XAPP1122 (v1.1) November 10, 2008

Parameterizable 8b/10b Encoder

Author: Paula Vo

Summary

This application note describes a parameterizable 8b/10b Encoder, and is accompanied by a reference design that replaces the 8b/10b Encoder core previously delivered through the CORE Generator software. The implemented 8b/10b coding scheme is an industry standard, DC-balanced, byte-oriented transmission code ideally suited for high-speed local area networks and serial data links.

For all new FPGA designs targeting Virtex®-5, Virtex-4, Virtex-II, Virtex-II Pro, Spartan®-3, Spartan-3E, Spartan-3A, Spartan-3A DSP FPGAs, and newer architectures, use the 8b/10b Encoder reference design. All the features and interfaces included in the reference design are backward compatible with the LogiCORE IP 8b/10b Encoder v5.0 core. In addition, because the reference design is provided in plain text VHDL format, users have full visibility into the implementation of the function and can easily debug and modify the code.

Introduction

The 8b/10b transmission code specifies the encoding of an 8-bit byte (256 unique data characters) and an additional 12 special characters into a 10-bit symbol; thus the 8b/10b designation. It also specifies the decoding of the 10-bit symbol back into an 8-bit word. The 8b/10b code is a well-established, industry standard first proposed by A.X. Widmer [1] [2], and has been used in the physical layer (PHY) of a number of current and emerging standards, including Fibre Channel, Gigabit Ethernet, and Rapid IO.

The transmission code is DC-balanced allowing receivers to function at lower signal-to-noise ratios, which is specifically beneficial to the active gain, threshold setting, and equalization of optical receivers. The code has a limited run length of no more than five consecutive ones or zeros, and a guaranteed transition density, which permits clock recovery from the data stream. The special characters are useful as packet delimiters. A subset of them, referred to as commas, are unique in that their bit pattern never occurs in a string of data symbols and hence can be used to determine symbol boundaries at the receiving end. Additional rules embedded in the code design allow the receiver to detect most transmission errors.

Features of the 8b/10b Encoder reference design include:

- Encoding of 8-bit bytes and an accompanying command bit KIN into 10-bit symbols
- Encoding of 268 unique transmitted characters: 256 data characters (KIN=0) and 12 special characters (KIN=1)
- Encoder tracks running disparity to ensure that the disparity sequence of the transmitted symbols is valid
- Choice of a LUT-based implementation, which uses FPGA slices, or a block-memory based implementation that uses a dedicated on-chip block memory
- Optional control inputs: Clock Enable (CE), command input (KIN), Force Code (FORCE_CODE), Force Disparity (FORCE_DISP) and Disparity Input (DISP_IN)
- Optional status outputs: Running Disparity (DISP_OUT), command error (KERR), and New Data (ND)

Interface

The 8b/10b Encoder reference design is synchronous. All inputs, Data Input (DIN), Command Input (KIN), Force Code (FORCE_CODE), Force Disparity (FORCE_DISP), and Disparity Input (DISP_IN), are sampled by the rising edge of the clock (CLK) input when Clock Enable (CE) is active. The outputs, Data Output (DOUT), Running Disparity Output (DISP_OUT), Command Error (KERR), and New Data (ND), are registered at the rising edge of the clock (CLK) input when Clock Enable (CE) is active.

The parameterizable Encoder supports a dual-encoder configuration of two independent encoders. The block memory-based Encoder configuration may implement the two encoders within the same dual-port block RAM with minimal additional resources. A set of optional B ports are available for the second encoder. Figure 1 and Figure 2 display the input and output ports for the single and dual Encoder, respectively.

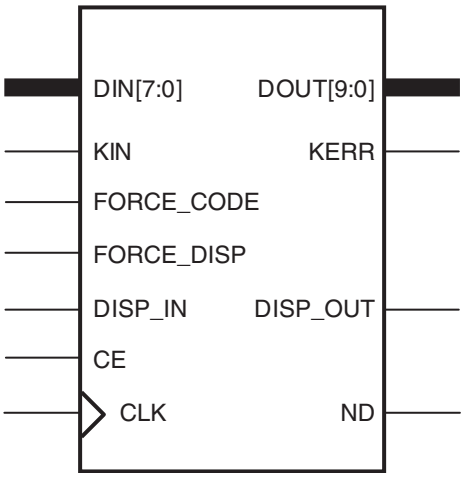

Figure 1: **8b/10b Encoder Schematic**

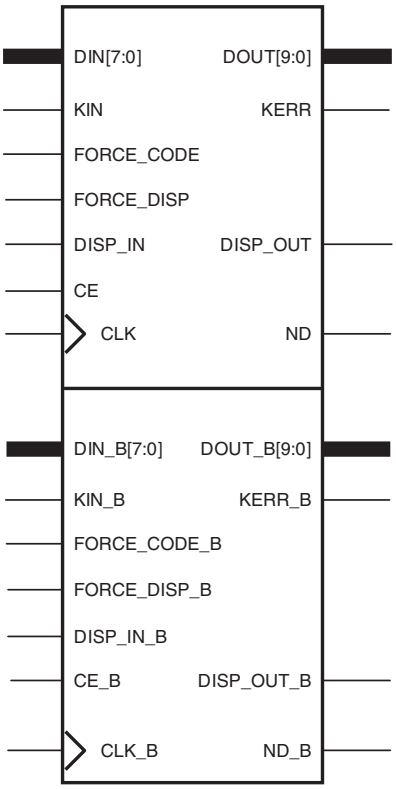

Figure 2: **8b/10b Dual Encoder Schematic Symbol**

Encoder Input and Output Ports

Table 1 describes the 8b/10b input and output ports. For detailed information about each port, see "Pinout," page 4.

