XAPP1112 (v1.1) November 10, 2008

Parameterizable 8b/10b Decoder

Author: Paula Vo

Summary

This application note describes a parameterizable 8b/10b Decoder, and is accompanied by a reference design that replaces the 8b/10b Decoder 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 Decoder reference design. All the features and interfaces included in the reference design are backward compatible with the LogiCORE IP 8b/10b Decoder v7.1 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 for this reason 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 Decoder reference design include:

- Decoding of 10-bit symbols into 8-bit bytes and an accompanying K bit
- Decoding of 268 unique transmitted characters: 256 byte values and 12 special (K) characters
- Decoder tracks running disparity to verify that the disparity sequence of the received 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 block
- Optional control inputs: Clock Enable (CE), Synchronous Initialization (SINIT), and Disparity Input (DISP_IN)
- Optional status outputs: Running Disparity (RUN_DISP), Disparity Error (DISP_ERR), Code Error (CODE_ERR), Symbol Disparity (SYM_DISP), and New Data (ND)

Interface

The 8b/10b Decoder Reference Design is synchronous. All inputs, Data Input (DIN), Synchronous Initialization (SINIT), and Disparity Input (DISP_IN) are sampled by the rising edge of the clock (CLK) input when Clock Enable (CE) is active. Outputs, Data Output (DOUT), Command Output (KOUT), Code Error (CODE_ERR), and New Data (ND) are registered at the rising edge of the clock (CLK) input when Clock Enable (CE) is active. Outputs, Symbol Disparity (SYM_DISP), Running Disparity (RUN_DISP), and Disparity Error (DISP_ERR) are registered outputs in the LUT-based Decoder, but are multiplexed outputs of a clocked memory in the block RAM-based Decoder.

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

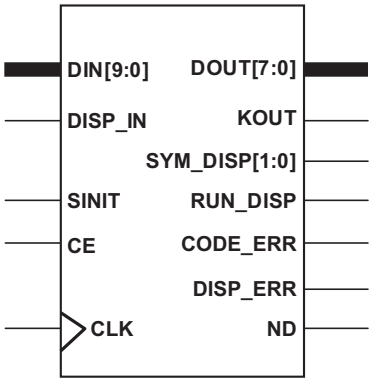

Figure 1: **8b/10b Decoder Schematic**

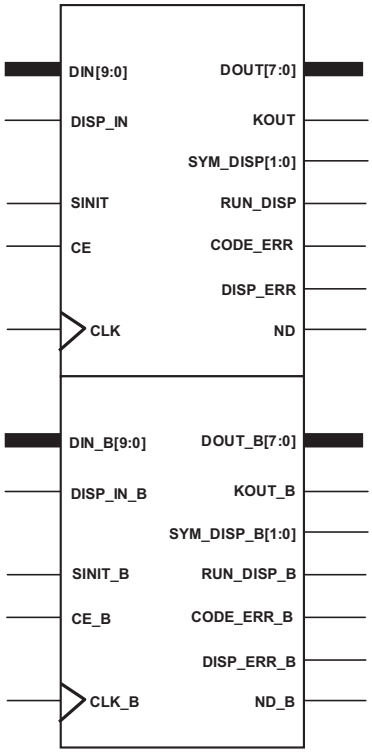

Figure 2: **Dual Decoder Schematic Symbol**

Decoder Input and Output Ports

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

I/O Pin Name	Direction Port Status	Description
CLK (CLK_B)	Input	**Clock**. All Decoder inputs are sampled and all outputs are updated synchronously on the rising edge of the CLK input. CLK_B is the clock input for the optional secondary B Decoder.
CE (CE_B) optional	Input	**Clock Enable**. The optional CE input port controls whether or not the Decoder responds to a change in the CLK input. When present, the CE input must be active (high) or the Decoder will not change state in response to the CLK input. When not present, the Decoder always changes state on the rising clock edge. CE_B is the clock enable input for the optional secondary B Decoder.
SINIT (SINIT_B) optional	Input	**Synchronous Initialization**. When the SINIT input port is present and set active (high), the Decoder is initialized on the rising edge of the clock to a defined state. DOUT is set to a user-selectable symbol. KOUT is set to 0. RUN_DISP is set to a user-defined disparity. SINIT_B is the synchronous initialization port for the B Decoder.
DIN[9:0] (DIN_B[9:0])	Input	**Data Input**. DIN is the Encoded 10-bit symbol input bus. DIN_B is the data input bus for the B Decoder.
DISP_IN (DISP_IN_B) optional	Input	**Disparity Input**. If present, this port is a manual input for the running disparity to which the current symbol's disparity is added. If not present, the running disparity used is the Decoder's internal running disparity. DISP_IN_B is the disparity input port for the B Decoder.
DOUT[7:0] (DOUT_B[7:0])	Output	**Data Output**. DOUT is the decoded 8-bit byte resulting from decoding the input 10-bit encoded symbol. DOUT_B is the data output bus for the B Decoder.
KOUT (KOUT_B)	Output	**Command Output**. If DIN is an encoding of one of the 12 special control characters, KOUT is active (high). KOUT_B is the command output flag for the B Decoder.
CODE_ERR (CODE_ERR_B) optional	Output	**Code Error**. If present, CODE_ERR is active (high) whenever DIN does not correspond to a valid member of the code set. CODE_ERR_B is the code error output flag for the B Decoder. CODE_ERR flags errors only when the 10-bit code on the DIN bus is inherently invalid. To capture all errors in an input sequence, this value must be combined with DISP_ERR.

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

I/O Pin Name	Direction Port Status	Description
SYM_DISP[1:0] (SYM_DISP_B[1:0]) optional	Output	**Symbol Disparity**. If present, SYM_DISP[1] designates a disparity error of the most recently decoded symbol, and SYM_DISP[0] designates the new running disparity. SYM_DISP_B is the symbol disparity for the B Decoder.
RUN_DISP (RUN_DISP_B) optional	Output	**Running Disparity**. If present, RUN_DISP designates the running disparity, including the disparity of the data symbol that has been decoded. 1 = running disparity of +1 0 = running disparity of -1 RUN_DISP_B is the running disparity output for the B Decoder.
DISP_ERR (DISP_ERR_B) optional	Output	**Disparity Error**. If present, DISP_ERR is active (high) whenever the most recently decoded symbol violated running disparity rules. Causes for a disparity error include invalid symbols and running disparity violations at both the symbol block and sub-block levels. DISP_ERR_B is the disparity error output flag for the B Decoder. Invalid symbols may or may not cause disparity errors depending on the symbol itself and the disparities of the previous and subsequent symbols. For this reason, DISP_ERR should always be combined with CODE_ERR to detect all errors.
ND (ND_B) optional	Output	**New Data**. If present and CE is present, ND is active (high) whenever the CE pin was high on the prior clock cycle, indicating that new data resides on the output pins. ND_B is the new data output flag for the B Decoder.

