XAPP336 (v1.0) July 15, 2000

Design of a 16b/20b Encoder/Decoder Using a CoolRunner CPLD

Summary

This document details the VHDL implementation of a fibre channel byte-oriented transmission encoder and decoder in a Xilinx CoolRunner CPLD. CoolRunner CPLDs are the lowest power CPLDs available today and can be utilized in any network design where reliable point-to-point transceivers are required. CoolRunner CPLDs utilize the patented Fast Zero Power (FZP) design technique to simultaneously deliver high performance and low power consumption. These devices offer pin-to-pin delays of 5.0 ns, and less than 100 µA of standby current (approximately 1/3 of the power consumed by other competing CPLDs at f_{MAX}). To obtain the complete VHDL 16b/20b encoder or decoder design code described in this document, refer to section "VHDL Code Download" on page 25 for instructions.

Disclaimer

THIRD PARTIES INCLUDING IBM MAY HAVE PATENTS ON A STANDARD BYTE ORIENTED DC BALANCED 8B/10B PARTITIONED BLOCK TRANSMISSION CODE. BY PROVIDING THIS REFERENCE DESIGN AS ONE POSSIBLE IMPLEMENTATION OF THIS STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THE PROVIDED IMPLEMENTATION OF THIS STANDARD IS FREE FROM ANY CLAIMS OF INFRINGEMENT BY ANY THIRD PARTY. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WITH RESPECT TO THE ADEQUACY OFTHE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OR REPRESENTATION THAT THE IMPLEMENTATION IS FREE FROM CLAIMS OF ANY THIRD PARTY. FURTHERMORE, XILINX IS PROVIDING THIS REFERENCE DESIGNS "AS IS" AS A COURTESY TO YOU.

Notice

This application note is not intended to provide complete documentation on 8b/10b. This document gives a brief introduction to 8b/10b, how the encoding/decoding scheme is generated, and how it is implemented in a 16b/20b VHDL application targeted to a CoolRunner CPLD. For more information on 8b/10b coding rules, please refer to section "References" on page 26.

Introduction

Today binary codes are used in transmission schemes across fiber optic links. The binary on/off mode provides a transmission scheme ideal for high-speed local area networks and computer links. The 8b/10b data transmission scheme has become the standard for high-speed serial links today. An 8b/10b module provides byte synchronization and the encode/decode scheme for fibre channel communication links. The 8b/10b scheme is part of the physical network layer and can be utilized in any gigabit Ethernet, ATM, wireless or fiber optic transmission link.

Many communication systems today transmit information in the form of packets with a defined field structure for both communication and error control. The 8b/10b encoding scheme translates byte-wide data of random "1s" and "0s" into a 10-bit serial data stream. The encoding mechanism generates a balanced bit stream, in which an equal number of bit transitions occur between a logic High and a logic Low level. The 8b/10b encoding rules create a DC balanced code that provides optimum coding efficiency, clock recovery, error detection, and suitability for ring or point-to-point topologies.

A16b/20b transmission scheme incorporates the idea of the 8b/10b transmission code by combining two 8b/10b modules side-by-side. With 16b/20b encoding, a 16-bit word can be encoded and transmitted serially as shown in Figure 1.

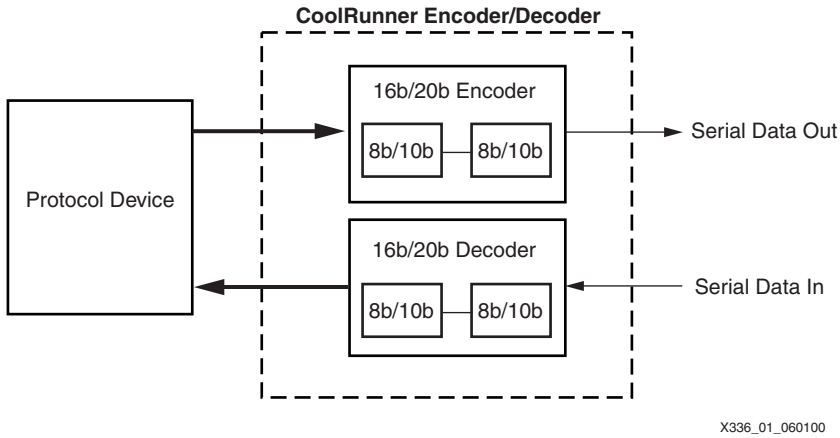

Figure 1: **16b/20b Block Diagram**

The 8b/10b code provides many advantages for fiber optic and electromagnetic wire links. High-gain fiber optic receivers need an AC coupling stage near the front end as well as simplified control of transmitter level, receiver gain, equalization, and maintenance of precise signal power. This becomes more important at high data rates to maintain constant byte rate and reduce redundancy checks to each byte. Lower signaling rates of the 8b/10b code can be utilized to minimize crosstalk and compensate for an increase in signal-to-noise ratio.

8b/10b Background

The 8b/10b transmission code includes serial encoding and decoding rules, special characters, and error detection. The characters defined by this code ensure that short run lengths and sufficient transitions are present in the serial bit stream to make clock recovery possible at the receiver. For this reason, the 8b/10b encoding scheme has the ability to control the characteristics of each code word by creating a limited change of "0s" and "1s". The encoding greatly increases the likelihood of detecting single or multiple errors during the transmission of data.

Each 8b/10b encoding and decoding module is capable of the following main functions:

- Error detection
- Frame delimination with data transparency
- Clock recovery: good signal transition density helps to find the center of each data bit
- DC voltage balancing
- Fiber optic, wireless or ATM implementation capabilities

Structure of 8b/10b Code

The 8b/10b transmission code includes D-characters (used for data transmission) and K-characters (used for control and protocol functions). Parity is monitored in each byte of transmission and both D- and K-characters are based on positive or negative parity. The parity of each code word is selected by the encoder to maintain a balanced running parity in the data stream.

For encoding purposes, each incoming byte of data is partitioned into two subblocks as shown in Figure 2. The five binary lines, **ABCDE**, are encoded into six binary lines, **abcdei**, which

follow the rules of 5b/6b encoding. Similarly, the three bits, **FGH**, are encoded into **fghj**, which follow the 3b/4b encoding rules.

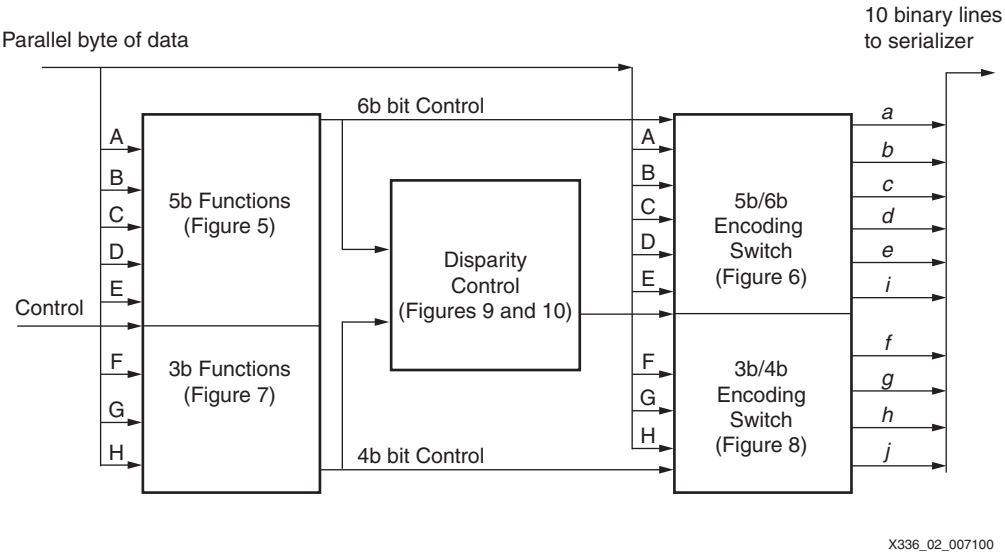

Figure 2: **8b/10b Encoding Block Diagram**

The combination of a 5b/6b and 3b/4b encoder blocks limits error propagation to five bits as well as simplifies the encoder and decoder modules. Both the 5b/6b and 3b/4b encoding rules will be discussed in detail.