Table 1: **Encoder Input and Output Ports**

I/O Pin Name	Direction Port Status	Description
CLK (CLK_B)	Input	**Clock**. All Encoder inputs are sampled and all outputs are synchronous to the rising edge of the CLK input. CLK_B is the clock input for the optional B Encoder.
CE (CE_B) optional	Input	**Clock Enable**. Gates the clock input if the optional CE input port is present. The CE input must be active (logic 1) or the Encoder will not change state in response to a CLK input. CE_B is the clock enable input for the optional B Encoder.
DIN[7:0] (DIN_B[7:0])	Input	**Data Input**. DIN is the input byte to be encoded. DIN_B is the input byte for the B Encoder.
KIN (KIN_B)	Input	**Command Input**. The KIN input controls how DIN is encoded. • if KIN = 0, DIN is encoded as data • if KIN = 1, DIN, if defined, is encoded as a special character See Table 10 for more information.
FORCE_DISP (FORCE_DISP_B) (optional)	Input	**Force Disparity**. FORCE_DISP, when active (logic 1), overrides the current running disparity with the external DISP_IN input. FORCE_DISP_B controls the disparity function for the optional B Encoder.
DISP_IN (DISP_IN_B) (optional)	Input	**Disparity Input**. DISP_IN sets the running disparity for the current input to be encoded. • If DISP_IN = 0, a negative running disparity is set for the current input • If DISP_IN = 1, a positive running disparity is set for the current input **Note**: DISP_IN has no effect if FORCE_DISP is inactive; the Encoder's internal running disparity is used to encode the input data. DISP_IN_B has the same function for the optional B Encoder.
FORCE_CODE (FORCE_CODE_B) (optional)	Input	**Force Code**. FORCE_CODE drives the Encoder into a pre-defined initialization state. When FORCE_CODE is active (logic 1), all inputs (with the exception of CLK) are *don't care* bits. FORCE_CODE_B has the same function for the optional B Encoder.
DOUT[9:0] (DOUT_B[9:0])	Output	**Data Output**. DOUT is the encoded 10-bit symbol. DOUT_B is the encoded 10-bit symbol for the B Encoder.

Table 1: **Encoder Input and Output Ports *(Cont'd)***

I/O Pin Name	Direction Port Status	Description
DISP_OUT (DISP_OUT_B) (optional)	Output	**Disparity Output**. DISP_OUT tracks the Encoder's running disparity. • If DISP_OUT = 0, the next symbol is encoded with a negative starting running disparity. • If DISP_OUT = 1, the next symbol is encoded with a positive starting running disparity DISP_OUT_B performs the same function for the optional B Encoder.
KERR (KERR_B) (optional)	Output	**Command Error**. KERR is used for debugging. This output becomes active (logic 1) if KIN is active and DIN does not map to a defined special character (see Table 10). KERR_B performs the same function for the optional B Encoder.
ND (ND_B) (optional)	Output	**New Data**. ND is active (logic 1) whenever the CE pin was high on the prior clock cycle, indicating that new data resides on the output pins.

Pinout

Clock

The Encoder is fully synchronous to its appropriate clock; CLK, or CLK_B for the B Encoder. All Encoder input ports have their setup time referenced to the rising edge of the CLK (or CLK_B) input. Clock inputs are rising edge active by default; to make the Encoder respond to the falling edge of a system clock, insert an inverter between the system clock and the Encoder's CLK input.

Clock Enable

Clock Enable (CE, or CE_B for the B Encoder), an optional input, can be used to gate the clock input to the Encoder. If the Encoder has a CE port and the CE input is inactive (logic 0), transitions on the clock port have no effect. If CE is active (logic 1) or is not present, the inputs are read and outputs updated on every rising edge of the CLK.

Data Input

The Data Input bus (DIN[7:0], or DIN_B[7:0] for the B Encoder) provides the data bytes to be encoded into code symbols. When KIN is inactive, a data symbol is produced; if KIN is active, a special character is encoded, provided the DIN input maps to a defined special character. Each pin of the bus should be driven to a valid logic level, for example, if data words of fewer than 8 bits are to be encoded, the unused DIN bits must still be driven to a valid logic level.

Command Input

The Command Input bus (KIN, or KIN_B for the B Encoder) is used to differentiate between the encoding of data bytes and special characters. If KIN is inactive (logic 0), the DIN bus inputs are encoded into a data symbol representing one of the 256 possible permutations of the data byte (DIN). When KIN is active (logic 1), a special character, determined by the DIN input, is encoded (see Table 10). Note that only 12 special characters are defined and for this reason many combinations of KIN=1 and DIN are not valid. While the behavior of the Encoder with these invalid inputs is deterministic, the only guarantee is that the KERR output will become active when an undefined special character is requested.

Force Disparity

Force Disparity (FORCE_DISP, or FORCE_DISP_B for the B Encoder), an optional input, when present can be used to override the Encoder's internal running disparity. When active (logic 1), the DIN/KIN input is encoded based on a running disparity set by the DISP_IN port. This input can be used to force data packets to start with a given disparity or, alternatively, can be tied to logic 1 when chaining Encoders together.

Disparity Input

Disparity Input (DISP_IN, or DISP_IN_B for the B Encoder), an optional input, specifies the starting disparity for symbol encoding when FORCE_DISP is present and active (logic 1). To encode DIN with a positive running disparity, DISP_IN should be driven high (logic 1). To encode DIN with a negative running disparity, DISP_IN should be driven low (logic 0). DISP_IN has no effect on the symbol encoding when FORCE_DISP is inactive (logic 0).

Force Code

Force Code (FORCE_CODE, or FORCE_CODE_B for the B Encoder), an optional input, when present, can be used to force the Encoder to output a pre-selected data symbol. If active (logic 1), the Encoder outputs an encoded symbol representing the symbol and running disparity specified by the parameters C_FORCE_CODE_VAL and C_FORCE_CODE_DISP (see Table 5 and Table 6). Users should note that if active for more than one clock period, the consecutive symbols are generated with the same running disparity and for this reason have the potential (if the disparity of the generated symbol is not zero) to violate the coding scheme's disparity rules.

Data Output Bus

When CE is inactive, the Data Output bus (DOUT, or DOUT_B for the B Encoder) holds the previously encoded symbol. When CE is active, DOUT updates at the rising edge of CLK. The output symbol is the result of encoding DIN, KIN, and the starting disparity, where the starting disparity is either the internal running disparity or DISP_IN, depending on whether FORCE_DISP is active.