Pinout

The following sections provide detailed information about the 8b/10b Decoder signals.

Clock

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

Clock Enable

Clock Enable (CE, or CE_B for the B Decoder), an optional input, can be used to stall the Decoder, preventing it from responding to changes on the inputs. If the Decoder has a CE port and the CE input is inactive (logic 0), transitions on the clock port will 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.

Synchronous Initialization

Synchronous Initialization (SINIT, or SINIT_B for the B Decoder), an optional input, initializes the Decoder to a defined state. If SINIT is active (logic 1) and CE is active, the outputs and internal state of the Decoder are set on the rising edge of the clock. In this case, DOUT is set to a user-defined value, KOUT is set to zero, and the internal running disparity is set to a user-defined value. It is recommended to assert SINIT at times when the Decoder's state must be known, such as before receiving a packet, to ensure that the running disparity is initialized correctly before receiving.

Data Input Bus

Data Input Bus (DIN[9:0], or DIN_B[9:0] for the B Decoder) is the input bus that provides the 10-bit encoded symbol to the Decoder. In the case of serial data transmission, it is important to note that DIN[0] is the least-significant bit of the 10-bit encoded symbol. DIN must be aligned on symbol boundaries DIN[0]=a, DIN[9]=j for the Decoder to produce correct results. See “Appendix,” page 18 for additional information.

Disparity Input

Disparity Input (DISP_IN, or DISP_IN_B for the B Decoder), an optional input, overrides the Decoder's internal disparity. Normally, the running disparity (seen on the RUN_DISP output) is fed back, and on the rising edge of the clock added to the symbol disparity of the current symbol to produce the new running disparity. However, when DISP_IN is present, the new running disparity after the rising clock edge is defined as the symbol disparity combined with DISP_IN (instead of the old running disparity). A logic 0 asserted on the DISP_IN input indicates that an input disparity of -1 should be combined with the current symbol disparity, and a logic 1 asserted on the DISP_IN input indicates an input disparity of +1 should be used.

Data-Output Bus

Data-Output Bus (DOUT, or DOUT_B for the B Decoder) is the decoded output byte. When CE is inactive, DOUT holds the previously decoded data byte. The decoded data byte is the decoded value of the 10-bit symbol on the DIN bus after the rising clock edge.

Command Symbol Output Flag

Command Symbol Output Flag (KOUT, or KOUT_B for the B Decoder) indicates that the decoded 10-bit symbol is a command symbol or special character. Like DOUT, KOUT is updated after the rising clock edge to reflect the decoded value of the input symbol on the DIN bus. When KOUT is inactive (logic 0), the 10-bit symbol being decoded is a representation of an ordinary 8-bit byte (presented on DOUT). When KOUT is active (logic 1), the 10-bit symbol is an encoding of one of the 12 special characters and the value on the DOUT bus indicates the identity of the special character.

Code Error Flag

Code Error Flag (CODE_ERR, or CODE_ERR_B for the B Decoder), an optional output, indicates that the 10-bit symbol on the data-in bus during the rising edge of the clock was not a valid member of the code set. Active (logic 1) represents a CODE_ERR, and inactive (logic 0) indicates that the 10-bit code was a valid member of the code set (although other errors may still exist).

Symbol Disparity

Symbol Disparity (SYM_DISP[1:0], or SYM_DISP_B [1:0] for the B Decoder) is an optional output. SYM_DISP[1] represents a disparity violation at the 10-bit symbol currently being decoded, and SYM_DISP[0] represents the new running disparity for the Decoder.

The symbol disparity is combined with the current running disparity (+1 or -1) to generate the new running disparity for the Decoder. Most applications do not require symbol disparity, which is most useful as a debugging aid. To detect transmission faults, these implementations monitor DISP_ERR.

Running Disparity

Running Disparity (RUN_DISP, or RUN_DISP_B for the B Decoder), an optional output, exposes the current running disparity in the Decoder to external logic. Running disparity, by definition, must be +1 or -1. The RUN_DISP output uses a logic 1 to represent a running disparity of +1, and a logic 0 to represent a running disparity of -1.

The running disparity, on each rising edge of the clock, is combined with the symbol disparity of the symbol being decoded to produce the new running disparity. When the DISP_IN port is present, the running disparity is based on the current 10-bit symbol and the DISP_IN pin. When the DISP_IN port is not present, the running disparity is based on the current 10-bit symbol and the Decoder's internal running disparity. The initial state of the RUN_DISP output can be configured using the CORE Generator.

Disparity Error Flag

Disparity Error Flag (DISP_ERR, or DISP_ERR_B for the B Decoder), an optional output, indicates the presence of an error in the running disparity of the Decoder. A disparity error (represented by a logic 1) can be caused by many things:

- If the symbol disparity of the 10-bit symbol currently being decoded does not abide by the rules for running disparity, a disparity error is produced. It may be a valid 10-bit encoded symbol, but it must be used in the correct context of running disparity. Specifically, if the current running disparity is +1, the symbol disparity must be 0 or -2. Or, if the current running disparity is -1, the symbol disparity must be 0 or +2. In general, the running disparity must always remain -1 or +1.
- The same rules for running disparity apply to the 6-bit and 4-bit symbol sub-blocks. Violating running disparity rules at this level will generate a disparity error. See “Appendix,” page 18 for additional information.
- A disparity error also occurs for a majority of inputs that are not a member of the valid code set. These additional cases can also be detected because the CODE_ERR output is set.

New Data

New Data (ND, or ND_B for the B Decoder) an optional output, indicates to downstream logic that all outputs (DOUT, KOUT, all error flags, and so forth) have been updated with the results of the newly decoded symbol at the last rising edge of the clock. The ND output is a delayed version of CE, taking on the value of CE at the rising edge of the clock on each clock cycle. If the CE input port is not present, ND is not available. ND is set to logic 0 on the clock cycle following an SINIT input of a logic 1.

