ESC (0x1b)	Go back to the main menu	

XBERT Reset Sequence

To establish a link between two XBERT modules the software executes a XBERT reset sequence on both XBERT modules. The XBERT reset sequence consists of a master reset on the RocketIO transceivers followed by multiple TX resets on the XBERT transmitter (GigabitBER_TX), and a final RX reset on the XBERT receiver (GigabitBER_RX) to reset all the counters. Since XBERT requires long delays after any master reset or TX reset, it is easy to deploy these delays in software using the CPU sleep function. Figure 10 illustrates the flow diagram of the XBERT reset sequence.

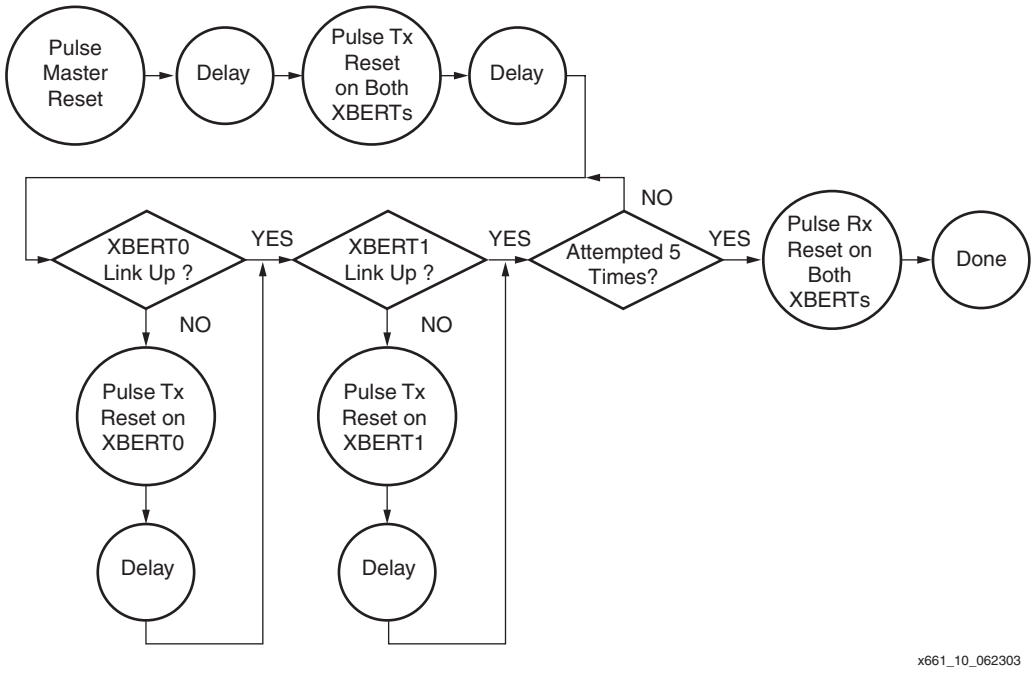


Figure 10: **Flow Diagram of XBERT Reset Sequence**

XBERT Statistics and Control Registers

A complete list of supported two-channel XBERT control and statistics registers is described in Table 11 and Table 12. The bit mapping of these registers are based on big Endian.

Table 11: **Two-Channel XBERT Statistics Register Address Map**

Address	Description	Attributes	Size (bits)
A0010000	Higher 32 bits of XBERT 0 total received frame count value	Read	32
A0010004	Lower 32 bits of XBERT 0 total received frame count value	Read	32
A0010008	XBERT 0 total dropped frame count value	Read	32
A001000C	Higher 32 bits of XBERT 1 total received frame count value	Read	32
A0010010	Lower 32 bits of XBERT 1 total received frame count value	Read	32
A0010014	XBERT 1 total dropped frame count value	Read	32
A0010018	XBERT status value	Read	32
A001001C	Higher 32 bits of XBERT 0 error factor value	Read	32
A0010020	Lower 32 bits of XBERT 0 error factor value	Read	32
A0010024	Higher 32 bits of XBERT 1 error factor value	Read	32
A0010028	Lower 32 bits of XBERT 1 error factor value	Read	32
A001002C	XBERT 0 total bit-error count value	Read	32
A0010030	XBERT 1 total bit-error count value	Read	32