Every 10-bit encoded data group has one of three possibilities to help limit the number of consecutive "1s" or "0s" in any 2-code words:

- Five "1s" and five "0s"
- Four "1s" and six "0s"
- Six "1s" and four "0s"

Byte Synchronization

Byte synchronization is accomplished through the use of special characters. Select special characters allow for byte stuffing to control the number of transitions in a data stream. Special characters are discussed in more detail in section "Special Characters" on page 5.

Running Disparity

DC balancing is achieved through the use of running disparity. This function controls the number of ones and zeros in a single transmission. This helps balance the DC level between the "1" and "0" voltage level. Running disparity is positive (+) if more "1s" than "0s" have been transmitted and negative (–) when more "0s" than "1s" are transmitted. The running disparity remains unchanged from the previous transmission if the code has an equal number of "1s" and "0s". No distinction is made between 6b and 4b blocks, so they are used to compensate each other. Similarly, the disparity in each 8b/10b module is used to compensate the other in a 16b/20b design. For this design, the disparity out of each 16-bit word data transmission is translated as the running disparity when sending a packet of data. The running disparity is then used as the input disparity when encoding the next 16-bit data word to send. Since the disparity is monitored, the transmitted code is free of any DC component.

Error Detection

Error detection in the 8b/10b code is done in two manners:

- Error checking using redundancy: Validates transmission of each packet, in which a higher level OSI protocol layer would detect errors based on start and end delimiters.
- Cyclic redundancy checks (CRC): Detects errors on individual 6b or 4b subblocks based on the rules for 8b/10b encoding.

8b/10b Code Definition

The 8b/10b encoding is accomplished by encoding the ABCDE and FGH inputs as two units, the 5b/6b and 3b/4b, respectively. The 10b/8b decoder module is discussed in detail later, and provides the complementary function of the 8b/10b encoder.

5b/6b Definition

Table 1 shows a few examples of the 5b/6b encoding rules. **ABCDE** represents the lower five bits of the data to send, DATA_IN[0:4] respectively. The transmitted data, **abcdei**, is shown with the corresponding alternate data that is sent based on the disparity value. The running disparity, Din, can be either positive (+), negative (–), or don't care (x). The disparity out of the encoded word, Dout, is shown with the transmitted data. When the alternate data is sent, disparity out (Dout) will be complemented. Dout can be either positive (+), negative (–), or not affected (0). For each data byte, the bit encoding shows the category each incoming data byte falls into according to the number of zeros and ones. Based on the bit encoding the incoming data bits, **ABCDE** are translated. If the bit encoding column is left blank, then the data bits do not change and the added **i** bit is "0". For example, in Table 1, D.0 has bit encoding classification L04, which represents zero "1s" and four "0s" in the **ABCD** bits of the data, D.0. The result encoding is then 011000 when the previous running disparity is (+), now resulting in a current negative (–) disparity out. If the input running disparity is (–) then the alternate data, 100111, is sent and the disparity out is (+). The disparity classification shows which functions are used in generating the disparity out for each data byte. These functions are described in more detail in "8b/10b Design" on page 10. Table 1 shows examples of non-special characters, where the **K** control character is held at "0".

Table 1: **5b/6b Encoding**

Name	Data to Send						Classification		Din	Transmitted Data						Dout	Alternate Data
	A	B	C	D	E	K	Bit Encoding(1)	Disparity(1,2,3)		a	b	c	d	e	i		
D.0	0	0	0	0	0	0	L04	L22' ·L31' ·E'	+	0	1	1	0	0	0	–	100111
D.1	1	0	0	0	0	0	L13·E'	L22' ·L31' ·E'	+	1	0	0	0	1	0	–	011101
D.3	1	1	0	0	0	0	L22·E'		x	1	1	0	0	0	1	0	
D.7	1	1	1	0	0	0		L31 ·D' ·E'	–	1	1	1	0	0	0	0	000111
D.15	1	1	1	1	0	0	L40	L22' ·L31' ·E'	+	1	0	1	0	0	0	–	010111
D/K.23	1	1	1	0	1	x		L22' ·L13' ·E	–	1	1	1	0	1	0	+	000101
D.31	1	1	1	1	1	0	L40, L40·E	L22' ·L13' ·E	–	1	0	1	0	1	1	+	010100

Notes:

1. The (·) represents the AND function.
2. The (') represents negation, for example, L22' represents the negative assertion of L22.
3. The variable E represents bit E in the Data to Send or DATA_IN[4].

3b/4b Definition

Table 2 shows examples of the coding scheme for the 3b/4b module. The 3b/4b follows the same convention and notation as previously shown in Table 1 for the 5b/6b code. Notice under the Name column in Table 2 that D.x.1 represents any pattern of **ABCDE** with the decimal 1 for the **FGH** data. For the case of D/K.x.3 in Table 2, the assertion of **K** indicates whether the data **F** through **H** represents data or control information. Note the **FGH** bits of the data to send in Table 2, represent DATA_IN[5:7] respectively.

Table 2: **3b/4b Encoding**

	Data to Send				Classification			Transmitted Data					
Name	F	G	H	K	Bit Encoding(1)	Disparity(1,2,3)	Din	f	g	h	j	Dout	Alternate Data
D/K.x.0	0	0	0	x	F'·G'·H'	F' ·G'	+	0	1	0	0	–	1011
D.x.1	1	0	0	0	(F≠G)·H'		x	1	0	0	1	0	
D/K.x.3	1	1	0	x		F·G	–	1	1	0	0	0	0011
D.x.P7	1	1	1	0		F·G, F·G·H	–	1	1	1	0	+	0001
D/K.y.A7	1	1	1	x	F·G·H·(S+K)(1)	F·G,F·G·H	–	0	1	1	1	+	1000
K.28.1	1	0	0	1	(F≠G)·H'	(F≠G)·K	+	1	0	0	1	0	0110
K.28.5	1	0	1	1		(F≠G)·K	+	1	0	1	0	0	0101

Notes:

1. The (·) represents the AND function.
2. The (') represents negation, for example, H' represents the negative assertion of bit H.
3. S={c·i·(D–1= –)} OR {c' ·i' ·(D–1= +)} (where c and i come from the 5b/6b encoded data)

The 3b/4b code has a few exceptions and requires an explanation. The encoding of D.x.P7 (primary 7) is replaced by D/K.y.A7 (alternate 7) to eliminate the run length of five continuous "1s" or "0s" with the **abcdei** data when the following equation holds valid:

[(e=i=1) and ((Din) = (-))] or
[(e=i=0) and ((Din) = (+))] or
(K=1)

The special characters section identifies this exception with corresponding encoding patterns of the **ABCDE** data.