Hardware Implementation

The 8b/10b Decoder 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 Decoder essentially performs a translation from a 10-bit input to a 9-bit output (where the 9-bit output is comprised of an 8-bit data output and an additional K output to indicate whether the decoded byte is a special character). Meanwhile, additional logic determines whether 8b/10b disparity rules have been violated. The basic data decoding functionality may be readily mapped to a look-up table with a 1-to-1 correspondence between any given 10-bit encoded symbol and the corresponding decoded data byte. Any input symbol that does not correspond to a valid encoded symbol activates the CODE_ERR output. In the case of a code error, where the encoded symbol cannot be found in the look-up table, DOUT defaults to 1111_1111.

LUT-based Decoder

The LUT-based Decoder implementation uses slice logic to translate the 8b/10b decoding tables into VHDL case statements (Tables VI-VII in [2]). These case statements implement the partitioned 6-bit to 5-bit decoding and 4-bit to 3-bit decoding steps. Additional combinational logic implements the decoder error checking equations to detect invalid codes. See page 19 of the *ANSI Fibre Channel Transmission Code* research report [2].

Block RAM-based Decoder

The block RAM-based Decoder implementation is a true look-up table using a block RAM configured as a ROM. The table is mapped into a 1024 x 14-bit ROM with the 10-bit encoded symbol as the ROM address, as illustrated in Figure 3. The 1024-entry table describes the set of decoder outputs for all combinations of 10-bit encoded symbols. The 14-bit entries are composed of the 8-bit decoded data output: the KOUT flag, the Code Error flag, and the disparity outputs Running Disparity (RUN_DISP) and Disparity Error (DISP_ERR).

The lookup table contains two possible values for the Running Disparity and the Disparity Error outputs. Their final output values are selected based on whether the prior Running Disparity (or the Disparity Input, if this optional port is enabled) was positive or negative, as illustrated in Figure 4 and Figure 5.