Disparity Output

Disparity Output (DISP_OUT, or DISP_OUT_B for the B Encoder), an optional output, is the internal running disparity of the Encoder. This output allows the user to see the internal state of the running disparity within the core. If DISP_OUT = 0, the new internal running disparity is negative, indicating that the next value encoded must be encoded with either a neutral or positive code disparity to comply with the disparity rules. Similarly, if DISP_OUT = 1, the new internal running disparity is positive, which forces the next encoded symbol to be encoded with a neutral or negative code disparity. If multiple Encoders are chained together, the DISP_OUT signal should be connected to the DISP_IN pin of the next Encoder (the Encoder that produces the next symbol to be serialized).Table 2 identifies the disparity rules.

Table 2: **Disparity Rules**

Current Running Disparity	Code Disparity	New Running Disparity
-	0	-
-	+	+
+	0	+
+	-	_
-	-	error/invalid
+	+	error/invalid

Command Error

Command Error (KERR, or KERR_B for the B Encoder), an optional output, indicates that KIN was active on the immediate prior clock cycle, but the inputs on the DIN[7:0] bus did not correspond to a valid special character. This port is primarily provided as a debugging aid, as inputs of this type violate the coding scheme and for this reason should not occur in normal-use scenarios.

New Data

New Data (ND, or ND_B for the B Encoder), an optional output, indicates that all Encoder outputs have been updated on the most recent clock, indicating to downstream logic that a new encoded symbol is available for transmission. ND requires that the Encoder also have a CE input, and is simply a registered version of CE. ND is not active on any clock cycle following an active FORCE_CODE input.

Hardware Implementation

The 8b/10b Encoder reference design is parameterizable and may be configured to implement a look-up table in LUTs or in a block RAM. The RTL is described entirely in VHDL using inference. This inference-based approach has the advantage of portability across existing and future device families. The disadvantage is susceptibility to synthesis tool inference limitations.

The 8b/10b Encoder essentially performs a table lookup with a 10-bit input and a 12-bit output. The 10-bit input is comprised of the 8-bit data (DIN), a command bit (KIN), and the current disparity (the external DISP_IN input or the internal, previous running disparity DISP_OUT). The 12-bit output is comprised of the 10-bit encoded symbol (DOUT), a command error bit (KERR) and the resulting disparity (DISP_OUT). If the command bit (KIN) is active and the input data byte does not correspond to one of the 12 special characters, then the command error (KERR) output is activated.

LUT-based Encoder

The LUT-based Encoder implementation uses slice logic to translate the 8b/10b encoding tables into VHDL case statements (see Tables IV-V in [2]). These case statements implement the partitioned 5-bit to 6-bit encoding and 3-bit to 4-bit encoding steps.

Block RAM-based Encoder

The block RAM-based Encoder implementation is a true look-up table using a block RAM configured as a ROM. The table is mapped into a 1024 x 12-bit ROM, as illustrated in Figure 3. The 1024-entry table describes the set of encoder outputs (DOUT, KERR, and DISP_OUT) for all possible combinations of DIN, KIN, and the current disparity (either DISP_IN or the previous DISP_OUT).

The ROM contents are generated from exhaustive simulation of the LUT-based Encoder and capturing the results to an ASCII .mif file. This .mif file is used to initialize the ROM.

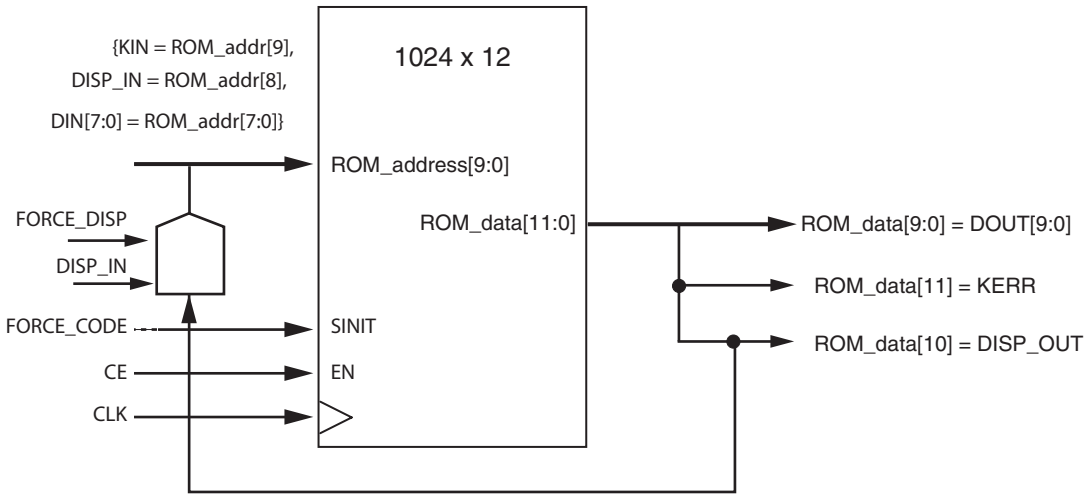

Figure 3: **Encoder Implemented as a ROM Lookup Table**

Possible Enhancements

Define an Error Code

When encoding an 8-bit data byte, the 8b/10b Encoder uses a command input (KIN) to determine whether the input data is a data character (KIN = 0) or a special character (KIN = 1). The 8b/10b transmission code identifies 256 valid data characters and 12 special characters. With only 12 defined special characters, this leaves 244 invalid input bytes when KIN = 1.

The 8b/10b Encoder flags these invalid special characters with the KERR output, but continues to encode the data as DC-balanced codes of the appropriate disparity. The encoded symbol is then transmitted to the Decoder without indicating whether KERR was activated. If the resulting code appears to be an invalid encoded symbol, the downstream 8b/10b Decoder flags the error with its CODE_ERR output. If the resulting code appears to be a valid encoded symbol, the downstream 8b/10b Decoder will assume that the symbol is valid and will not flag a code error.

To properly communicate the code error to the 8b/10b Decoder, both the Encoder and Decoder can be modified to recognize a defined, DC-balanced error code. If the Encoder is presented with an invalid special character, it could flag the error with its KERR output and transmit the defined error code, with appropriate disparity, to the Decoder. The Decoder could interpret this specific code and flag a code error. This modification to the Encoder would entail changing the LUT-based Encoder case statements and subsequently regenerating an updated .mif file, if the block RAM-based Encoder is the targeted Encoder implementation.