Table 12: **Two-Channel XBERT Control Register Address Map**

Address	Description	Attributes	Size (bits)
A0010000	Register input to override DIP switch and push button inputs on board I/O. Bits 15 down to 12 of this register correspond to the four push button bits 3 down to 0 in Table 4. Bits 11 down to 0 correspond to the combination of the bits [19:16] and [27:20] in Table 3.	Write	16
A0010002	Register input to override DIP switch inputs on board I/O. Bits 15 down to 0 of this register correspond to combination of the bits [7:0] and [15:8] in Table 3.	Write	16
A0010004	Register input to override DIP switch inputs on board I/O. Bits 15 down to 12 of this register correspond to the bits 31 down to 28 in Table 3. The other bits of this register are reserved.	Write	16

Calculation of RocketIO Transceiver Line Rate

The software calculates line rates on both RocketIO transceivers by the following equation. It first subtracts two sampled total received frame counter numbers (total_frame, total_frame_prv). Then it divides the result by the CPU cycles between the two sampling points (delta_cycles), and multiplies the result with parameters such as CPU_PERIOD and FRAME_BYTE_LENGTH to get the line rate in megabits per second (Mb/s). FRAME_BYTE_LENGTH is a user-defined parameter. It gives the number of bytes in each frame. There are two in the reference design. CPU_PERIOD is another user-defined parameter that gives the CPU clock (typical 200 MHz) in nanoseconds. Using functions (e.g., mltime_gettime) provided in the software, a user can acquire the CPU cycles for executing a group of instructions.

$$LineRate(Mbps) = \frac{(total\ frames - total\ frames\ prv) \times 8 \times FRAME\ BYTE\ LENGTH \times 1000}{delta\ cycles \times CPU\ PERIOD}$$

Calculation of RocketIO Transceiver Bit-Error Rate

RocketIO transceiver bit-error rate can be calculated through the total bit error counter and total frame counter outputs from the XBERT using the following simple equation.

$$BitErrorRate = \frac{total\ bit\ errors}{total\ frames \times 20}$$

The following section describes how to calculate the precision and confidence of the bit-error rate test. It also illustrates the derivation of precision and confidence numbers from elementary principles of stochastic methods.

Derivation of the Bit-Error Rate

The bit-error rate measured by the XBERT is an estimate of the true bit-error rate. For completeness when reporting any statistical value you must also report the precision of your measurement and the probability your measurement is within that precision (the confidence). This section will explain how to calculate the precision and confidence of your bit-error rate test. Also this section will illustrate the derivation of these two numbers from elementary principles of stochastic methods. It assumes a basic working understanding of random variables.

The next section uses the following variables:

p = The probability of a bit error

n = total_frames_out $\times$ 20

e = dropped_frames_out

k = error_factor_out $\times$ 20

$\in$ = Interval of certainty (i.e., $\pm\in$)

c = The confidence

Some Useful Random Variables

The Bernoulli random variable describes the outcome of a Bernoulli trial. The most famous example of a Bernoulli trial is a coin flip. In the case of bit errors an outcome of 0 means the data is fine, and an outcome of "1" indicates an error. An error occurs with probability p. Thus, p is the theoretical bit-error rate.

Definition of the Bernoulli Random Variable

Domain

$$B_j \in \{0,1\} \quad (1.1)$$

Expected Value and Variance

$$E[B_j] = p \quad (1.2)$$

$$V[B_j] = p(1 - p) \quad (1.3)$$

Probability Density Function

$$P[B_j = 1] = p \quad (1.4)$$

The Binomial random variable describes the outcome of two or more Bernoulli trials. The "0" or "1" outcomes are added together to produce a number X_n representing the number of occurrences of a bit error in n trials. The XBERT in effect does this by keeping track of the number of trials and the number of bit errors.