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

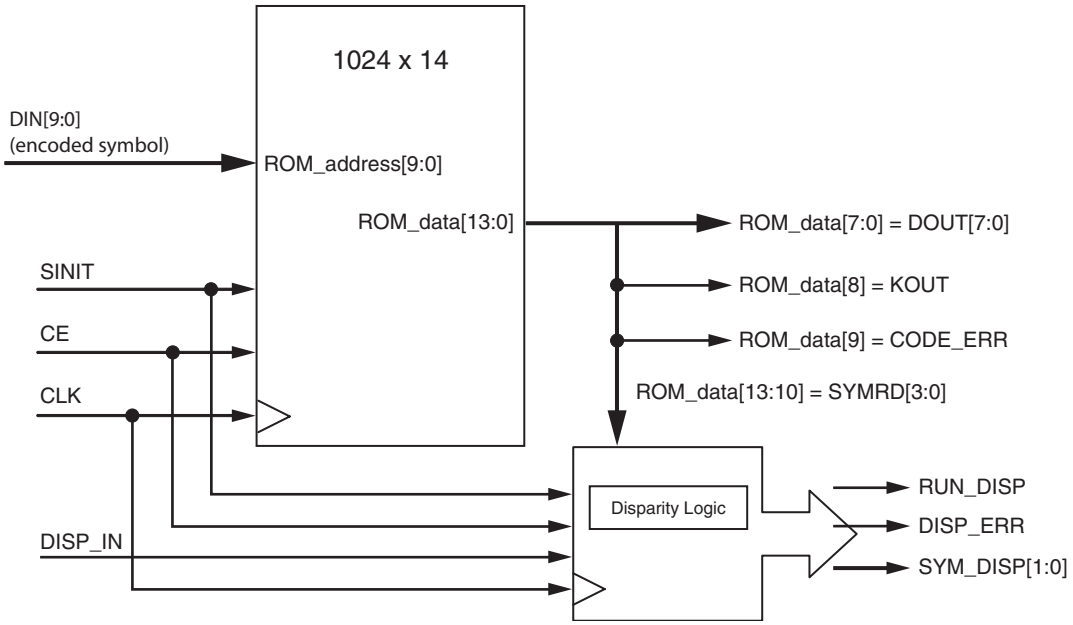

Figure 3: **Decoder Implemented as ROM Lookup Table**

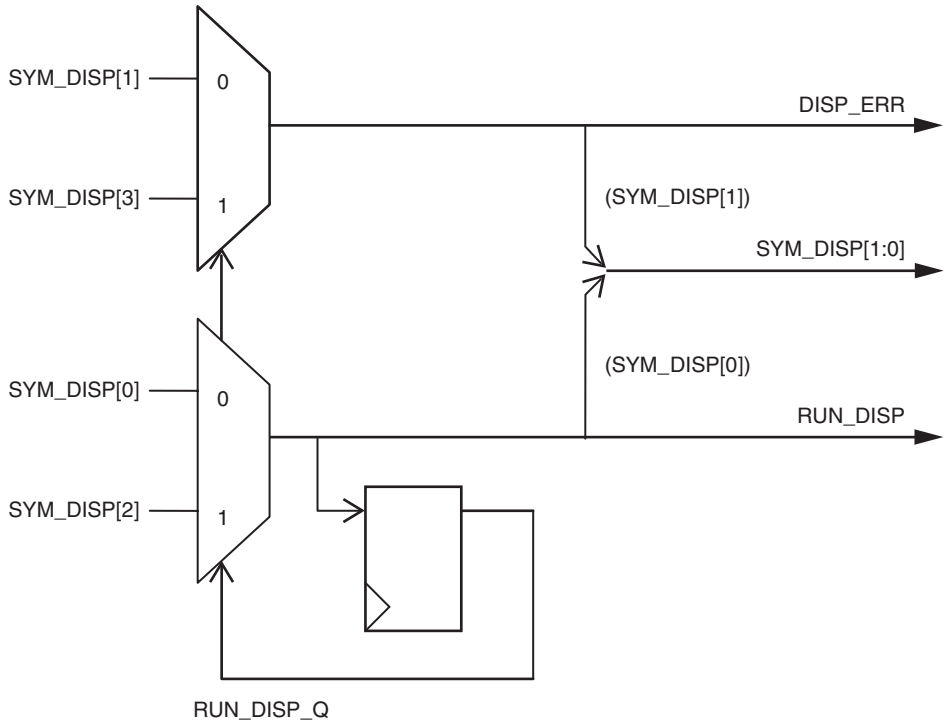

Figure 4: **Disparity Logic: Without DISP_IN Input**

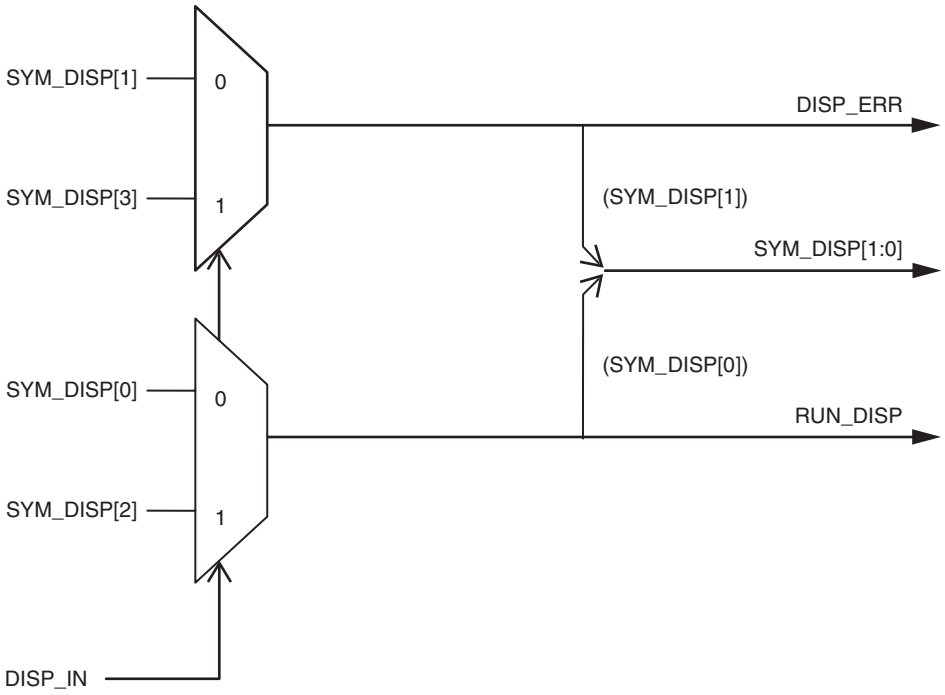

Figure 5: **Disparity Logic: With DISP_IN Input**

Possible Enhancements

Define an Error Code

When encoding an 8-bit data byte, the 8b/10b Encoder uses a K input (KIN) to determine whether the data byte is a regular data character (KIN = 0) or a special character (KIN = 1). For more information, see the 8b/10b Encoder application note, XAPP1122. 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 a DC-balanced code of the appropriate disparity. 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 assumes the symbol is valid and does 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 can flag the error with its KERR output and transmit the defined error code, with appropriate disparity, to the Decoder. The Decoder can be modified to interpret this specific code and flag a code error. This modification would entail changing the LUT-based Decoder error checking equations and subsequently regenerating an updated .mif file, if the block RAM-based Decoder is the targeted Decoder implementation.

Expand the Datapath Width

The 8b/10b Decoder datapath width may be doubled from an 8b/10b to a 16b/20b through the use of multiple decoders and minimal overhead logic. Two possible configurations are illustrated in Figure 6 and Figure 9. As illustrated in these examples, the two-decoder configuration (Figure 6) requires two clocks and lower resource utilization, and the three-decoder configuration (Figure 9) requires one clock and higher resource utilization.

Two-decoder Configuration

The two-decoder design displayed in Figure 6 can be implemented using the dual decoder configuration of the 8b/10b reference design. For compliance with the 8b/10b disparity rules, the lower sub-block is decoded before the upper sub-block, and the Running Disparity output of the lower sub-block drives the Disparity Input to the upper sub-block.

To ensure that the lower sub-block is decoded prior to the upper sub-block, the input data must arrive just prior to the active edge of the LSB Decoder, which can be accomplished by registering the input data on the opposite clock edge as the LSB Decoder clock (as illustrated in Figure 6).

Each decoder's Running Disparity output port is tied to the Disparity Input port of the opposite decoder. The two decoders can operate on opposite edges of the same clock (as shown in Figure 7), resulting in a single cycle of latency for the decoding operation. If additional latency can be tolerated in the design (two cycles per decode), the clocking can be simplified by operating the two decoders on the same clock, but with their Clock Enables toggling on alternate cycles, as shown in Figure 8.

Three-decoder Configuration

The three-decoder design illustrated in Figure 9 can be implemented using a single decoder combined with a dual decoder. In this configuration, both the upper and lower sub-blocks are decoded on the same clock edge. This may be accomplished by duplicating the MSB Decoder logic and precalculating the results for the case where the Disparity Input = 1 and the case where the Disparity Input = 0.

After the running disparity of the LSB Decoder is known, it is used to select the correct results from the precalculated set. This select logic is combinational, and may cause glitches on the Running Disparity, Disparity Error, and Symbol Disparity outputs of the MSB Decoder, as shown in Figure 10. For this reason, it is recommended that all outputs of this Decoder architecture should be registered. Select logic is not required for the MSB Decoder's Data Output (DOUT), K Output (KOUT_B), Code Error (CODE_ERR_B), and New Data (ND_B) signals, because they are independent of the running disparity and can be taken from either of the instantiated MSB Decoders.