Expand the Datapath Width

The 8b/10b Encoder datapath width may be doubled from an 8b/10b to a 16b/20b through the use of multiple Encoders and minimal overhead logic. Two possible configurations are shown in Figure 4 and Figure 7. The two-encoder configuration of Figure 4 requires two clocks and lower resource utilization. The three-encoder configuration of Figure 7 requires one clock and higher resource utilization.

The two-encoder design of Figure 4 may be implemented using the dual Encoder configuration of the 8b/10b reference design. For compliance with the 8b/10b disparity rules, the LSB sub-block is encoded before the MSB sub-block, and the Disparity Output of the LSB sub-block drives the Disparity Input to the MSB sub-block. To ensure that the LSB sub-block is encoded prior to the MSB sub-block, the input data must arrive just prior to the active edge of the LSB Encoder. This can be accomplished by registering the input data on the opposite edge of the LSB Encoder clock as indicated in the diagram. Each Encoder's Disparity Output port will be tied to the Disparity Input port of the opposite Encoder. The two Encoders may operate on opposite edges of the same clock, as shown in the timing diagram of Figure 5, which would result in a single cycle of latency for the encoding operation. If additional latency may be tolerated in the design (two cycles per encode), the clocking may be simplified by operating the two Encoders on the same clock, but with their Clock Enables toggling on alternate cycles (Figure 6).

The three-encoder design of Figure 7 can be implemented using a single Encoder combined with a dual Encoder. In this architecture, both the MSB and LSB sub-blocks are encoded on the same clock edge. This may be accomplished by duplicating the MSB Encoder logic (MSB Encoder A and MSB Encoder B) and precalculating the results for the case where the Disparity Input = 1 and the case where the Disparity Input = 0. Once the running disparity of the LSB Encoder is known, it will be used to select the correct results from the precalculated set. This select logic is combinational, and may cause glitching on the Data Output (DOUT), command error (KERR), and Disparity Output (DISP_OUT) of the MSB Encoder as shown in Figure 8. For this reason, it is recommended that all outputs of this Encoder architecture should be registered. Select logic is not required for the MSB Encoder's New Data (ND) signal, because the signal is identical between the two instantiated MSB Encoders.