Definition of the Binomial Random Variable

Domain

$$X_n \in \{0, 1, 2, ..., n\} \quad (2.1)$$

Definition

$$X_n = \sum_{j=1}^{n} B_j \quad (2.2)$$

Expected Value and Variance

$$E[X_n] = nE[B_j] = np \quad (2.3)$$

$$V[X_n] = nV[B_j] = np(1 - p) \quad (2.4)$$

Probability Density Function

$$P[X_n = x] = \binom{n}{x} p^x (1 - p)^{(n - x)} \quad (2.5)$$

The geometric random variable is similar to the binomial random variable, except that instead of counting the number of bit errors, the geometric random variable counts the number of good bits between bit errors. The error_figure_out gives the minimum value of G measured in a given test period. As shown later, this number is very useful in calculating a bit-error rate.

Definition of the Geometric Random Variable

Domain

$$G \in Z^{+} \tag{3.1}$$

Expected Value and Variance

$$E[G] = \frac{1}{p} \tag{3.2}$$

$$V[G] = \frac{1-p}{p^2} \tag{3.3}$$

Probability Density Function

$$P[G = x] = p\,(1-p)^{x-1} \tag{3.4}$$

Cumulative Distribution Function (CDF)

$$P[G \leq x] = 1 - (1-p)^{x} \tag{3.5}$$

Sampling

To measure the bit-error rate, samples of bit errors over a finite time interval are taken, then the results are used to make an assertion about the system at all times. This can be justified if the data gathered over the finite interval is representative of the operation of the system and the interval is long enough. A special case of the Binomial random variable is used to understand how the interval effects the measurement.

Definition of the Sampled Binomial Random Variable

Domain

$$0 \le M_n \le n \tag{4.1}$$

Definition

$$M_n = \sum_{j=1}^{n} \frac{B_j}{n} \tag{4.2}$$

Expected Value

$$E[M_n] = E\left[\sum_{j=1}^{n} \frac{B_j}{n}\right] = \frac{1}{n}\sum_{j=1}^{n} E[B_j] = p \tag{4.3}$$

The expected value of M_n is p. The sampling process is an unbiased estimator of the bit-error rate.

Variance

$$V[M_n] = E\left[\sum_{j=1}^{n}\left(\frac{M_j - E[M_n]}{n}\right)^2\right] \tag{4.4}$$

$$V[M_n] = \frac{1}{n^2}\sum_{j=1}^{n} E\left[(M_j - E[M_n])^2\right] \tag{4.5}$$

Substituting for the variance of the binomial random variable (2.4) produces:

$$V[M_n] = \frac{1}{n^2}\sum_{j=1}^{n} V[X_n] = \frac{p(1-p)}{n} \tag{4.6}$$

From equation (4.6), the variance of the sample gets smaller with the number of samples. The larger the sample taken, the more representative the sample.

Making an Estimate of the Bit-Error Rate

To derive the precision and confidence of the experiment, an estimate is made of the upper bound of p. The estimate, p_0, is derived using the binomial random variable. The calculation requires specification of a confidence value. That value, c_1, is used later to calculate the overall confidence value.

Calculation of p_0

Using the CDF of the binomial random variable (3.5), solving for p and substituting the variables yields:

$$p_0 = 1 - (1 - c_1)^{\frac{1}{k}} \tag{5.1}$$

From Equation (5.1), the value of p is less than or equal to p_0 with the probability of c_1. The higher the value of c_1 the looser the bound becomes, and the calculation becomes more credible.

Calculating the Precision

Chebychev's inequality is used to calculate the value of ε (the precision). Similar to the calculation of p_0, calculating ε requires a confidence value. This value is denoted as c_2.