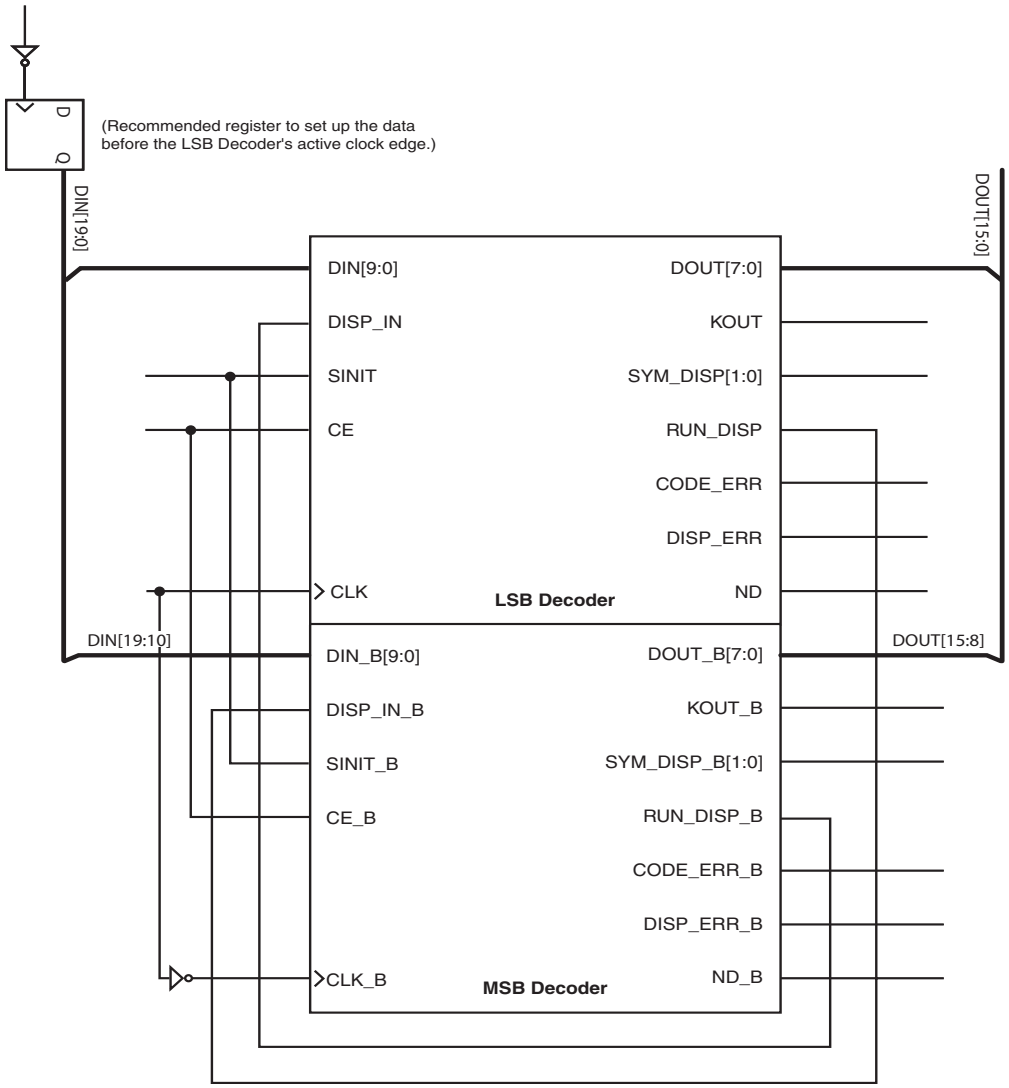

Figure 6: **Two-decoder Configuration**

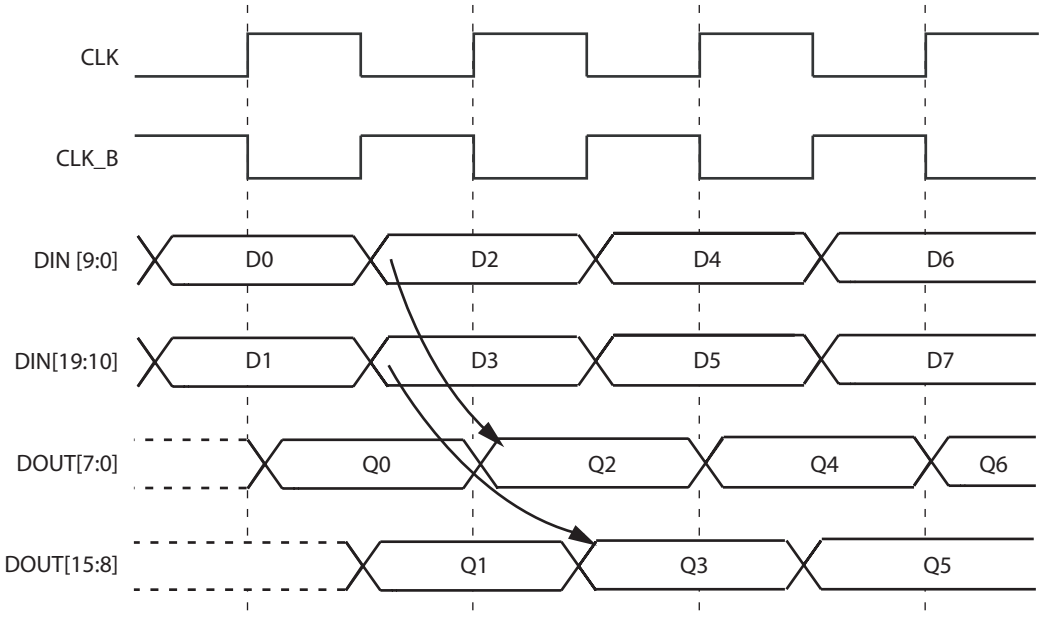

Figure 7: **Using Rising and Falling Clock Edges**

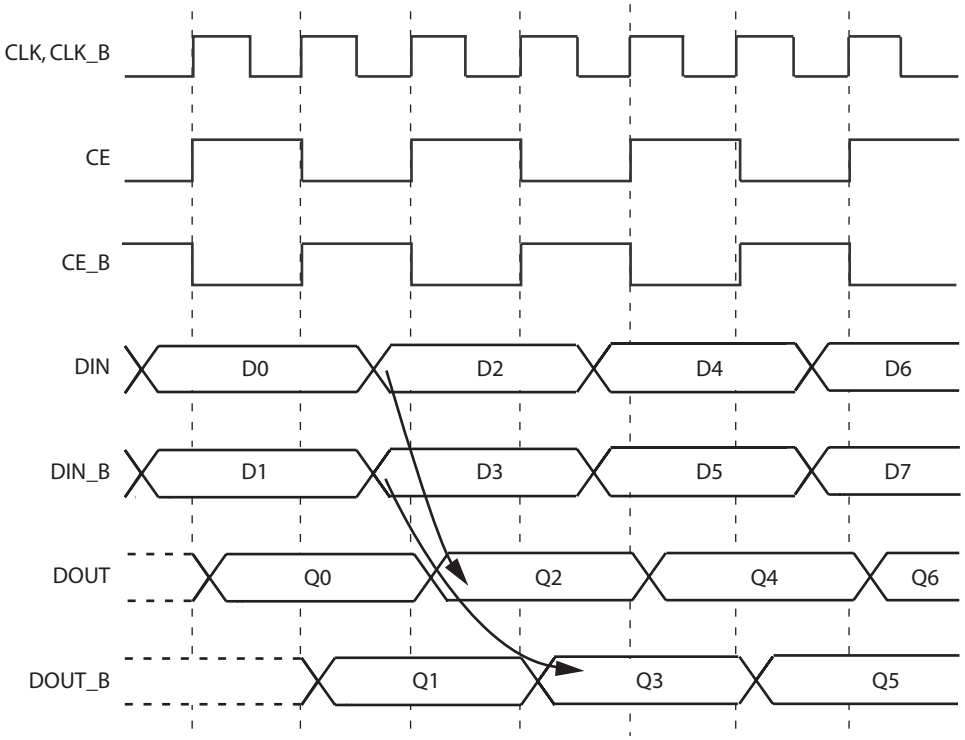

Figure 8: **Using Two Cycles to Decode**

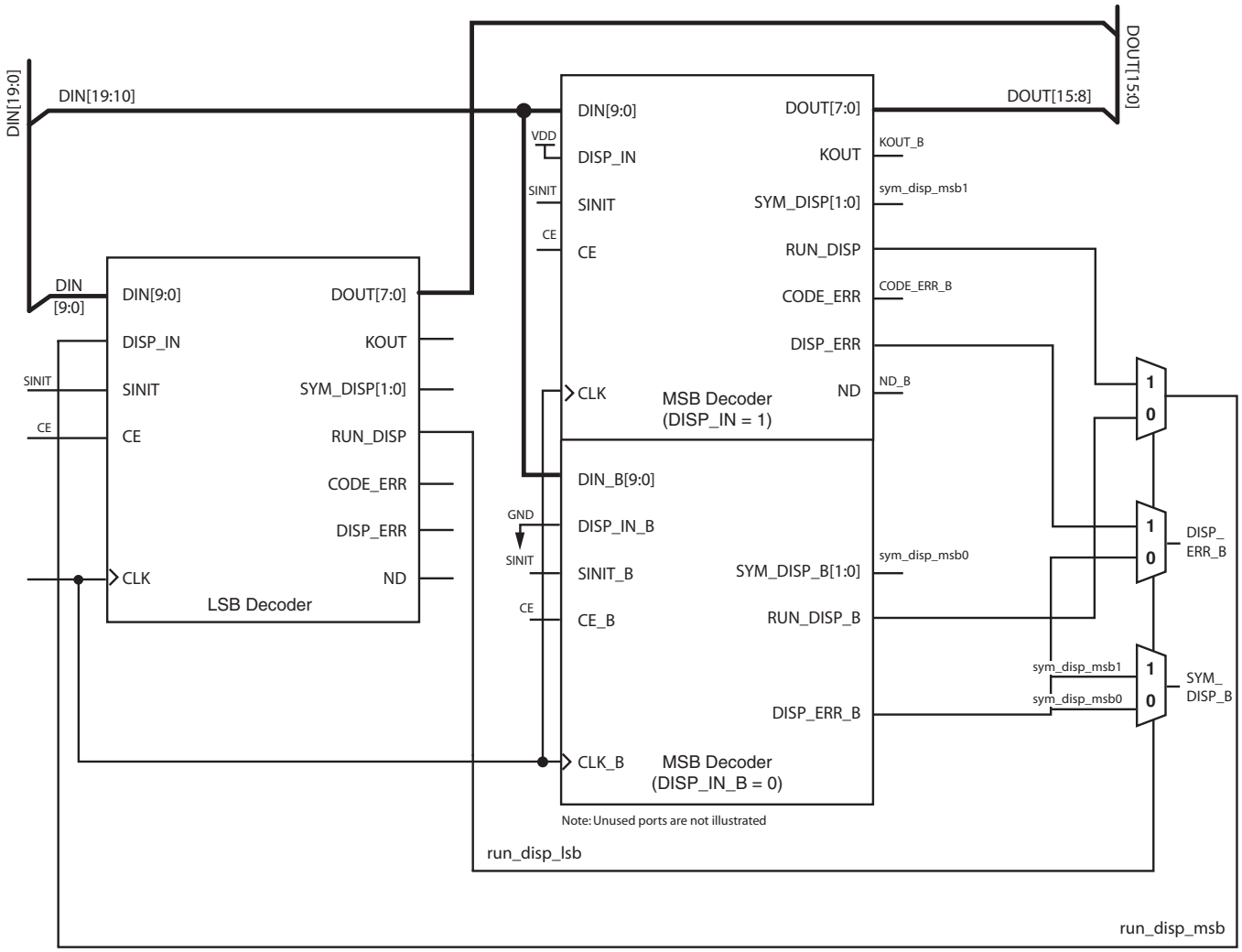

Figure 9: **Three-decoder Configuration**