Special Characters

Special characters are defined as extra signal codes needed beyond the 256 (2^8) characters already established with K=0. When asserted, the input K character (K=1) recognizes that a special character is being transmitted. Special characters are generally used for transmitting code words such as ABORT, RESET, SHUTDOWN, IDLE, and link diagnostics. In the 8b/10b code, 12 special characters exist as shown below in Table 3, where **A-H** is the data byte to encode, and **a-j** is the encoded transmitted data. Note, under the Name column K.23.7 represents the decimal 23 in bits **A-E** and 7 in bits **FGH**. Also note, the data to send **ABCDE FGH** is encoded DATA_IN[0:7] respectively. Also in Table 3, Din is the running parity and Dout

is the parity of the encoded data. All special characters comply with the general coding constraints of a maximum run length of 5.

Table 3: **Special Characters (K = 1)**

Name	Data to Send ABCDE FGH K	Din	Transmitted Data abcdei fghj	Dout	Alternate Data
K.28.0	00111 000 1	–	001111 0100	0	110000 1011
K.28.1(1)	00111 100 1	–	**001111 1**001	+	**110000 0**110
K.28.2	00111 010 1	–	001111 0101	+	110000 1010
K.28.3	00111 110 1	–	001111 0011	+	110000 1100
K.28.4	00111 001 1	–	001111 0010	0	110000 1101
K.28.5(1)	00111 101 1	–	**001111 1**010	+	**110000 0**101
K.28.6	00111 011 1	–	001111 0110	+	110000 1001
K.28.7(1)	00111 111 1	–	**001111 1**000	0	**110000 0**111
K.23.7	11101 111 1	–	111010 1000	0	000101 0111
K.27.7	11011 111 1	–	110110 1000	0	001001 0111
K.29.7	10111 111 1	–	101110 1000	0	010001 0111
K.30.7	01111 111 1	–	011110 1000	0	100001 0111

Notes:
1. Singular Comma

Singular Commas

Byte synchronization is accomplished through the use of the singular comma and the IDLE sequence. The singular comma in the 8b/10b code is a sequence with one run length of 2 ending at position **b**, contiguous with a sequence of the inverted signal with a run length of 5. These singular comma bit patterns are shown in bold and noted in Table 3. For example in Table 3, singular comma K.28.5 has the encoding bit pattern **001111 1**000, where the first two bits are "0" and the next five are "1s" and the alternate data **110000 0**111 is just the opposite. These bit patterns are used to synchronize frames of data during packet transmissions.

10b/8b Decoding

Decoding for each 10b/8b module can be divided for each 6b/5b and 4b/3b decoding scheme, even though all inputs (**abcdei fghj**) are used in decoding each bit (AOUT.. HOUT). The implementation of the decoding logic is a complement to the encoding logic described later in "8b/10b Design" on page 10, but does not depend on the disparity. The disparity classifications for 6b/5b and 4b/3b decoding are needed only for error detection. Table 4 and Table 5 show a few examples of the 6b/5b and 4b/3b decoding rules. In Table 4 and Table 5, Pxy indicates the number of "1s" (x) and the number of "0s" (y) in the **abcd** incoming data.

Table 4: **6b/5b Decoding Scheme**

Name	Received Data a	b	c	d	e	i	Classification Decoding Class(1,2)	Disparity(1,2)	Din	ABCDE K	Dout
D.0	0	1	1	0	0	0	P22·b·c·(e=i)	P22·e’ ·i’	+	00000 0	–
D.0	1	0	0	1	1	1	P22·b’ ·c’ ·(e=i)	P22·e·i	–	00000 0	+
D.2	0	1	0	0	1	0	P13·i’	P13·i’	+	01000 0	–
D.2	1	0	1	1	0	1	P31·i	P31·i	–	01000 0	+

Table 4: **6b/5b Decoding Scheme**

Name	Received Data						Classification		Din	ABCDE K	Dout
	a	b	c	d	e	i	Decoding Class(1,2)	Disparity(1,2)			
D.5	1	0	1	0	0	1			x	10100 0	0
D.15	1	0	1	0	0	0	P22·a·c·(e=i)	P22·e' ·i'	+	11110 0	–
D.15	0	1	0	1	1	1	P22·a' ·c' ·(e=i)	P22·e·i	–	11110 0	+
D.28	0	0	1	1	1	0			x	00111 0	0
K.28	0	0	1	1	1	1	c·d·e·i	P22·e·i	–	00111 1	+
K.28	1	1	0	0	0	0	c' ·d' ·e' ·i'	P22·e' ·i'	+	00111 1	–

Table 5: **4b/3b Decoding Scheme**

Name	Received Data				Classification		Din	FGH K(3)	Dout
	f	g	h	j	Decoding Class(1,2)	Disparity(1,2)			
D/K.x.0	0	1	0	0	f' ·h' ·j'	f' ·h' ·j'	+	000 x	–
D/K.x.0	1	0	1	1	f·h·j	f·h·j	–	000 x	+
D/K.x.1	1	0	0	1			x	100 x	0
K/28.1	0	1	1	0	c' ·d' ·e' ·i' ·(h≠j)			100 1	0
D/K.x.6	0	1	1	0			x	011 x	0
K.28.6	1	0	0	1	c' ·d' ·e' ·i' ·(h≠j)			011 1	0
D.x.7	1	1	1	0		f·g·h	–	111 0	+
D.x.7	0	0	0	1	f' ·g' ·h'	f' ·g' ·h'	+	111 0	–
D/K.x.7	0	1	1	1	g·h·j	g·h·j	–	111 x	+
D/K.x.7	1	0	0	0	g' ·h' ·j'	g' ·h' ·j'	+	111 x	–

Notes:

1. The (·) represents the AND function.
2. The (') represents negation, for example, j' represents the negative assertion of bit j.
3. K = (c=d=e=i) or (P13·e'·i·g·h·j) or (P31·e·i'·g'·h'·j')

Error Detection

Error detection for 8b/10b data transmission is done in two manners. The first method of error checking uses redundancy checks on each packet transmission. For checking the validity of each data packet, this is the responsibility of a higher level network interface. This interface at the OSI transport layer manages error control, connection management, and flow control. The network interface verifies that each packet sent starts and ends with a delimiter that contains at least one nonzero-disparity subblock (either positive or negative disparity).

The second method, cyclic redundancy checking, evaluates the validity of data received. The following five checks are required for error checking on the transmission of each data byte.

- All 6b and 4b subblocks of a packet must have either positive, negative, or zero disparity (difference between "1s" and "0s")
- The disparity out of nonzero disparity blocks must alternate in polarity (positive and negative)
- All data bytes must follow the disparity rules
- The following conditions also apply to coding rules and are attributed to errors:

a=b=c=d	i≠e=g=h=j
P13·e' ·i'	(e=i≠g=h=j)·(c=d=e)'
P31·e' ·i	P31'·e·i' ·g' ·h' ·j'
f=g=h=j	P13' ·e' ·i·g·h·j
e=i=f=g=h	

Note: Data characters **abcdei fghj** apply to transmitted data characters. P13 and P31 are the preliminary disparity functions used when decoding.

Operational Flows

Figure 3 shows the high-level main operational flow for encoding data within the CoolRunner 16b/20b encoder module. For Figure 3, determining the bit encoding and disparity out is discussed in "8b/10b Design" on page 10.

