Verification of the Triple Modular Redundancy 8B/10B Encoder

A Component of the GBTX ASIC

CMS-doc-5691

Stephen Goadhouse
University of Virginia
Physics Dept.
Charlottesville, VA, USA
sgoadhouse@virginia.edu

***Abstract*—This document describes the setup and results of the GBTX 8B/10B encoder verification. The design was verified by embedding it within a working GBT emulator port to the ngCCM board and the miniCTR2 board, running in two Xilinx Virtex 5 FPGAs linked by their GTX SerDes over optical fiber.**

Keywords-component; 8B/10B encoder; GBTX; GBT; verification

I. Introduction

The GBTX ASIC is a communications device designed to transmit and receive data, control and clock signals over a 4.8 Gbps optical fiber link. An 8B/10B encoder was designed in triple-mode redundancy logic to be embedded within the GBTX design, which is still within the design phase. After the 8B/10B design was verified via simulation, it was embedded within a working port of the GBT−FPGA emulator so that it can be tested at full speed within an environment similar to its intended use.

II. Test Setup

A. Hardware

The verification hardware consisted of a prototype ngCCM (next generation Clock Control Module) and a prototype miniCTR2 as shown in Figure 1. SFP modules were inserted into the appropriate locations and connected with a 2 m LC to LC duplex 10Gb multimode OM3 fiber optic patch cable. The GBT emulator, which is VHDL RTL logic from the GBT−FPGA project that functions similar to the GBTX, was ported to the Xilinx XC5VFX70T-1 FPGA on the prototype miniCTR2 board and to the Xilinx XC5VFX30T-2 FPGA on the prototype ngCCM board.

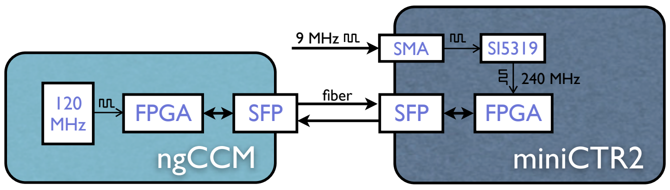

Figure 1. Block Diagram of the Verification Hardware

The primary difficultly in getting the two asynchronous boards to communicate reliably was design of the clocking structure. The structure that was ultimately used with much success has the miniCTR2 board generating a 240 MHz reference clock for its FPGA GTX SerDes from an external 9.0000 MHz clock originating from a waveform generator. The 9 MHz clock was then multiplied by the miniCTR2's SI5319 clock synthesizer chip to create the 240 MHz reference clock. On the ngCCM, an internal 120 MHz oscillator was used as the reference clock for the FPGA's GTX SerDes. The GTX on each platform was configured to generate the appropriate internal clocks from these reference clocks in order for the serial link to operate at 4.8 Gbps. The GTXs were also configured so that the receive, or deserializer, logic first recovered the receive clock from the optical link and then used the recovered clock to clock all of the deserializer logic. Therefore, the ngCCM serializer logic and the miniCTR2 deserializer logic were both synchronized to the ngCCM's 120 MHz reference clock while the miniCTR2 serializer logic and the ngCCM deserializer logic were both synchronized to the miniCTR2's generated 240 MHz reference clock. This means that the serializer and deserializer within the same FPGA were asynchronous, which does not impact the 8B/10B verification.

B. FPGA Serializer Logic

Once the GBT link was verified using the GBT emulator self-test, the RTL code was modified to include the triple modular redundancy 8B/10B encoder logic that was to be verified. The encoder was inserted as a multiplexed source for the serializer stream. By changing a single bit in a control register, the output of the 8B/10B encoder could be switched into the parallel data stream of the transmitting GTX. When the 8B/10B encoder was not switched in, the forward error-correcting GBT link logic was used. This made it possible to independently confirm the integrity of the communications link before enabling the 8B/10B encoder for testing. Two different test data sources were created and formed the input to the 8B/10B encoder via a switchable multiplexer. Figure 2 shows a block diagram of the serializer and how the triple modular redundancy (TMR) 8B/10B encoder was inserted in the data stream.