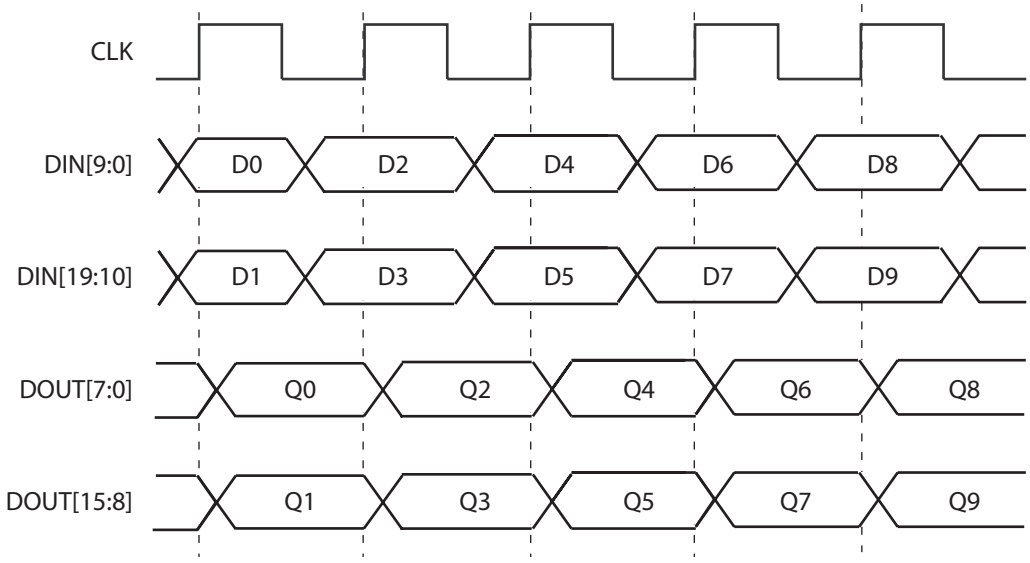

Figure 10: **Using One Clock, One Cycle to Decode**

Reference Design

The 8b/10b Decoder 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 regenerate the block RAM initialization .mif file, if desired. Table 2 defines the reference design source files, and Table 3 defines the scripts, project files, and documentation.