Derivation of ε

Chebychev's inequality:

$$P[\left|M_n - E[M_m]\right| \geq \varepsilon] \leq \frac{V[M_n]}{\varepsilon^2} \quad (6.1)$$

Substituting for the expected value and variance of the sampled binomial random variable (4.3) and (4.6) and also using c_2 to represent the confidence:

$$c_2 \leq \frac{p(1-p)}{n\varepsilon^2} \quad (6.2)$$

Equation (6.2) requires knowing the theoretical value of p. Instead, take advantage of the inequality and substitute p with the previously calculated upper bound p_0. Simple algebra then yields:

$$\varepsilon \leq \sqrt{\frac{p_0(1-p_0)}{c_2 n}} \quad (6.3)$$

Scientific quality results are calculated with these equations.

The Bit-Error Rate With Precision and Confidence

Calculate the upper bound:

$$p_0 = 1 - (1 - c_1)^{\frac{1}{k}} \quad (5.1)$$

Then calculate the bit-error rate and the precision:

$$BER = \frac{e}{n} \pm \sqrt{\frac{p_0(1-p_0)}{c_2 n}}$$

Then determine the confidence:

$$c = c_1 c_2$$

Design Configuration and Implementation Instructions

Installation of the Design and Tools

1. Install Xilinx ISE software, ModelSim SE, and EDK tools.
2. Install the SmartModel Library supplied in the Xilinx ISE software. The Xilinx Answer Record #15501 and #14019 gives further information regarding SmartModel/Swift Interface and installation of SmartModels.
3. Extract xapp661.zip into a directory <BERT_ROOT>.
4. Modify the `config.bash` (on Windows) or `config.csh` (on UNIX) file under <BERT_ROOT> to update all paths of the tools and libraries as specified in these files.
5. Call `config.bash` or `config.csh` script to setup environment on Windows or UNIX:

   On Windows, launch XYGWIN, enter <BERT_ROOT>, then type the following command:

   ```
   > source config.bash
   ```

   On UNIX, enter <BERT_ROOT>, then type the following command:

   ```
   > source config.csh
   ```

6. Run compxlin tool to compile simulation libraries (unisim, simprim, etc.) of Virtex-II Pro family for ModelSim. Library should be compiled into $MTI_LIBS directory defined in `config.bash` or `config.csh` file.

   The following are examples use the compxlib tool:

   On Windows: (In XYGWIN window)

   ```
   > compxlib -s mti_se -f virtex2p -l all -o $MTI_LIBS
   ```

   On UNIX:

   ```
   > compxlib -s mti_se -f virtex2p -l all -o $MTI_LIBS
   ```

7. Run `vmap_edk_libs` tool to compile behavioral libraries of IP cores provided in EDK for ModelSim. Library should be compiled into $edk_nd_libs directory defined in `config.bash` or `config.csh`.

   The following examples use the `vmap_edk_libs` tool:

   On Windows: (In XYGWIN window)

   ```
   > vmap_edk_libs.bat -install $edk_nd_libs
   ```

   On UNIX:

   ```
   > vmap_edk_libs -install $edk_nd_libs
   ```

8. After completing these steps, run step 7 again after restarting Windows or UNIX system.

Design Configuration

1. Select and copy MHS file to `system.mhs`.

Assumes <BERT_ROOT> is the root directory of an installation of the RocketIO transceiver BERT reference design. This reference design provides two MHS files, `<BERT_ROOT>/system_1cpu.mhs` for a single CPU platform (ML320 or ML321), and `<BERT_ROOT>/system_2cpu.mhs` for a dual CPU platform (ML323). Depending on the platform used, copy one of these MHS files to `<BERT_ROOT>/system.mhs`.

2. Modify `synth_defs.v` and/or `sim_defs.v`

The `<BERT_ROOT>/data/synth_defs.v` and `<BERT_ROOT>/testbench/sim_defs.v` files supply information on customizing design options. Several configuration variables in these files are listed below that are required to be set properly before synthesis and implementation, or functional simulation of the design.