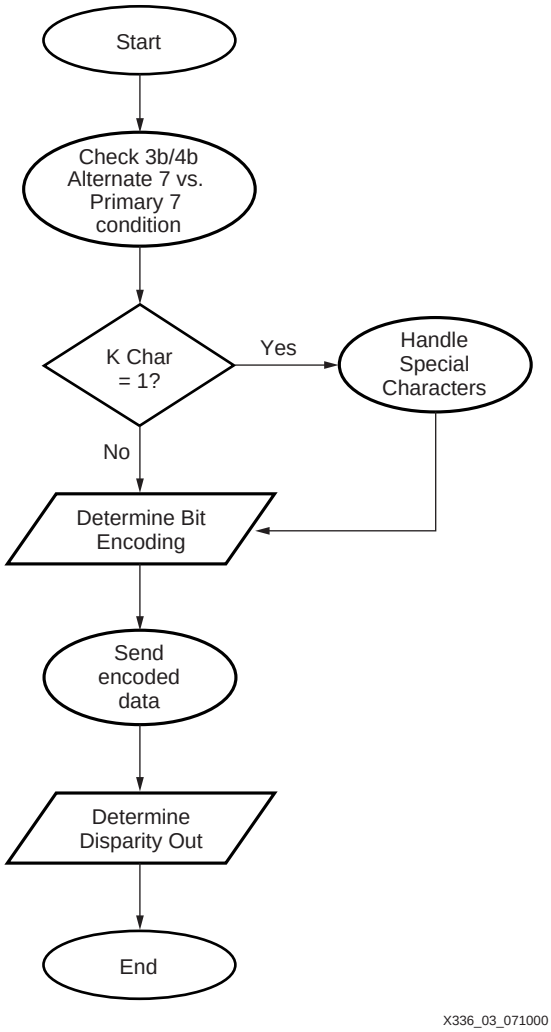

Figure 3: **Encoding Flow Diagram**

Figure 4 shows the high-level main operation flow for the CoolRunner 20b/16b decoder module. This module decodes the 20-bit incoming serial stream of data into a 16-bit parallel word, detecting errors for each byte sent. This module is only responsible for detecting errors of

each data byte and it is the responsibility of the higher level protocol to handle the retransmission of incorrect data packets.

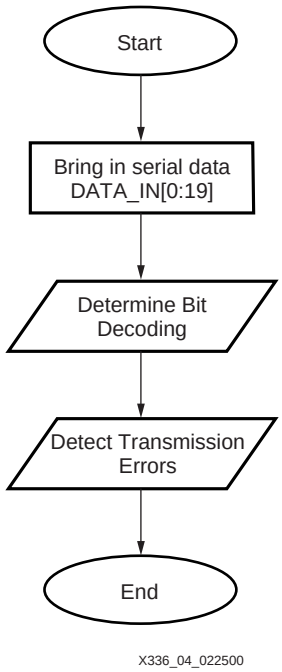

Figure 4: **Decoding Flow Diagram**

8b/10b Design

5b/6b Encoding

For encoding the lower parallel bits, **ABCDE**, two stages are involved as described in Figure 2. The first stage creates the bit encoding and disparity as described in Table 1 on page 4 and the second stage encodes the data to transmit serially. Figure 5 below shows the generation of the bit encoding functions for the 5b/6b encoder. As described in Table 1, the encoding functions L40, L04, L13, L31, and L22, represent the number of "1s" and "0s" in the **ABCD** data bits. Each input is represented by its assertion (+) or negation (–). Also note, (·) represents the AND function.

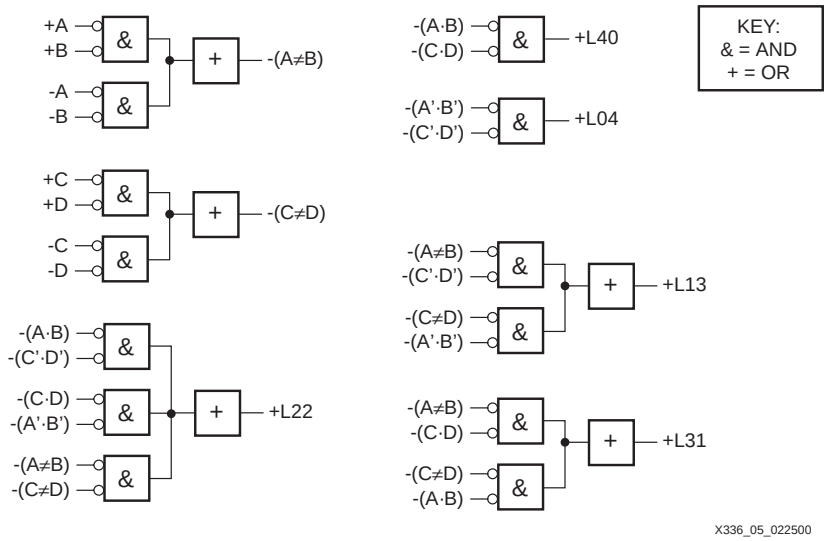

Figure 5: **5b/6b Bit Encoding Functions**

The encoded data **abcdei,** is created as shown in Figure 6. The encoding scheme generation uses the 5b/6b function outputs (i.e., L13, etc.) described in Figure 5.

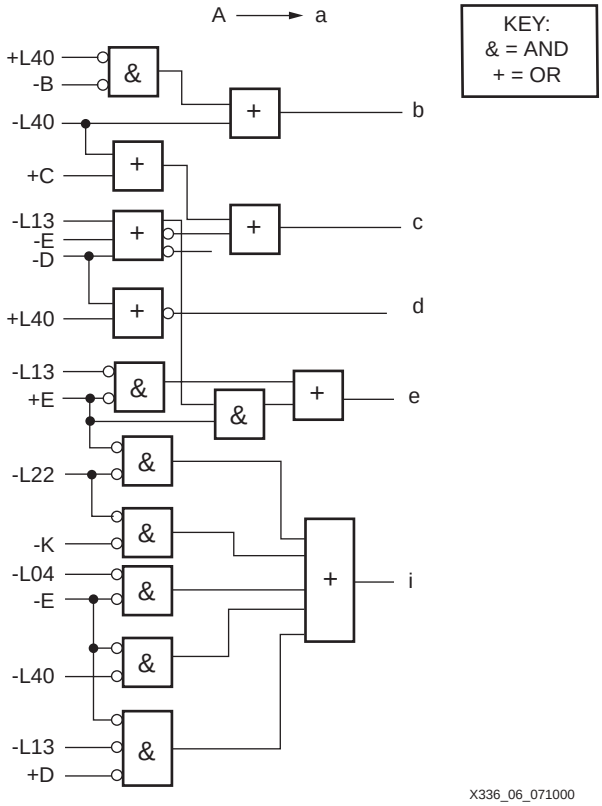

Figure 6: **5b/6b Encoding Generation**

3b/4b Encoding

The 3b/4b encoding mechanism is similar to that of 5b/6b. The 3b/4b encoding includes the same two stages, generating the bit encoding functions and creating the encoded data (see Figure 2). This is shown below in Figure 7 and 8 and includes generation of the S control term as discussed in Table 2. The inputs for generating the S control term include the running disparity functions for each 8b/10b module, PD–1S6 and ND–1S6. These control signals

represent the assertion and negation of the running disparity from the 5b/6b module (see Figure 9). Figure 8 shows the generation of the encoded output data, **fghj**.