Table 2: **8b/10b Decoder Design Files**

Filename	Description
decode_8b10b_wrapper.vhd	VHDL RTL for the top-level core wrapper file
decode_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 25 generics in the top-level core file.
decode_8b10b_pkg.vhd	VHDL package file of constants and functions
decode_8b10b_rtl.vhd	VHDL RTL for the top-level core file
decode_8b10b_lut.vhd	VHDL RTL for the LUT-based Decoder
decode_8b10b_lut_base.vhd	VHDL RTL for a single LUT-based Decoder
decode_8b10b_bram.vhd	VHDL RTL for the block RAM-based Decoder
decode_8b10b_disp.vhd	VHDL RTL for the disparity logic in the block RAM-based Decoder
dec.mif	ASCII text file containing the 1024 x 14-bit table to initialize the block RAM-based Decoder

Table 3: **8b/10b Decoder Documentation and Script Files**

Filename	Description
README_XAPP1112.txt	Describes the reference design files and script files, and instructions for executing 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 Decoder 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_dec.vhd and the LUT-based Decoder RTL in ModelSim to generate a new dec.mif initialization file if the Decoder behavior (RTL) is modified
mifgen_dec.vhd	Provides stimulus to the LUT-based Decoder that exercises all possible code points and captures the resulting table to the dec.mif file
MakeMIF_mti.do	ModelSim .do script to generate a new dec.mif file

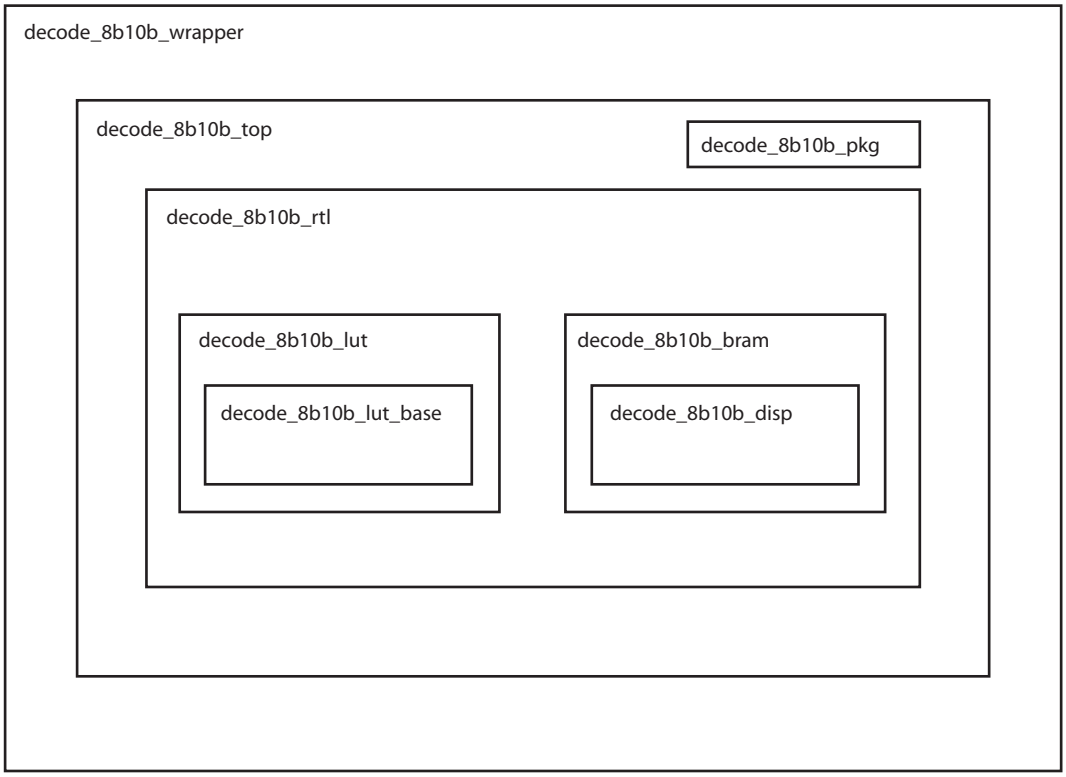

Figure 11: **RTL Hierarchy**

Compilation Parameters

The 8b/10b Decoder 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 Decoder implementation, a user may manually edit the 25 complete generics in the top-level core file and implement the design without the wrapper files. Table 4 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-Decoder option is selected (C_HAS_BPORTS = 1), the second decoder is configured with the same optional ports as the first Decoder. However, their initialization values may differ.

Table 4: **RTL Parameters**

Generic	Simplified Generics	Description
C_DECODE_TYPE	C_DECODE_TYPE	Implementation: 0=LUT-based, 1=BRAM-based
C_ELABORATION_DIR	*none*	Hard-coded path to dec.mif file location
C_HAS_BPORTS	C_HAS_BPORTS	1 indicates second decoder should be generated
C_HAS_CE	C_HAS_CE	1 indicates CE(_B) port is present
C_HAS_CE_B		

Table 4: **RTL Parameters *(Cont'd)***

Generic	Simplified Generics	Description
C_HAS_CODE_ERR	C_HAS_CODE_ERR	1 indicates CODE_ERR(_B) port is present
C_HAS_CODE_ERR_B		
C_HAS_DISP_ERR	C_HAS_DISP_ERR	1 indicates DISP_ERR(_B) port is present
C_HAS_DISP_ERR_B		
C_HAS_DISP_IN	C_HAS_DISP_IN	1 indicates DISP_IN(_B) port is present
C_HAS_DISP_IN_B		
C_HAS_ND	C_HAS_ND	1 indicates ND(_B) port is present
C_HAS_ND_B		
C_HAS_RUN_DISP	C_HAS_RUN_DISP	1 indicates RUN_DISP(_B) port is present
C_HAS_RUN_DISP_B		
C_HAS_SINIT	C_HAS_SINIT	1 indicates SINIT(_B) port is present
C_HAS_SINIT_B		
C_HAS_SYM_DISP	C_HAS_SYM_DISP	1 indicates SYM_DISP(_B) port is present
C_HAS_SYM_DISP_B		
C_SINIT_DOUT	C_SINIT_VAL	10-bit binary string to initialize DOUT, KOUT, and RUN_DISP
C_SINIT_KOUT		
C_SINIT_RUN_DISP		
C_SINIT_DOUT_B	C_SINIT_VAL_B	10-bit binary string to initialize DOUT_B, KOUT_B, and RUN_DISP_B
C_SINIT_KOUT_B		
C_SINIT_RUN_DISP_B		