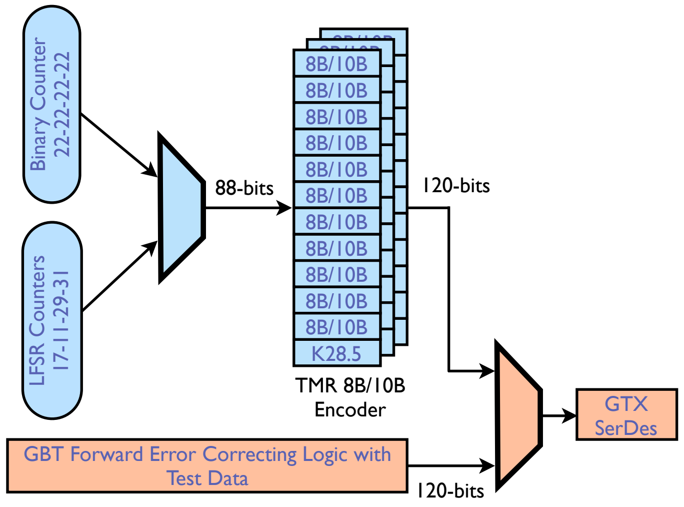
Binary Counter 22-22-22-22
LFSR Counters 17-11-29-31
88-bits
8B/10B
K28.5
TMR 8B/10B Encoder
120-bits
GBT Forward Error Correcting Logic with Test Data
120-bits
GTX SerDes

Figure 2. Block Diagram of the Serializer

The first source of test data was a 22-bit binary up counter that continually counted up and “rolled-over” to 0 when reaching the maximum 22-bit count. The 22-bit output of the counter was concatenated with itself 4 times to form the 88−bit test word. Each time the 22-bit counter “rolled-over” from (2^{22} − 1) to 0, an idle sequence would be inserted into the data stream. The number of idle words within the idle sequence was controlled by a 10-bit binary up counter which was incremented with each roll-over of the 22-bit counter. The idle sequence would always be the 10-bit idle count + 1 idle words. This 10-bit idle counter would also roll over from 1023 to 0. So the full test data sequence did not repeat until after ($2^{22} \bullet 2^{10}$) + sum(1 to 2^{10}) = 2^{32} + 524800 = 4295492096 words. With a 40 MHz clock, it would take just over 107 seconds to repeat. See Table 1 for a representative listing of this test word sequence.

The 88-bit word shown for the idle sequence in the table is its 8-bit representation. A control bit not shown in the table was used to denote that this was an idle control sequence and not just another 88-bit data word. So when the 8B/10B encoder encoded each 8-bits of the 88-bit idle word to 10-bits, the appropriate K28.5 control idle characters would be used. Also not shown in the table is the same K28.5 character that was used as an alignment character at the beginning of each encoded 88-bit data word. It was a function of the 8B/10B encoder to insert the K28.5 characters when appropriate.

The second source of test data came from four linear feedback shift register (LFSR) counters of different sizes concatenated together to form a pseudo−random bit sequence (PRBS). The LFSR counters used were 17−bit, 11−bit, 29−bit and 31−bit counters, in that order from left-most and most significant bit to right-most and least significant bit. These LFSR bit sizes are standard bit sizes often used in bit error rate testing[1]. The 17−bit LFSR used taps at bits 13 and 16 (msb), the 11−bit LFSR used taps at bits 8 and 10 (msb), the 29−bit LFSR used taps at bits 26 and 28 (msb) and the 31−bit LFSR used taps at bits 27 and 30 (msb). Additionally, the 29−bit and 31−bit LFSR counter outputs were inverted, per their standards. The 17−bit and 11−bit counters were started at all 1‘s and the inverted 29−bit and 31−bit counters were started at all 0‘s. All four counters continually counted and rolled over at their own pace with the largest, 31−bit, counter requiring 54 seconds before it repeated itself. In theory, by concatenating LFSR counters of different bit lengths, nearly every possible bit combination would be tested[2]. However, the full 88-bit test word sequence would need 7.7^{18} seconds, or 245 billion years, before it repeated, which is roughly 20 times longer than the universe is thought to exist. So although each LFSR counter tested nearly every bit combination within its bit size, the full 88-bit combinations could never be completely tested. Since the 88-bit word is encoded by eleven 8B/10B encoders in parallel and each single encoder was an instantiation of the exact same logic, every possible 8B/10B bit pattern was tested in most of the parallel 8B/10B encoders with a different 8B/10B bit pattern tested simultaneously in each concurrent encoder. A few counts of the PRBS test words are shown in Table 2 at the counts where the 11−bit, 17−bit, 29−bit and 31−bit LFSR counters “roll-over”. In fact, the 11−bit counter is shown rolling over twice.