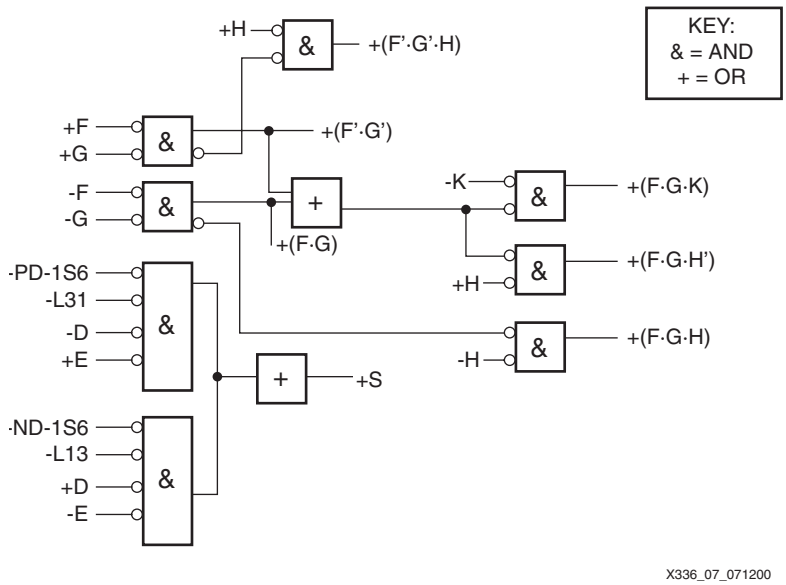

Figure 7: **3b/4b Bit Encoding Functions**

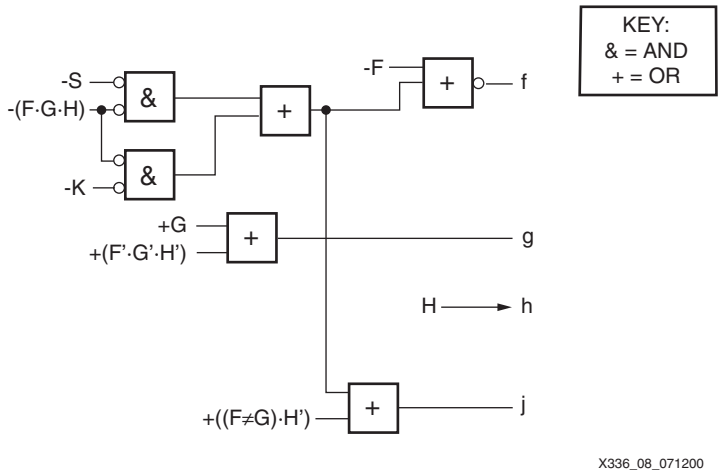

Figure 8: **3b/4b Encoding Scheme Generation**

Disparity Control

The disparity for each 8b/10b block is diagrammed in Figure 9. The disparity controller determines the positive and negative disparity for both the incoming data and data out of each 6b and 4b module. In this diagram, P=positive, N=negative, and S4 or S6 denote each 5b/6b or 3b/4b module respectively. For example when PD-1S6 is asserted, the disparity of the data being encoded (D–1 or running disparity) is positive. See Table 1 on page 4 for the 5b/6b

encoding characteristics. Similarly, ND0S4 when asserted, denotes a negative disparity out (Dout) for the data to be transmitted by the 3b/4b module.

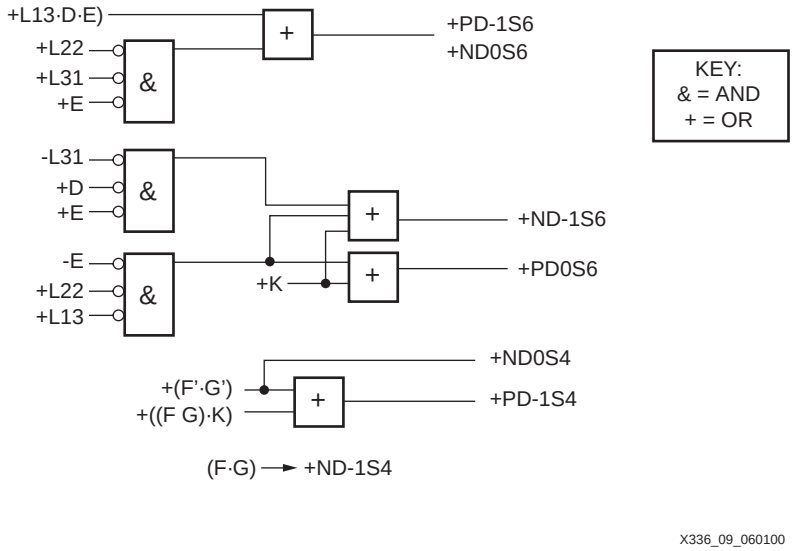

Figure 9: **Disparity Classification**

6b/5b and 4b/3b Decoding

The 6b/5b and 4b/3b decoding schemes are similar to the 5b/6b and 3b/4b encoding rules described previously. Based on the decoding class and disparity class functions for the incoming data described in Table 4 and Table 5 on page 7, each data bit out can be determined.

16b/20b Encoder VHDL Implementation

The CoolRunner CPLD implementation of the 16b/20b encoder/decoder module is done in two separate units, encoder logic and decoder logic. Figure 10 shows the block diagram for the VHDL implementation of the 16b/20b encoder module. An 8b/10b encoder is used for each upper and lower byte of incoming data. The disparity out of the lower byte of data is the incoming disparity of the upper byte and vice-versa, to balance the disparity between both bytes of transmitted data. The main logic for each 8b/10b module is managed by MAIN CONTROL. The encoding function, ENC_FUNC, is broken up for each 5b/6b and 3b/4b encoding schemes. The disparity generation, DIS_GEN, determines the disparity out for each block based on inputs **ABCDE FGH**, **K** and DIS_IN. The function S_GEN, determines the S control signal needed for the encoding function. The signal FRAME_IN is asserted when

DATA_IN is stable by the protocol device, and FRAME_OUT is asserted when the encoding is complete and ENCODED_DATA is ready to be clocked out serially.