Define only one of the following parameters at a time to configure the RocketIO clock source:

```
`define USE_DIFF_CLK            // IF this parameter is defined, XBERT will
   (default)                    // use differential clock inputs to drive
                                // the RocketIO transceivers.
`define NOT_USE_DIFF_CLK        // IF this parameter is defined, XBERT will
                                // use a single clock inputs to drive
                                // the RocketIO transceivers.
```

Define only one of the following parameters at a time to configure the RocketIO user clock generation:

```
`define NOT_USE_MGT_DCM         // If this parameter is defined, XBERT will
   (Default)                    // simply use the buffered reference clock
                                // as RocketIO User Clocks
`define USE_MGT_DCM             // If this parameter is defined, XBERT will
                                // use a dedicated DCM to generate RocketIO
                                // User Clocks
```

Define only one of the following parameters at a time to configure the XBERT:

```
`define XBERT_MGT_DEMO_BOARD// If this parameter is defined, XBERT
   (for implementation)     // will load its parameter set defined in
                            // BERT_CONFIG.v for implementation purposes.
`define XBERT_FOR_SIM_ONLY  // If this parameter is defined, XBERT
   (for simulation)         // will load its parameter set defined in
                            // BERT_CONFIG.v for simulation purposes.
```

Define only one of the following parameters at a time to customize Idle MGTs:

```
`define USER_USE_MGT_IDLE_yes // Implement two more MGTs as Idle
   (default)                  // MGTs. The placement of Idle
                              // MGTs depends on the placement of
                              // the Active MGT.
`define USER_USE_MGT_IDLE_no  // Do not implement Idle MGTs.
```

Define both of the following parameters at a time to specify the user defined pattern (Pattern ID = 13) in pattern generator.

```
`define USER_PATTERN_L 10'b0011111010// Lower 10-bit value of
                                     // user defined pattern
`define USER_PATTERN_H 10'b1100000101// Higher 10-bit value of
                                     // user defined pattern
```

Define one of the following parameters at a time to attach ChipScope ILA on one of the pattern followers. Otherwise, there will be no ChipScope ILA attached.

```
`define USE_ILA_0        // Attach ILA core on pattern follower in XBERT 0.
`define USE_ILA_1        // Attach ILA core on pattern follower in XBERT 1.
```

Define only one of the following parameters at a time to specify if the RocketIO transceivers in two-channel XBERTs are placed separately in the top and bottom banks

```
`define USER_USE_TOP_BOT_GT_no  // All MGTs will be placed on the same bank
   (default)                    // on the FPGA. This is a single-bank
                                // configuration.
`define USER_USE_TOP_BOT_GT_yes // MGTs will be split into top and bottom
                                // bank on the FPGA. This is a split-bank
                                // configuration.
```

3. Modify UCF files and implementation scripts

UCF files and implementation scripts supplied in the reference design assume the target device and platform is one of the following:

a. ML321 platform with a Virtex-II Pro XC2VP7 FPGA in an FF672 package
b. ML320 platform with a Virtex-II Pro XC2VP7 FPGA in an FG456 package
c. ML323 platform with a Virtex-II Pro XC2VP20 FPGA in an FF1152 package

To port this design to another device or hardware platform, modify the UCF files in <BERT_ROOT>/data and Makefile under <BERT_ROOT>.

4. ChipScope™ Pro ILA Core

The RocketIO BERT reference design embeds one ChipScope Pro 5.1i ILA core attached to a pattern follower in the two-channel XBERT module. The ILA core is used to capture data on the pattern follower, including 20-bit received data, 20-bit expected data, and a 1-bit lock (i.e., "data match") signal. An ILA core will be implemented in the design if a global parameter, USE_ILA_0 or USE_ILA_1, is defined in the `<BERT_ROOT>/data/synth_defs.v` file. Otherwise, the ILA core will be removed from the design to save space. A ChipScope Pro Analyzer project file is provided at `<BERT_ROOT>/chipscope/bert.cpj`.