C. FPGA Deserializer and Verification Logic

On the receiver, or deserializer, end of the communications, the FPGA GTX SerDes was configured so that the incoming 8B/10B encoded data stream was switched through the GTX’s built-in 8B/10B decoder. This evaluated the 8B/10B encoded data stream against a widely tested and used 8B/10B decoder. Logic was then added to re-align the data so that the synchronization character, K28.5, was always the most significant character followed by the 11 test data bytes, parallelizing the result back into 88-bit words. The received data was then compared against an expected value.

Since the 22-bit binary up counter and the LFSR counters are deterministic, the expected value for the current word is equivalent to the word received in the previous cycle fed through the same algorithm that created the current word. This prevents the need to manually synchronize the test data generator and the expected data verifier. Due to latencies within the system, the test data generator and the verifier will never be at the same test word at the exact same time. Having the verifier generate the expected value means that the verifier will only declare the received data to be good if the order of the test data is correct. Additionally, this method allows auto-recovery from transmissions errors.

If an acute bug causes the received data to have been erroneously encoded or erroneously transmitted, the received word will not match the expected value that is based on the penultimately received test word. The test word received in the next cycle will now be compared against an expected value based on the erroneous data. If this next word was sent error free, it will not match with the expected data based on the erroneous data and the error count will be incremented once more. However, this correct word will correctly generate the expected value for the following cycle and the system has resynchronized.

One danger in this method is if a chronic bug causes the data to be encoded incorrectly but is such that the order of the data at the received end is correct. The expected word and the actual received word will match in this case and the error may

[1] From Xilinx Application Note, XAPP 884, Table 3

[2] These LFSR counters skipped the bit sequence of all 0’s, or all 1’s for the inverted counters 29-bit and 31-bit, which is a by-product of the algorithm used to generate the LFSR count. For example, this meant that the 17-bit counter repeated after 2^{17} - 1 counts instead of the 2^{17} counts for a binary up counter.

be missed. By using both a binary up counter and multiple LFSR counters, this risk is greatly diminished. The 8B/10B encoding algorithm itself also diminishes this risk greatly. Not all 10−bit encoded characters are valid and if any of the invalid 10−bit characters are received, an error is flagged. Additionally, by using the idle sequences, the idle mechanism of the 8B/10B encoder is verified and the received data stream is monitored for an absolute expected value.

TABLE I. TEST DATA FROM 22-BIT BINARY UP COUNTER

Test Data Count	Test Data
	88-bit Word
1	0x0000000000000000000000
2	0x0000040000100000400001
3	0x0000080000200000800002
4	0x00000C0000300000C00003
⋮	⋮
4194301	0xFFFFF3FFFFCFFFFF3FFFFC
4194302	0xFFFFF7FFFFDFFFFF7FFFFD
4194303	0xFFFFFBFFFFEFFFFFBFFFFE
4194304	0xFFFFFFFFFFFFFFFFFFFFFF
4194305	0xBCBCBCBCBCBCBCBCBCBCBC
4194306	0x0000000000000000000000
4194307	0x0000040000100000400001
⋮	⋮
8388608	0xFFFFFBFFFFEFFFFFBFFFFE
8388609	0xFFFFFFFFFFFFFFFFFFFFFF
8388610	0xBCBCBCBCBCBCBCBCBCBCBC
8388611	0xBCBCBCBCBCBCBCBCBCBCBC
8388612	0x0000000000000000000000
8388613	0x0000040000100000400001
⋮	⋮
4295492092	0xBCBCBCBCBCBCBCBCBCBCBC
4295492093	0xBCBCBCBCBCBCBCBCBCBCBC
4295492094	0xBCBCBCBCBCBCBCBCBCBCBC
4295492095	0xBCBCBCBCBCBCBCBCBCBCBC
4295492096	0xBCBCBCBCBCBCBCBCBCBCBC
4295492097	0x0000000000000000000000
4295492098	0x0000040000100000400001
⋮	⋮
4299686399	0xFFFFFBFFFFEFFFFFBFFFFE
4299686400	0xFFFFFFFFFFFFFFFFFFFFFF
4299686401	0xBCBCBCBCBCBCBCBCBCBCBC
4299686402	0x0000000000000000000000
4299686403	0x0000040000100000400001
⋮	⋮
4303880704	0xFFFFFBFFFFEFFFFFBFFFFE
4303880705	0xFFFFFFFFFFFFFFFFFFFFFF
4303880706	0xBCBCBCBCBCBCBCBCBCBCBC
4303880707	0xBCBCBCBCBCBCBCBCBCBCBC
4303880708	0x0000000000000000000000
4303880709	0x0000040000100000400001
⋮	⋮