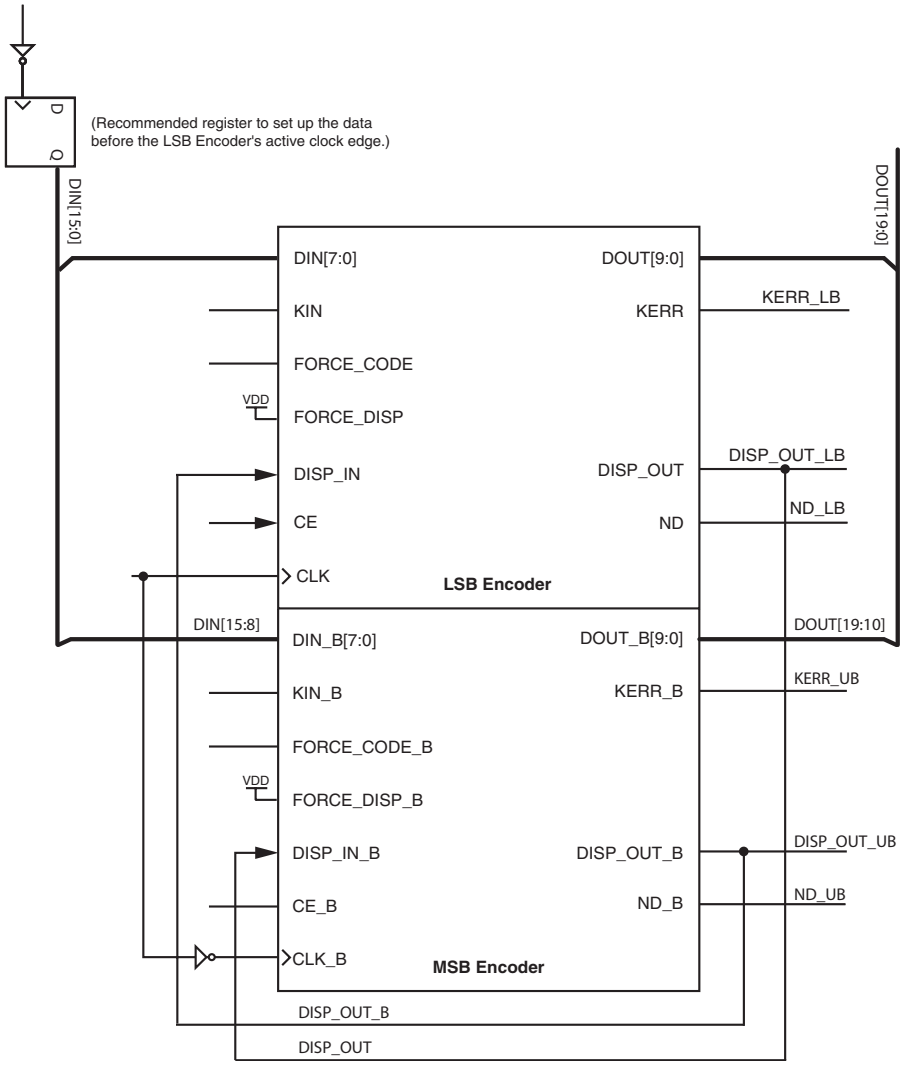

Figure 4: **Two-encoder Configuration**

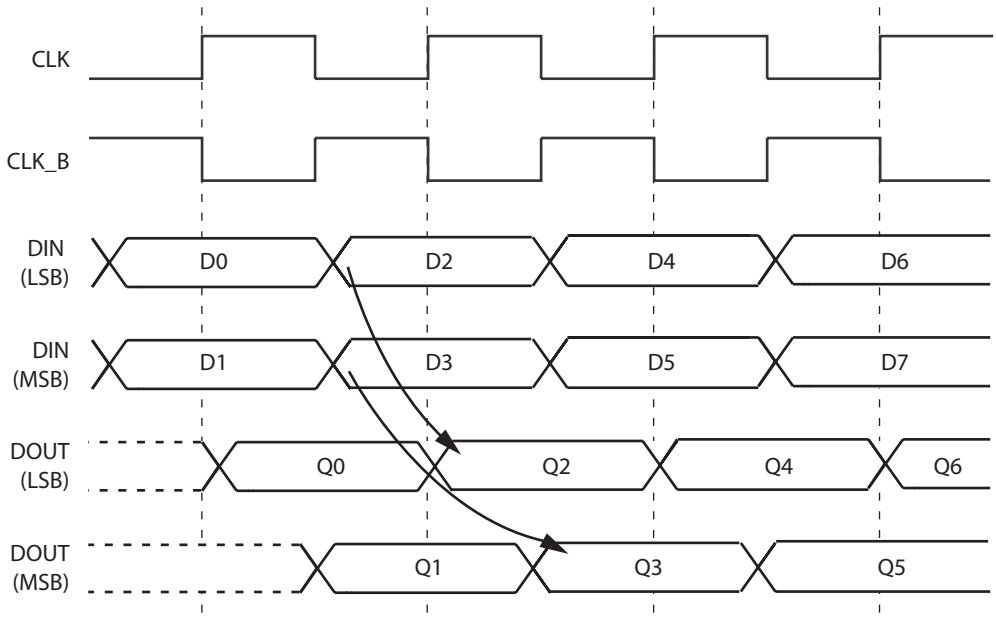

Figure 5: **Using Rising and Falling Clock Edges**

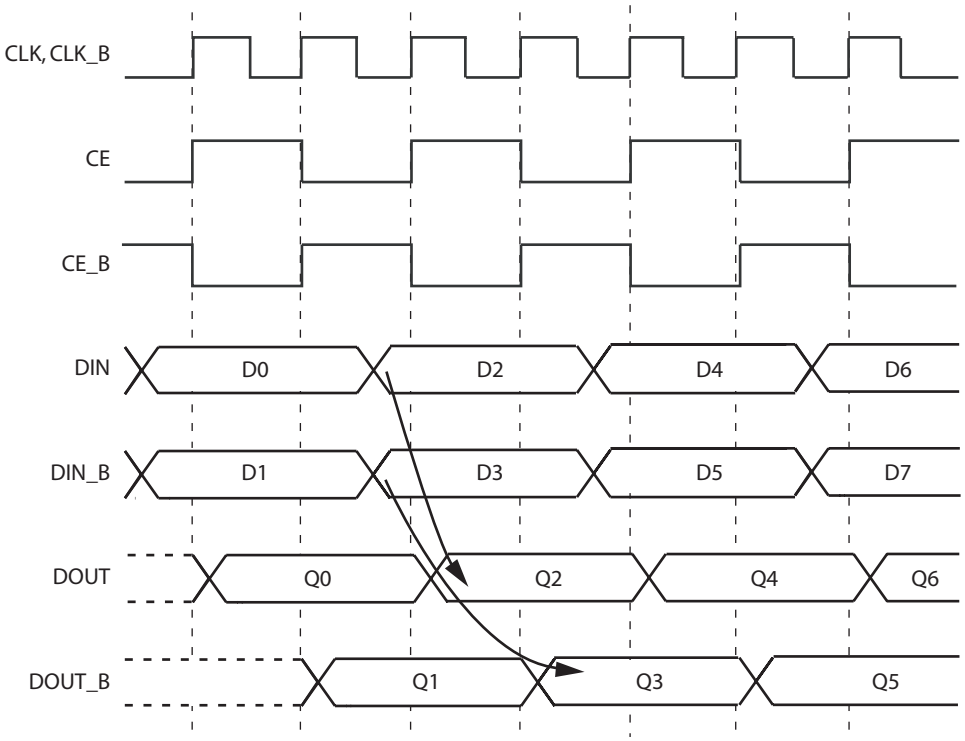

Figure 6: **Using Two Cycles to Encode**

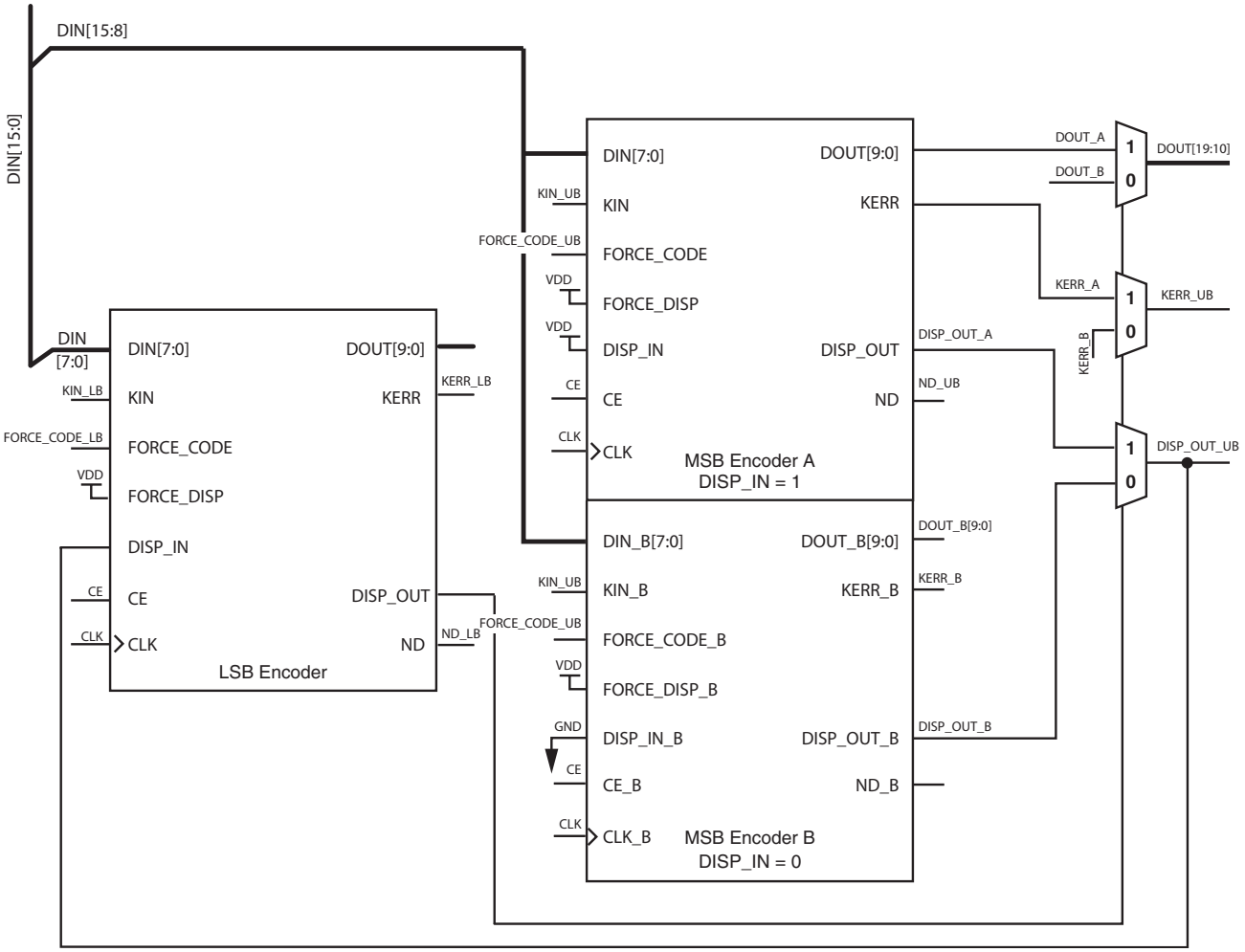

Figure 7: **Three-encoder Configuration**

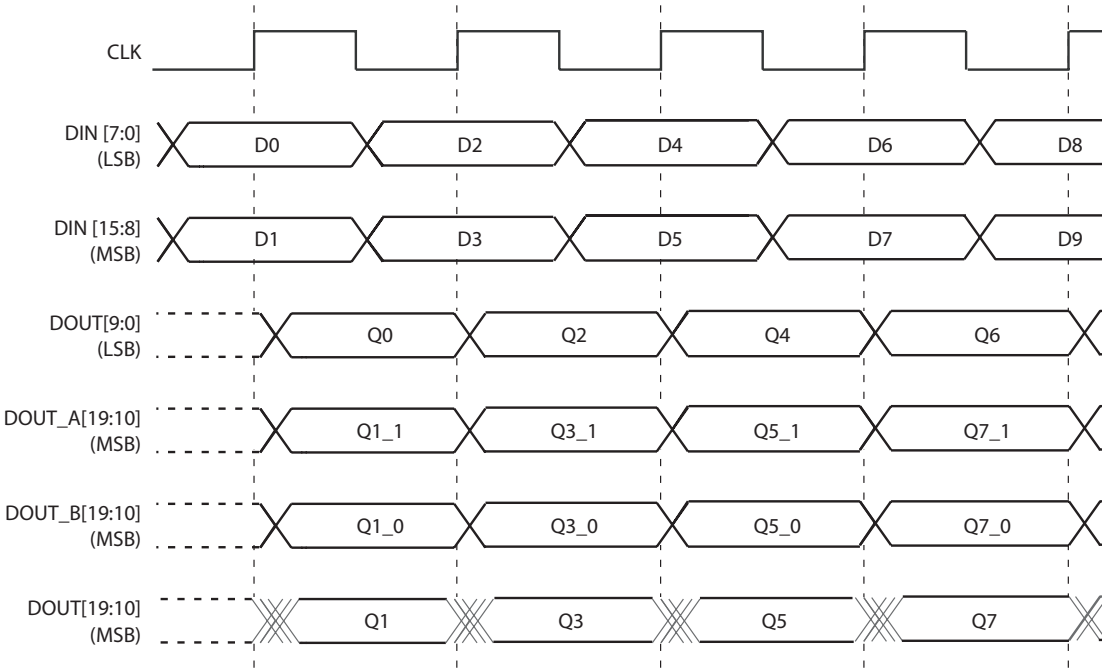

Figure 8: **Using One Clock, One Cycle to Encode**

Reference Design

The 8b/10b Encoder reference design that accompanies this application note contains VHDL source code and Perl scripts to customize the design, synthesize it in XST, and implement it through Ngdbuild, Map, and PAR. An additional Perl script is provided to generate the block RAM initialization .mif file, if desired. Table 3 and Table 4 describe the reference design source files, and the scripts, project files, and documentation files for the Encoder, respectively.

Table 3: **8b/10b Encoder Design Files**

Filename	Description
encode_8b10b_wrapper.vhd	VHDL RTL for the top-level core wrapper file.
encode_8b10b_top.vhd	VHDL RTL for the core wrapper file. Maps the set of 12 simplified generics in the top-level core wrapper file to the full set of 19 generics in the top-level core file.
encode_8b10b_pkg.vhd	VHDL package file of constants and functions.
encode_8b10b_rtl.vhd	VHDL RTL for the top-level core file.
encode_8b10b_lut.vhd	VHDL RTL for the LUT-based Encoder.