Implementation Instructions Using Command Lines

1. On Windows:
   Launch XYGWIN supplied in the EDK, then go to the <BERT_ROOT> directory.
   On UNIX:
   Go to the <BERT_ROOT> directory
2. Customize the design properties by modifying the `<BERT_ROOT>/data/synth_defs.v` file, and copy one of the MHS files to `<BERT_ROOT>/system.mhs`.
3. Generate HDL wrapper files and NGC netlists by running the following command from the command line:

```
> make netlist
```

This script first calls `Platgen` in the EDK. `Platgen` creates the HDL wrapper files, then synthesizes the modules using XST, and creates the NGC netlists.

4. Generate software libraries and the software executable file by running the following command:

```
> make program
```

The script will first call `libgen` to generate the software libraries. It will then compile and link all software source code into the software executable ELF file.

5. Place and route the design and generate a bitstream by running the following command with the MGT placement option:

```
> make <target>
```

Valid <target> values include top03, top12, bot03, bot12, top0bot3, as described in the next section "Placement of Target RocketIO Transceivers".

The script will first create a UCF file for the system by concatenating several UCF files supplied <BERT_ROOT>/data, then run xflow and bitgen to create all the implementation files for the target FPGA. Finally, the script will run data2bram to populate BRAM with software code and generate the final bitstream at implementation/download.bit

The script builds the bitstream for the ML321 platform by default. Use one of the following commands to build the bitstreams for ML320 and ML323 platforms:

```
> make PLATFORM=ml320 PART=XC2VP7 PACKAGE=FG456 <target>
> make PLATFORM=ml323 PART=XC2VP20 PACKAGE=FF1152 <target>
```

Placement of Target RocketIO Transceivers

Placement of the RocketIO transceivers is specified when the design is placed and routed using the script Makefile. When passing a proper target to the Makefile, the script will pick the right UCF files from <BERT_BOOT>/data and assemble them into the final UCF file. The UCF file determines the placement of the transceivers. Table 13 lists all available placement targets.

In most cases, "make netlist" is not necessarily run to rebuild the netlist when generating a new bitstream with a different placement of the transceivers. Only run "make" with a different target option (top03, top12, bot03, or bot12). The exception is the option "top0bot3" because it requires regenerating the netlist.

Table 13: **Options for Placement of RocketIO Transceivers**

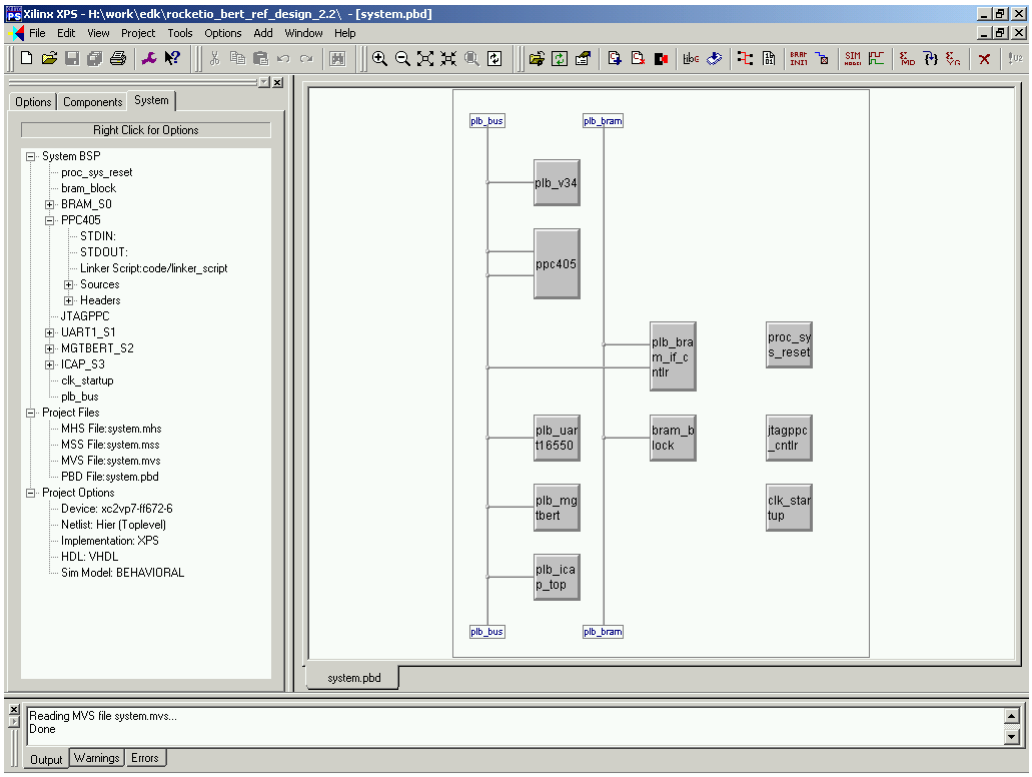
Option	Placement of RocketIO Transceivers
top03	Select MGT4 and MGT9 as active MGTs, and select MGT6 and MGT7 as idle MGTs.
top12	Select MGT6 and MGT7 as active MGTs, and select MGT4 and MGT9 as idle MGTs.
bot03	Select MGT21 and MGT16 as active MGTs, and select MGT19 and MGT18 as idle MGTs.
bot12	Select MGT19 and MGT18 as active MGTs, and select MGT21 and MGT16 as idle MGTs.
top0bot3	Select MGT4 and MGT16 as active MGTs, and select MGT6 and MGT18 as idle MGTs. User needs to first define USER_USE_TOP_BOT_GT_yes in `<BERT_ROOT>/data/synth_defs.v` to enable this placement, and rebuild the netlist.