TABLE II. TEST DATA FROM THE LFSR COUNTERS

Test Data Count	Test Data
	88-bit Word
1	0xFFFFFFF000000000000000
2	0xFFFF7FE000000080000001
3	0xFFFE7FC000000180000003
⋮	⋮
2047	0x13253FF074BFF3C0000003
2048	0x264AFFF0E97FE780000006
2049	0x4C957FE1D2FFCF8000000D
⋮	⋮
4094	0x06223FFFE78241C000001E
4095	0x0C447FFFCF04838000003C
4096	0x18887FEF9E090780000079
⋮	⋮
131071	0x7FFF970984D7CB40003FFF
131072	0xFFFFAE1309AF9600007FFE
131073	0xFFFF5C26135F2C0000FFFD
⋮	⋮
536870911	0x0C443E280000003FFFFFC0
536870912	0x18887C500000007FFFFF80
536870913	0x3110F8A0000000FFFFFF01
⋮	⋮
2147483647	0xF020CD50000001C0000000
2147483648	0xE0411AB0000003800000000
2147483649	0xC082B570000007800000001
⋮	⋮

If an error was ever detected by the logic, certain status bits would be asserted within status registers to flag the errors. A single error would cause the appropriate bit to be set and could only be cleared by a user control bit. Thus the error bits would never be automatically cleared. Additionally, a counter was incremented for every error detected so that the rate of errors could be deduced even though the error bits were latched with the first error. One error bit was used to denote any data errors discovered by the 8B/10B decoder, including invalid 10−bit characters, as well as data mismatches between the expected and actual test words. Because there would always be initial errors until the deserializer properly aligned the received data stream with the K28.5 alignment characters, another bit denoted any of the above errors after the automatic alignment procedure completed and the incoming data was considered valid. A third status bit indicated when the received data stream was aligned and considered valid. Additionally, a counter was incremented on the receive end each time the length of the idle sequence was detected to go from 1024 idle words to 1 idle word. Since the PRBS test data did not include idle sequences, this counter only incremented when the binary up counter test data was selected.

The primary difficulty with the deserializer was designing and implementing the alignment logic. The FPGA's GTX SerDes handled the 8B/10B decoding and also handled aligning the bits into bytes through the use of the K28.5 alignment character. However, the data was received by the 8B/10B decoder as 32-bit words with the alignment character appearing in any of the four byte positions in one out of every three 32-bit words. This required a smart FIFO that would search for the alignment character from the 32-bit data stream from the decoder and then align the bytes into 88-bit words following each found alignment character. Once this FIFO was successfully implemented, the design of the remaining deserializer and verification logic followed quickly.

III. Test Results

After the logic to integrate and verify the 8B/10B encoder was implemented and tested within both the ngCCM and miniCTR2 prototype platforms, it was time to run a long term test and look for any errors resulting from encoding and transmitting data using the 8B/10B encoder. Since both platforms used similar logic integrated with the 8B/10B encoder that was under test and both platforms included a transmitter and receiver, one link was tested with the binary up counter test words and idle sequences and the other link was tested with the PRBS test word sequence. This allowed the encoder to be simultaneously verified while running on different platforms with different reference clocks and different test data. The steps used to verify the encoder are as follows.