encode_8b10b_lut_base.vhd	VHDL RTL for a single LUT-based Encoder.
encode_8b10b_bram.vhd	VHDL RTL for the block RAM-based Encoder.
enc.mif	ASCII text file containing the 1024 x 12-bit table to initialize the block RAM-based Encoder.

Table 4: **8b/10b Encoder Documentation and Script Files**

Filename	Description
README_XAPP1122.txt	Describes the reference design files and script files, and includes instructions to execute the provided scripts.
CustomizeWrapper.pl	Interactive Perl script that facilitates configuration of the 12 simplified generics in the top-level core wrapper file.
WrapperTemplate.txt	Template file used by the CustomizeWrapper.pl script to generate the top-level core wrapper file.
RunXST.pl	Perl script that synthesizes the Encoder source files in XST.
vhdl_xst.scr	XST script file containing XST synthesis options, including the target part.
vhdl_xst.prj	XST project file containing the relative paths to the VHDL files to be synthesized.
Implement.pl	Perl script that runs Ngdbuild, Map, and PAR on the synthesized netlist.
MakeMIF.pl	Perl script that simulates mifgen_enc.vhd and the LUT-based Encoder RTL in ModelSim to generate a new enc.mif initialization file if the Encoder behavior (RTL) is modified.
mifgen_enc.vhd	Provides stimulus to the LUT-based encoder exercising all possible code points and captures the resulting table to the enc.mif file.
MakeMIF_mti.do	ModelSim .do script to generate the enc.mif file.