Implementation Instructions Using Xilinx Platform Studio (XPS)

1. On UNIX systems, go to the <BERT_ROOT> directory.
   On Windows systems, launch XYGWIN supplied with the EDK, then go to the <BERT_ROOT> directory.
   Copy one of the MHS files to `<BERT_ROOT> system.mhs`.
2. Launch XPS in EDK and load system.xmp as follows:

   ```
   > xps system.xmp
   ```

The XPS GUI window will display a block diagram of this reference design (Figure 11).



x661_11_060603

Figure 11: **XPS Screen Shot of the RocketIO BERT Reference Design**

3. Launch the XYGWIN shell from the XPS menu:

   ```
   Tools -> Xygwin Shell
   ```

   Then complete the following steps in the XYGWIN shell:

   a. Customize the design configuration by modifying the <BERT_ROOT>/data/synth_defs.v file; then copy this file to <BERT_ROOT>/pcores/plb_mgtbert_v1_00_a/hdl/verilog/ as follows:

      ```
      > cp -f data/synth_defs pcores/plb_mgtbert_v1_00_a/hdl/verilog/
      ```

   b. Create the implementation directory if it does not exist, then copy the ChipScope ILA core netlists into this directory as follows:

      ```
      > mkdir implementation
      > cp chipscope/*.edn implementation
      > cp chipscope/*.ncf implementation
      ```

   c. Run the script "make_ucf.pl" to assemble a final UCF file for the system. This script will concatenate several UCF files supplied in <BERT_ROOT>/data and generate <BERT_ROOT>/implementation/system.ucf:

      ```
      > xilperl make_ucf.pl -b <platform> -t <target>
      ```

      Valid <platform> values include ml321, ml320, and ml323.

      Valid <target> values include top03, top12, bot03, bot12, top0bot3, as listed in the section "Placement of Target RocketIO Transceivers".

   d. Exit the XYGWIN shell.

4. Modify the target device from the XPS menu:

   ```
   Options -> Project Options -> Target Device
   ```

5. Build the netlist from the XPS menu:

```
Tools -> Generate Netlist
```

6. Compile the software program from the XPS menu:

```
Tools -> Generate Libraries
Tools -> Compile Program Sources
```

7. Generate the bitstream from XPS menu:

```
Tools -> Generate Bitstream
```

8. Update the bitstream to initialize the contents of the BRAM with software code and generate the final bitstream to implementation/download.bit:

```
Tools -> Update Bitstream
```

Functional Simulation Instructions

Functional simulation can only be run from command line mode.