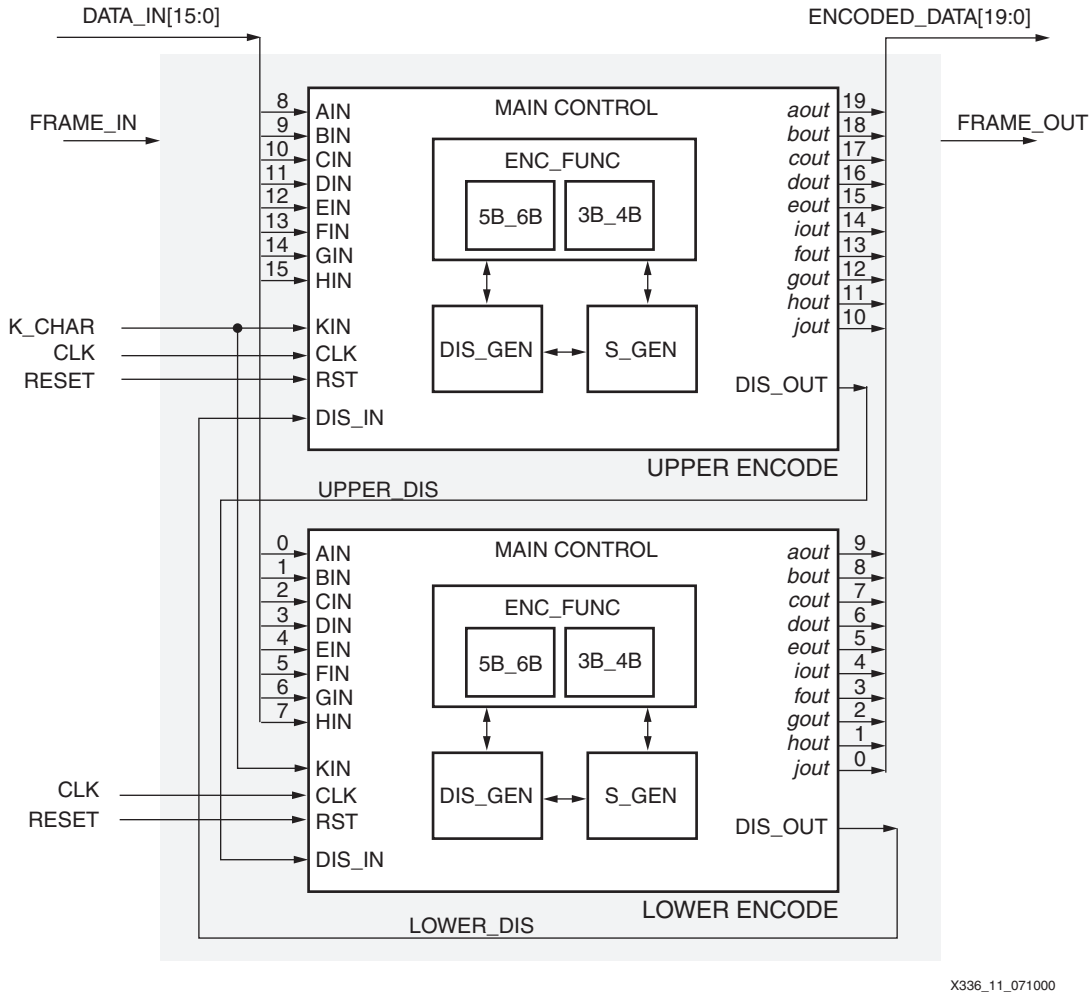

Figure 10: **Encoder Logic Block Diagram**

Encoding Main Control Logic

Figure 11 shows the main control block diagram for the 8b/10b encoder identifying all the interfaces between the state machines and control signals used between modules.

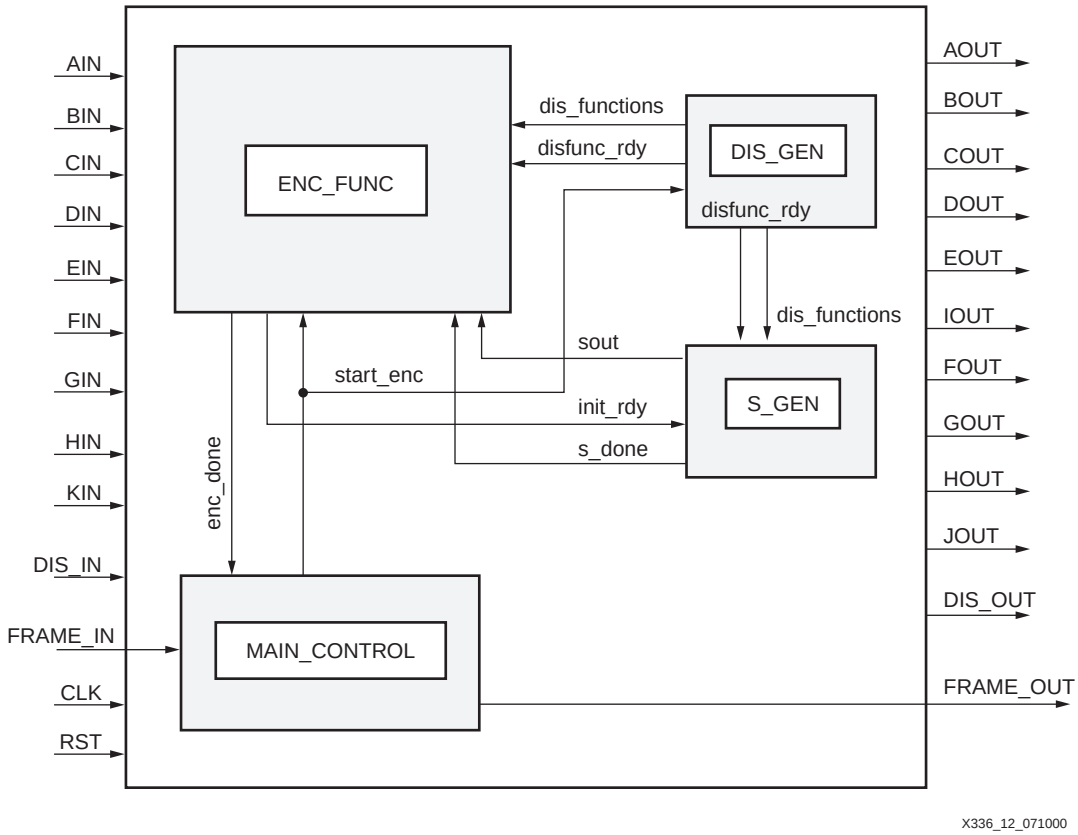

Figure 11: **Main Control Logic Block Diagram**

Figure 12 shows the main control state machine for the 8b/10b encoder module. The IDLE state waits for the FRAME_IN signal to be asserted, assuming the protocol device has waited for the data in to stabilize. The ASSIGN state makes signal assignments for the incoming byte of data, DATA_IN[0:7], to variables **AIN..HIN**. This state also initializes each of the three state machines within each 8b/10b encoding module, with the start_enc control signal. When the enc_func state machine is finished it will assert enc_done and the main control logic will then advance to DONE and asserts frame_out. This state will also present the output variables **AOUT..JOUT** to ENCODED_DATA[0:9], to allow the next parallel byte to be brought in the system while the output bits are clocked out serially.

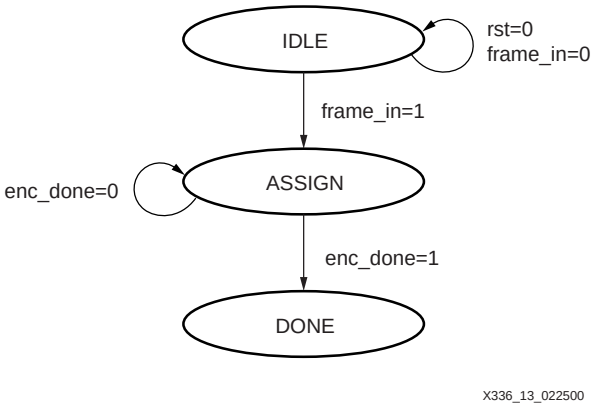

Figure 12: **Main Encode State Machine**

Encode Function Control Logic

Figure 13 shows the main logic for controlling each byte of incoming data, AIN..HIN, to 10 encoded serial bits through the 8b/10b encoding rules. Once the start_enc control signal is asserted, each 5b/6b and 3b/4b encoding function is activated. The ENCODE state generates the control signals described in Figure 5 and 7 respectively. The S_INPUT state asserts init_rdy to the S_GEN logic that pos_l31 and pos_l13 have been assigned. The S_INPUT state then waits for the sout control term to be generated.The ASSIGN state assigns the outputs and once the data is encoded and ready to be sent the enc_done signal is asserted for the main control logic to proceed to its DONE state.