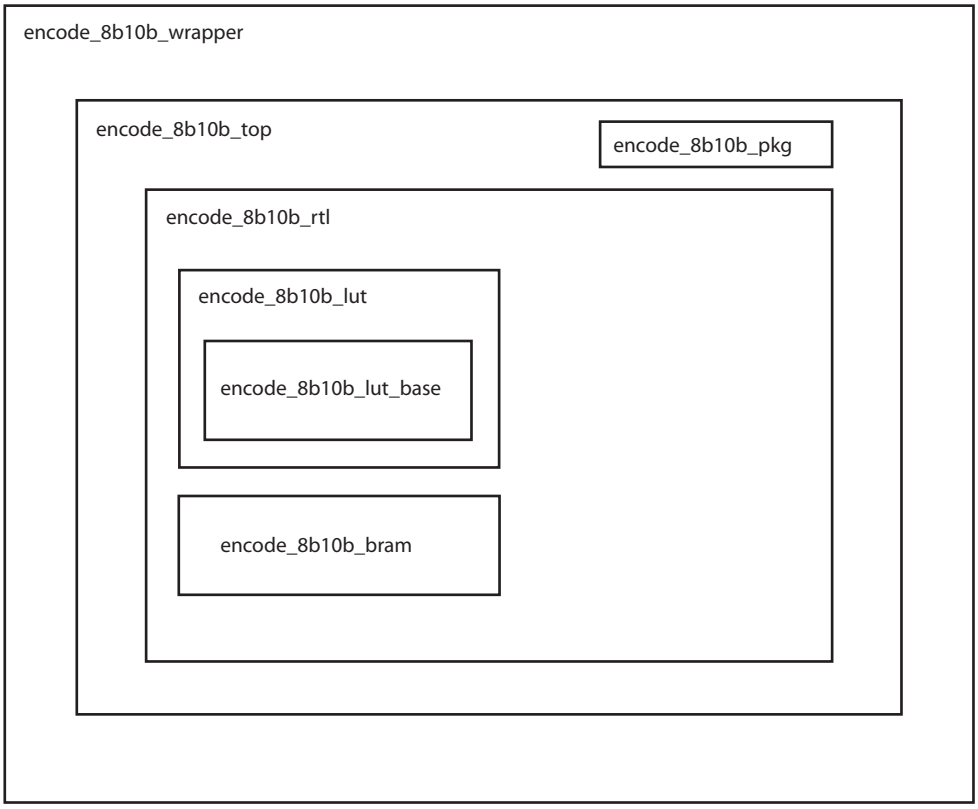

Figure 9: **RTL Hierarchy**

Compilation Parameters

The 8b/10b Encoder reference design is parameterizable with a set of 12 simplified generics. An interactive, command-line Perl script, CustomizeWrapper.pl, is provided to facilitate configuration of these parameters. This script creates a top-level core wrapper file with the desired configuration. For additional control over the Encoder implementation, a user may manually edit the 19 complete generics in the top-level core file and implement the design without the two wrapper files. Table 5 describes the full set of generics present in the top-level core file and the simplified set present in the top-level core wrapper file. In the simplified set of generics, if the dual-encoder option is selected (C_HAS_BPORTS = 1), the second encoder is configured with the same optional ports as the first encoder. However, their initialization values may differ.

Table 5: **RTL Parameters**

Generics	Simplified Generics	Description
C_ENCODE_TYPE	C_ENCODE_TYPE	Implementation: 0=LUT-based, 1=Block RAM-based
C_ELABORATION_DIR	none	Hard-coded path to enc.mif file location
C_FORCE_CODE_DISP	C_FORCE_CODE_DISP	Force code disparity: 0 neg, 1 pos
C_FORCE_CODE_DISP_B	C_FORCE_CODE_DISP_B	Force code disparity port B: 0 neg, 1 pos
C_FORCE_CODE_VAL	C_FORCE_CODE_VAL	Force code value (10 bits)
C_FORCE_CODE_VAL_B	C_FORCE_CODE_VAL_B	Force code value B (10 bits)

Table 5: **RTL Parameters *(Cont'd)***

Generics	Simplified Generics	Description
C_HAS_BPORTS	C_HAS_BPORTS	1 indicates second encoder should be generated
C_HAS_CE	C_HAS_CE	1 indicates CE(_B) port is present
C_HAS_CE_B		
C_HAS_DISP_OUT	C_HAS_DISP_OUT	1 indicates DISP_OUT(_B) port is present
C_HAS_DISP_OUT_B		
C_HAS_DISP_IN	C_HAS_DISP_IN	1 indicates FORCE_DISP(_B) and DISP_IN(_B) ports are present
C_HAS_DISP_IN_B		
C_HAS_FORCE_CODE	C_HAS_FORCE_CODE	1 indicates FORCE_CODE(_B) port is present
C_HAS_FORCE_CODE_B		
C_HAS_KERR	C_HAS_KERR	1 indicates KERR(_B) port is present
C_HAS_KERR_B		
C_HAS_ND	C_HAS_ND	1 indicates ND(_B) port is present
C_HAS_ND_B		

Table 6: **FORCE_CODE Values**

Symbol Choice	C_FORCE_CODE_VAL (j...a)	C_FORCE_CODE_DISP
D.10.2 (pos)	1010 101010	1
D.10.2 (neg)	1010 101010	0
D.21.5 (pos)	0101 010101	1
D.21.5 (neg)	0101 010101	0
K.28.5 (neg)	0101 111100	0

Supported Design Tools

- Xilinx ISE® 10.1 (including XST 10.1 and xilperl)
- ModelSim v6.3c

Resource Utilization and Performance

Resource utilization of the 8b/10b Encoder varies depending on the specific configuration implemented. The implementation may be Block RAM-based or LUT-based, and may have a variety of optional input and output ports. The following tables show the resource utilization and performance for the 8b/10b Encoder for four basic configurations. The configurations are:

- Block RAM-based with all optional ports present
- Block RAM-based with no optional ports present
- LUT-based with all optional ports present
- LUT-based with no optional ports present

Table 7: **Resource Utilization**

	Configuration		FF	LUT	Block RAM
Spartan-3A xc3s50a tq144	block RAM-based	ALL	2	4	2
		NONE	0	0	1
	LUT-based	ALL	28	250	0
		NONE	13	78	0
Virtex-5 xc5vlx20t ff323	block RAM-based	ALL	2	4	1
		NONE	0	0	1
	LUT-based	ALL	26	104	0
		NONE	11	29	0

Table 8: **Performance (MHz)**

		Spartan-3A		Virtex-5	
		xc3s50a-tq144		xc5vlx20t-ff323	
		-4	-5	-1	-2
block RAM-based	ALL	200	240	280	320
	NONE	270	320	320	370
LUT-based	ALL	130	150	320	390
	NONE	130	160	380	450

Appendix

Disparity

A number of terms in this application note are taken from the original *IBM Journal* articles [1] [2], and a thorough understanding of the concepts and terminology, most importantly the concept of *disparity*, is critical to the successful application of an 8b/10b Encoder.