1. On UNIX systems, go to the <BERT_ROOT> directory; On Windows systems, launch the XYGWIN shell supplied in EDK, then go to the <BERT_ROOT> directory.
2. Customize the design properties by modifying the `<BERT_ROOT>/testbench/sim_defs.v` file. Copy one of the MHS files to `<BERT_ROOT>/system.mhs`.
3. Generate software libraries and the software executable file by running the following command:

```
> make program
```

   The script will first call libgen to generate the software libraries. Then it will compile and link all software source code into the software executable ELF file. A simplified version of the MGT_DEMO software is used in SWIFT simulation. This allows the design to be simulated faster.

4. Compile the design files and invoke the simulator by running the following command:

```
> make sim
```

   The script will first run Platgen in the EDK to create the HDL wrapper files. Then it will run the data2bram application to create the BRAM initialization HDL file from the ELF and BMM files. The script will compile all the HDL files in ModelSim using mixed-language compilation, then start the ModelSim GUI to run SWIFT simulation for 100 µs.

Running the Bitstream

1. Install a terminal program on your PC. This reference design recommends Tera Term Pro v2.3, provided in the reference design package. Tera Term Pro is a free software terminal emulator. The version included in the package is for use with Windows 95, 98, ME, NT, and 2000.
2. Setup Tera Term Pro:
   a. Select Serial port (COM1 or COM2)
   b. Setup -> Terminal -> Terminal Size: 80X200
   c. Setup -> Serial Port -> Select COM1/COM2, 19200 Baud rate, 8-bit data, none parity, 1-bit stop, flow control none.
3. Connect a serial cable from the RS232 port on the board to the PC's COM1 or COM2 port.
4. Power up the board.
5. Download the bitstream. One should see a start-up screen in the terminal window.

Conclusion

This application note describes a RocketIO transceiver BERT reference design implemented in a Virtex-II Pro FPGA. This reference design demonstrates a 1.0 Gb/s to 3.125 Gb/s serial link between two RocketIO transceivers exchanging PRBS or a clock pattern. The reference design builds a simple PPC system using the Xilinx EDK that can be easily modified or extended. The hardware portion of the reference design is done in HDL and the software application is written in C language. The reference design can be downloaded from the Xilinx web site at:

http://www.xilinx.com/bvdocs/appnotes/xapp661.zip

Reference

1. Xilinx, Inc., " Embedded Development Kit", EDK 3.2, April 2003. (http://www.xilinx.com/edk)
2. Xilinx, Inc., XAPP662: "In-Circuit Partial Reconfiguration of RocketIO Attributes", v1.0, Jan. 2003. (http://www.xilinx.com/bvdocs/appnotes/xapp662.pdf)
3. Xilinx, Inc., "RocketIO Transceiver User Guide", v 2.0, March 2003. (http://www.xilinx.com/bvdocs/userguides/ug024.pdf)
4. ITU-T Recommendation O.150, "General Requirements for Instrumentation for Performance Measurements on Digital Transmission Equipment", May 1996.
5. IEEE Standard 802.3-2002, "Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications".
6. IEEE Standard 802.3ae-2002, "Amendment: Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation", August 2002.
7. Miron Abramovici, Melvin A. Breuer, Arthur D. Friedman, "Digital Systems Testing and Testable Design", 1990.
8. Xilinx, Inc., "Linear Feedback Shift Register v3.0", March 2003. (http://www.xilinx.com/ipcenter/catalog/logicore/docs/lfsr.pdf)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/13/03	1.0	Initial Xilinx release.
02/05/03	1.1	Edits to Table 2 and Figure 8.
04/14/03	1.2	Edits to Table 8.
06/25/03	2.0	Major revisions in all sections due to software changes (PDK to EDK). Added “Design Configuration and Implementation Instructions” section.
02/12/04	2.0.1	Corrected typographical/grammatical errors.
05/24/04	2.0.2	Updated URLs and clickable hyperlinks.