Table 5: **SINIT Value**

Dialog Option	C_SINIT_VAL		
Symbol Choice	C_SINIT_DOUT	C_SINIT_KOUT	C_SINIT_RUN_DISP
D.0.0 (pos)	000 00000	0	1
D.0.0 (neg)	000 00000	0	0
D.10.2 (pos)	010 01010	0	1
D.10.2 (neg)	010 01010	0	0
D.21.5 (pos)	101 10101	0	1
D.21.5 (neg)	101 10101	0	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 Decoder varies depending on the specific configuration implemented. The implementation may be a single decoder or dual decoder, block RAM-based or LUT-based, and may have a variety of optional input and output ports. Table 6 and Table 7 define the resource utilization and performance for the 8b/10b Decoder for the following four basic configurations:

- 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 6: **Resource Utilization**

	Configuration		FF	LUT	Block RAM
Spartan-3A xc3s50a tq144	**Block RAM-based**	**ALL**	4	8	2
		NONE	0	0	1
	LUT-based	**ALL**	26	218	0
		NONE	9	36	0
Virtex-5 xc5vlx20t ff323	**Block RAM-based**	**ALL**	4	8	1
		NONE	0	0	1
	LUT-based	**ALL**	26	128	0
		NONE	9	11	0

Table 7: **Performance (MHz)**

		Spartan3-A xc3s50a-tq144		Virtex-5 xc5vlx20t-ff323	
		-4	**-5**	**-1**	**-2**
Block RAM-based	**ALL**	200	240	260	300
	NONE	260	320	340	380
LUT-based	**ALL**	100	120	260	280
	NONE	200	260	630	700

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 a successful application of the 8b/10b Decoder.

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 values 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 the 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 Decoder tracks the running disparity of the input data stream for error checking purposes. At the end of each 10-bit encoded symbol, the running disparity is +1 or -1, which equates to a 1 or 0, respectively, on the RUN_DISP output. Each symbol's disparity is combined with the current running disparity to produce the new running disparity. Symbols with a disparity of 0 would then not modify the running disparity [+1 + (0) = +1] or [-1 + (0) = -1]. Symbols with a disparity of +2 or -2 would swap the running disparity between +1 and -1; for example, [+1 + (-2) = -1] or [-1 + (+2) = +1].

If the symbol disparity is anything other than +2, 0, or -2, the disparity of the incoming data is invalid and a disparity error is flagged (DISP_ERR = 1). Such an error also occurs if the running disparity is +1 and the symbol disparity is +2, or when the running disparity is -1 and the symbol disparity is -2. Most invalid input symbols will also result in a disparity violation.

Running disparity validity is also enforced at the sub-block level. Since the 10-bit block is actually composed of a 6-bit block and a 4-bit block, the same rules for running disparity apply at those sub-block boundaries. The disparity of each sub-block must be +2, 0, or -2. The running disparity at the end of the sub-block must follow the rule that running disparity must be -1 or +1. Failure to meet this criteria will also be flagged as a disparity error by the Decoder.

The Decoder uses the scheme in Widmer and Franaszek's paper to decode the 10-bit input symbol into an 8-bit output value. This paper's nomenclature defines the bits of the 10-bit symbol as abcdei_fghj, where a is the LSB and j is the MSB. This 10-bit encoded symbol is partitioned as a 6-bit sub-block (abcdei) and a 4-bit sub-block (fghj). In the same way, the 8-bit output value is defined as the concatenation of a 5-bit and a 3-bit sub-block (ABCDE and FGH respectively). As defined in the original paper, these 8 bits are named ABCDE_FGH where A is the LSB, and H is the MSB.

When decoding the 10-bit input symbol, the decoder first decodes the six least significant bits abcdei of the input symbol into the five least significant bits of the output, ABCDE. Table 8 shows an example of this decoding for the symbol designated D31.1. The five output bits ABCDE equate to 31 in decimal (EDCBA=11111), thus providing the first part of the symbol name. Similarly, the four most significant bits fghj of the input symbol decode into the three most significant bits of the output FGH. The three output bits, FGH, represent a decimal 1, providing the second part of the symbol name (HGF=001).

Table 9 defines the decoding of the 12 special command symbols for both positive and negative disparity cases. These special characters are distinguished from the 256 standard characters by the KOUT output being asserted when the symbol is decoded. The DOUT value identifies which of the twelve special characters have been decoded.

Table 8: **Example Decoding of d31.1 for Both Running Disparity Cases**

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

Table 9: **DIN, KOUT, and DOUT for Valid Special Character Decoding**

	DIN[9:0]										KOUT	DOUT[7:0]								DOUT Unsigned (7 as MSB)
	j	h	g	f	i	e	d	c	b	a	K	H	G	F	E	D	C	B	A	
K28.0	0	0	1	0	1	1	1	1	0	0	1	0	0	0	1	1	1	0	0	28
K28.0	1	1	0	1	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	28
K28.1	0	1	1	0	0	0	0	0	1	1	1	0	0	1	1	1	1	0	0	60
K28.1	1	0	0	1	1	1	1	1	0	0	1	0	0	1	1	1	1	0	0	60
K28.2	1	0	1	0	1	1	1	1	0	0	1	0	1	0	1	1	1	0	0	92
K28.2	0	1	0	1	0	0	0	0	1	1	1	0	1	0	1	1	1	0	0	92
K28.3	1	1	0	0	1	1	1	1	0	0	1	0	1	1	1	1	1	0	0	124
K28.3	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	124
K28.4	0	1	0	0	1	1	1	1	0	0	1	1	0	0	1	1	1	0	0	156
K28.4	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	0	0	156
K28.5	0	1	0	1	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0	188
K28.5	1	0	1	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	188
K28.6	0	1	1	0	1	1	1	1	0	0	1	1	1	0	1	1	1	0	0	220
K28.6	1	0	0	1	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	220
K28.7	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	252
K28.7	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	252
K23.7	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1	1	1	247
K23.7	1	1	1	0	1	0	1	0	0	0	1	1	1	1	1	0	1	1	1	247
K27.7	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	251
K27.7	1	1	1	0	1	0	0	1	0	0	1	1	1	1	1	1	0	1	1	251
K29.7	0	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	0	1	253
K29.7	1	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	1	0	1	253
K30.7	0	0	0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	254
K30.7	1	1	1	0	1	0	0	0	0	1	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. XAPP1122: Parameterizable 8b/10b Encoder

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