The disparity of any block of data is defined as the difference between the number of 1s and 0s in the block. Positive and negative refer to an excess of 1s over 0s, or 0s over 1s, respectively. Each encoded symbol can be considered to be a block. The code scheme guarantees that an encoded symbol's disparity is always either 0 (five ones, five zeros), +2 (six ones four zeros), or -2 (four ones, six zeros). Some byte inputs have more than one potential symbol encoding with the encoded symbol pattern. This is determined by the *running disparity*, which is simply a record of disparity for the aggregate of all the previously encoded symbols. For packet-based networking applications, the running disparity is typically tracked from the start of a packet.

The code scheme stipulates that the running disparity at the end of any symbol (block) is always +1 or -1. To ensure that this rule is maintained, the Encoder will track the current running disparity. If the currently encoded byte produces a symbol of zero disparity, the running disparity remains unchanged. When the input byte produces a nonzero disparity symbol, the Encoder will encode the data such that the running disparity is swapped, for example, [+1 + (-2) = -1] or [-1 +(+2) = +1]. See Table 9 for an example of the two possible encoded symbol patterns for the D31.1 data symbol.

The code scheme actually partitions the input byte into 5-bit and 3-bit sub-blocks, which in turn are encoded into 6- and 4-bit blocks respectively. The original nomenclature defines the symbols in terms of these sub-blocks. The five input bits are defined as A, B, C, D and E (A is LSB) and the 3-bit block is F, G, and H (F is LSB). A prefix of D or K in the symbol name is used to distinguish between data and special characters respectively. For example, D31.1 is a data symbol with all ones on the 5-bit block (11111) and a single one as the LSB of the 3-bit block (100) (see Table 9). Note that the 5-bit sub-block precedes the 3-bit sub-block and the ordering (LSB to MSB) is ABCDE_FGH. The encoded sub-blocks are described with lowercase letters a, b, c, d, e, i and f, g, h, j respectively. The digital weighting of the individual bits of a data byte to be encoded is somewhat arbitrary. However, the limited number of special characters (K symbols) makes the bit ordering for special character generation critical. See Table 10 for the appropriate state of the DIN input when generating special characters.

A number of details must be adhered to when using the encoder in a serial link—if these guidelines are not followed, the implementation of the link will not match the original function of the code scheme.

- When serializing encoded symbols, DOUT[0] should be transmitted first, DOUT[9] transmitted last. This will produce a serial stream of abcdei_fghj as stipulated in the original paper.
- The Encoder uses the same convention as the Fibre Channel specification in which DIN[0] is the least significant bit of the input byte. This doesn't have any implications for data words, but care must be taken when generating special characters. See Table 10 when determining correct input combinations for special character generation.
- If more than one encoder is used to encode larger words into multiple symbols, their disparity inputs and outputs must be chained together. The RUN_DISP output of the first Encoder drives the DISP_IN input of the second Encoder with the final Encoder's RUN_DISP output connected back to the DISP_IN of the first Encoder.

Table 9: **Example Encoding of D31.1 for Both Running Disparity Cases**

Data Byte Name	DIN[7:0]								RD (prior)	DOUT[9:0]									
	7	6	5	4	3	2	1	0		9	8	7	6	5	4	3	2	1	0
	H	G	F	E	D	C	B	A		j	h	g	f	i	e	d	c	b	a
D31.1	0	0	1	1	1	1	1	1	+1	1	0	0	1	0	0	1	0	1	0
D31.1	0	0	1	1	1	1	1	1	-1	1	0	0	1	1	1	0	1	0	1

Table 10: **DIN, KOUT, and DOUT for Valid Special Character Decoding**

Special Character Name	KIN	DIN[7:0]								DIN Unsigned (7 as MSB)
	K	H	G	F	E	D	C	B	A	
K28.0	1	0	0	0	1	1	1	0	0	28
K28.1	1	0	0	1	1	1	1	0	0	60
K28.2	1	0	1	0	1	1	1	0	0	92
K28.3	1	0	1	1	1	1	1	0	0	124
K28.4	1	1	0	0	1	1	1	0	0	156
K28.5	1	1	0	1	1	1	1	0	0	188
K28.6	1	1	1	0	1	1	1	0	0	220
K28.7	1	1	1	1	1	1	1	0	0	252
K23.7	1	1	1	1	1	0	1	1	1	247
K27.7	1	1	1	1	1	1	0	1	1	251
K29.7	1	1	1	1	1	1	1	0	1	253
K30.7	1	1	1	1	1	1	1	1	0	254

References

1. A. X. Widmer, P. A. Franaszek. *A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code* (IBM Journal of Research and Development, Vol. 27, Number 5, 1983). http://domino.research.ibm.com/tchjr/journalindex.nsf/0/b4e28be4a69a153585256bfa0067f59a?OpenDocument
2. A.X. Widmer. *The ANSI Fibre Channel Transmission Code* (IBM RC 18855, 1993) http://domino.watson.ibm.com/library/CyberDig.nsf/1e4115aea78b6e7c85256b360066f0d4/530f5ac982ac2e5b8525746600652c45?OpenDocument
3. XAPP1112: Parameterizable 8b/10b Decoder

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
10/31/08	1.0	Initial Xilinx release.
11/10/08	1.1	Updated to match release number of design files. No content changes.

Notice of Disclaimer