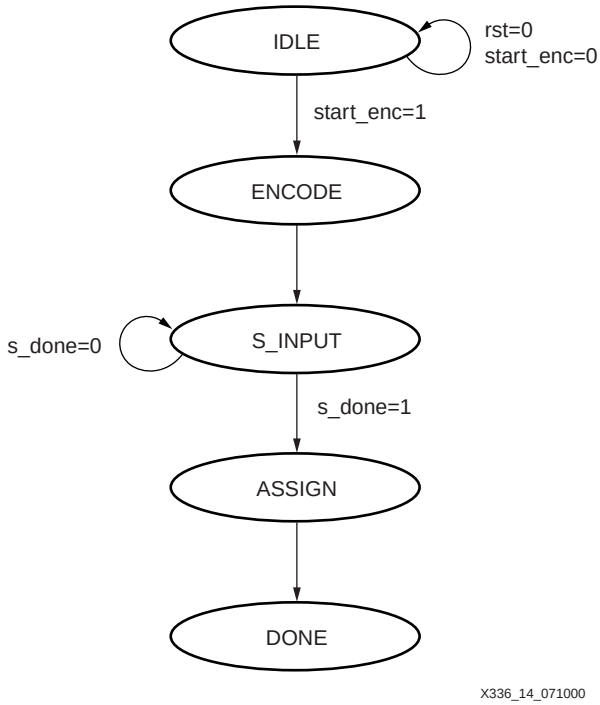

Figure 13: **Encoding Function State Machine**

Disparity Generation Control Logic

Upper Encoder Disparity

Figure 14 shows the main control logic for generating the disparity out, DIS_OUT, of the upper encoder module. The disparity of the upper module is fed back into the lower encoder as the disparity input. The DIS_OUT for each 16b/20b module is the running disparity out of the lower encoder module. The disparity generation state machine is activated once the start_enc control signal is asserted from the main control logic. The IDLE state initializes the DIS_OUT signal to "0". The next state, DIS_FUNC, creates the needed disparity control functions described earlier in Figure 9 and asserts the disfunc_rdy signal for the S generate state machine. The DIS_ASGN state combines the disparity functions created in the DIS_FUNC state (both negative and positive for each 5b/6b and 3b/4b blocks) and determines the running disparity

out of the 8b/10b module, DIS_OUT. The DIS_ASGN state asserts disout_rdy to the lower encoder module.

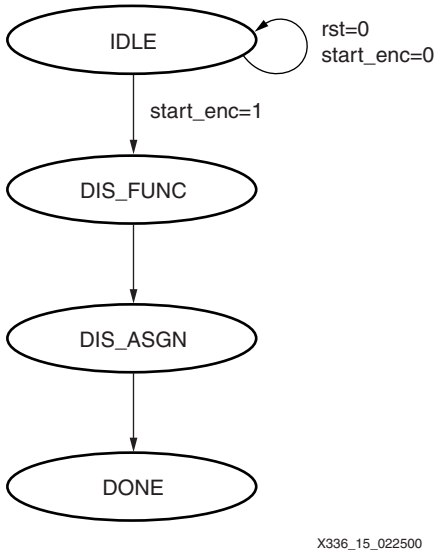

Figure 14: **Upper Disparity State Machine**

Lower Encoder Disparity

The logic for generating the disparity out in the lower encoder module is identical to the upper encoder module. The only exception being the lower encoder must wait for the assertion of disout_rdy from the upper encoder module as shown in Figure 15.

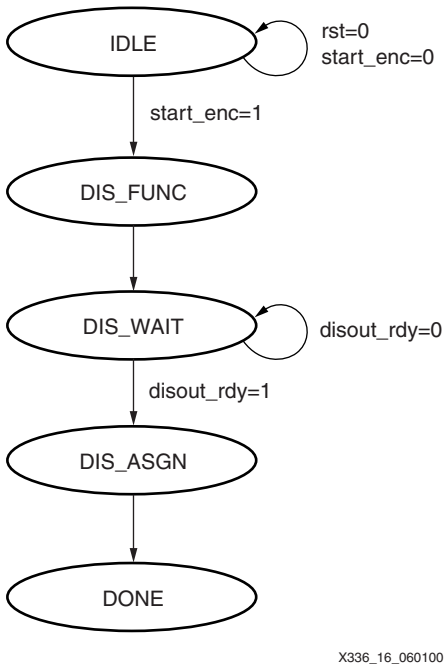

Figure 15: **Lower Disparity State Machine**

S Function Generation

The logic for the generation of the S control term is shown in Figure 16. This control term is generated after the IDLE state when the encoder function (asserted init_rdy) and disparity function (asserted disfunc_rdy) are assigned. The input functions for S are used as described in Figure 7 on page 12 in the ASSIGN_S state. The s_done signal is then asserted to represent the assignment of sout is complete.

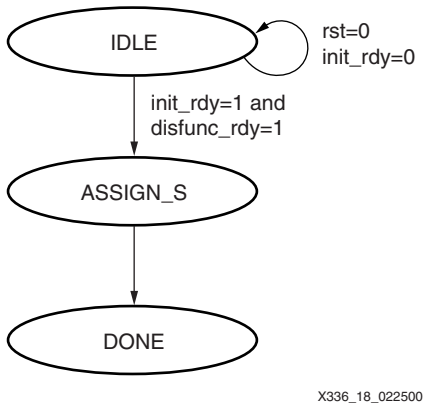

Figure 16: **S Control Term State Machine**

20b/16b Decoder VHDL Implementation

Figure 17 shows the main block diagram for the VHDL implementation of the 20b/16b decoder module. A 10b/8b decoder block is used for each upper and lower byte of incoming serial data. Once the ENCODED_DATA is read into the module, the decoding function, DEC_FUNC, determines the outputs **AOUT..HOUT** based on the serial inputs **ain..jin**. The error checking for each 10b/8b decoder, ERR_CHK, asserts the ERROR flag for each 10b/8b module. The overall error detecting module, ERROR_DETECT, recognizes the difference between special characters and errors that occur in transmission and asserts the illegal character detect flag, ILL_CHAR_DET. The 20b/16b decoder module also incorporates the FRAME_IN and FRAME_OUT control as the 16b/20b encoder module. FRAME_IN is asserted by the external controller when the ENCODED_DATA is presented to the decoder, and FRAME_OUT is asserted by the decoder when done decoding and asserting the appropriate error flags. This

20b/16b decoder does not take any action based on the detection of an illegal character, this is the responsibility of the higher-level protocol device.