1) *Established error free communications between the ngCCM and the miniCTR2:* The 2m fiber optic cable was connected between the two platforms using off-the-shelf SFP fiber modules that can handle 4.8 Gbps. Communications was enabled and the error bits and counts were checked to confirm that the two platforms were communicating without error.
2) *Enabled 8B/10B encoder on the ngCCM:* After switching the output of the 8B/10B encoder to the FPGA GTX serializer SerDes on the ngCCM, the error bits on the miniCTR2 were checked. Since the miniCTR2 platform was still expecting the standard GBT forward error correcting (FEC) encoded data, many errors were detected as expected. Once the miniCTR2 was switched to verify for 8B/10B encoded data, the errors ceased to occur. This confirms that the verification logic is properly detecting errors. The ngCCM serializer was configured to use the test words from the 22-bit binary up counter and idle sequences while the miniCTR2 was configured to expect such test data.
3) *Enabled 8B/10B encoder on the miniCTR2:* Enabled the 8B/10B encoder on the miniCTR2. The error bits within the ngCCM, which was still expecting FEC data, properly asserted. Once the miniCTR2 serializer and ngCCM deserializer were configured for 8B/10B data from the 22-bit binary up counter and idle sequences, the errors ceased as expected.
4) *Both platforms transmitted and received 8B/10B encoded test data until the binary up counter sequence repeated twice:* Based on the calculations presented earlier, this took just over 107*2, or 214, seconds for both platforms to receive the full test data sequence twice.
5) *Enabled PRBS test data generation on the miniCTR2 platform:* The test data generator on the miniCTR2 platform was next configured to generate the PRBS test data, which has no idle sequences. The verification logic on the ngCCM immediately asserted its error bits since it was still expecting the test data from the 22-bit binary counter. Once the ngCCM verification logic was configured to expect the PRBS test data, the errors ceased, as expected. This was another positive test of the verification logic.
6) *Unplug the fiber cable a few times:* Once both platforms were transmitting and receiving 8B/10B encoded data using both types of test data and without any reported errors, the fiber cable was unplugged. Error bits in the verification logic on both platforms immediately reported errors as expected. This was yet another test of the verification logic to confirm that it was indeed verifying the data and when the data ceased to change with each clock cycle due to the fiber being removed, errors were reported. This test was repeated a second time just to make sure. At this point, the verification logic was felt to be doing its job reporting data errors as they were detected. The fiber cable was reinserted and the errors were cleared on both platforms.
7) *Allow the test to run for over 6 hours:* The test was enabled with 8B/10B encoded data being transmitted from both platforms. For the ngCCM to miniCTR2 path, the test data came from the 22-bit binary up counter and idle sequence generator. For the miniCTR2 to ngCCM path, the test data came from the PRBS generator. No errors were reported once the automatic alignment procedure completed (ie. data was valid). All error bits were cleared. The platforms were left transmitting and verifying received data for about 6 hours and 20 minutes. The error bits and error counters were checked intermittantly during the length of the test. **No errors were reported over this time and there were no errors reported at the end of the test time. The 8B/10B encoder logic performed flawlessly during the test.**

   a) At the end of the test, the sequence repeat counter, based on the idle sequence count, reported a total of 212 (0xD4) repeated binary test data sequences.

   b) Therefore, 212 * 107.387 = 6.32 hours or approximately 6 hours and 19 min. This is the amount of time that the test was left running, so no idle sequences were erroneously skipped.

   c) With 88-bits, or 11 bytes, of data transmitted every 25 ns during the nearly 6 hours and 20 minutes of test time, over 10 terabytes of data were transmitted and received by both platforms.

   d) For the miniCTR2 to ngCCM data path, which used the PRBS test data, no 88-bit test word ever repeated during the test time.
8) *As a final sanity check, use ChipScope to check the received data stream:* Since the forward error correcting (FEC) logic coexisted with the 8B/10B logic, there is a slim chance that an internal bug might have caused the FEC encoded data to be sent and received during the test and not 8B/10B encoded data. As a final sanity check, the ChipScope logic analyzer tool was used to look inside of the running FPGA on the miniCTR2 and a small bit of the received data was recorded in order to confirm that the 8B/10B encoded data was indeed being tested. The data captured by ChipScope was verified to be coming from the output of the built-in 8B/10B decoder and the data was confirmed to be 8B/10B encoded data since it was decoded without error.

   a) Additionally, if the data stream was erroneously not coming from the 8B/10B encoder, the idle sequence counter would have never seen an idle sequence and its count would have remained 0.