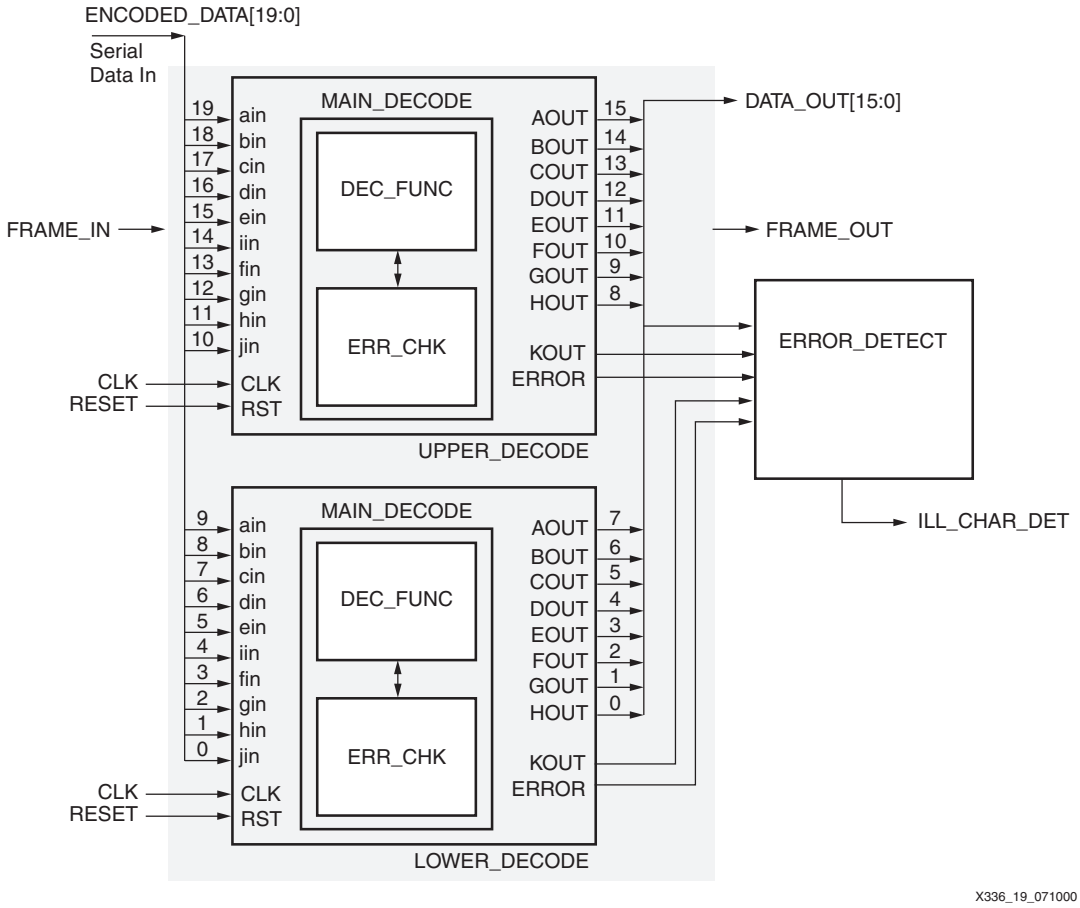

Figure 17: **Decoder Main Control Logic Block Diagram**

Main Decode Control Logic

The block diagram for the main decode logic is shown in Figure 18. It shows the main control interface between the decoding function, DEC_FUNC and the error checking module, ERR_CHK.

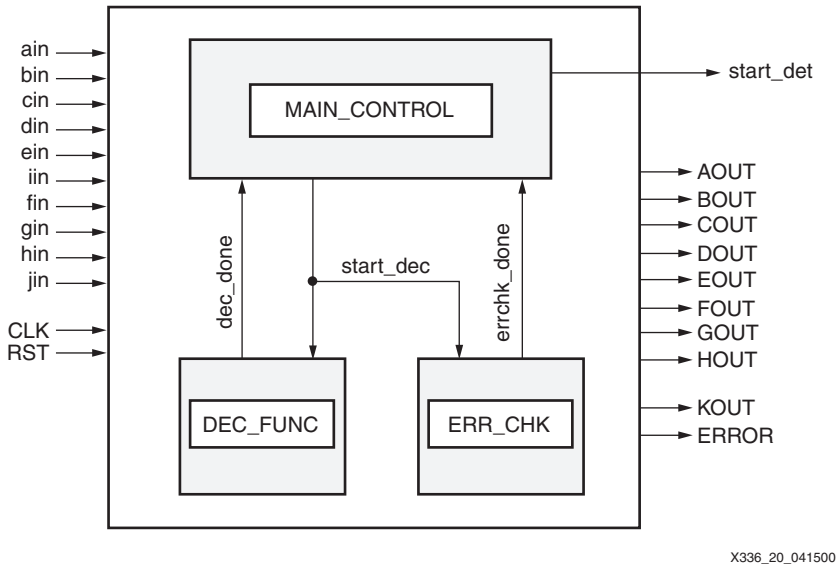

Figure 18: **8b/10b Decode Block Diagram**

The main decoder control logic is shown in Figure 19. The IDLE state waits for FRAME_IN to be asserted by the higher level control that the ENCODED_DATA[19:0] is stable and ready for the decoder module. The ASSIGN state assigns the incoming data bits, ENCODED_DATA[19:0] to each ain through jin for each 10b/8b module. This state asserts start_dec to start the DEC_FUNC and ERR_CHK state machines. Once these state machines are done, i.e., dec_done and errchk_done are asserted, the state machine then transitions to DONE. The DONE state asserts start_det, to start the ERROR_DETECT state machine as well as assert FRAME_OUT.

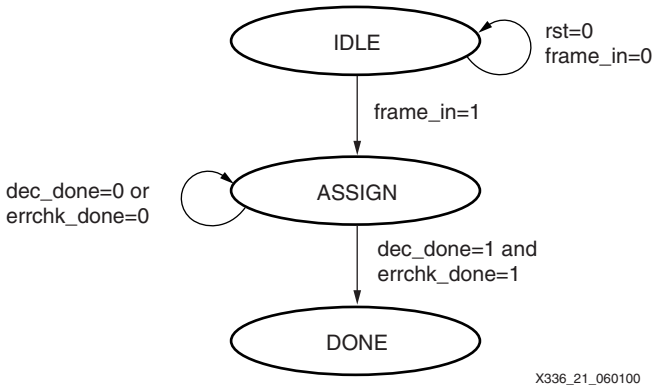

Figure 19: **Decoder Main Control Logic**

Decode Function Control Logic

The decoder functional control logic is shown in Figure 20. The assertion of start_dec transitions the state machine from the IDLE state to the PREL state. The PREL state creates the preliminary decoding signals from the inputs ain through jin. In the DECODE state, the decode functions are used in generating AOUT.. KOUT. To generate AOUT.. HOUT and KOUT, the 10b/8b decoding rules are implemented. The DONE state asserts the dec_done flag to the main control logic.

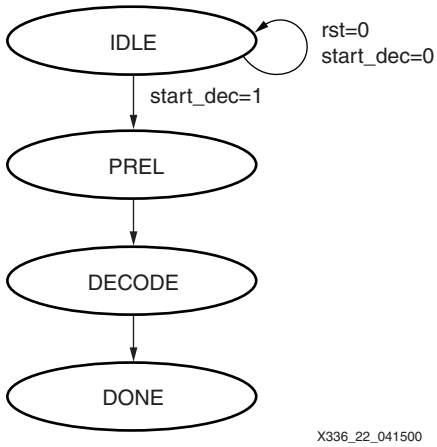

Figure 20: **Decoder Function State Machine**

Error Checking Control Logic

The error checking control logic, ERR_CHK, in each 10b/8b module, determines when an error has occurred and when special characters are transmitted. The assertion of the ERROR flag does not categorize errors from special characters. This is done in the ERROR_DETECT control logic. This module detects decoding errors such as too long of run length or violating disparity requirements. The ERROR flag is asserted in the ERROR_CHK state, and the transition to the DONE state asserts the done_errdet flag. The state machine is shown in Figure 21.

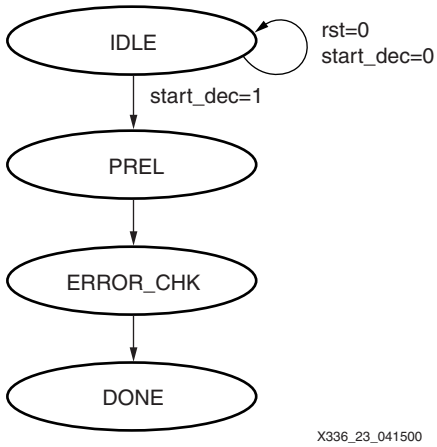

Figure 21: **Error Check State Machine**

Error Detection Control Logic

Figure 22 shows the main block diagram for the error detection module, ERROR_DETECT of the 20b/16b decoder. The inputs are the KOUT characters and ERROR flags from each upper and lower 10b/8b decoder modules. This ERROR_DETECT module recognizes the differences between special characters and actual errors in each data frame. The ILL_CHAR_DET signal is asserted for the higher level control when indeed an error has been detected and an illegal character has been sent.

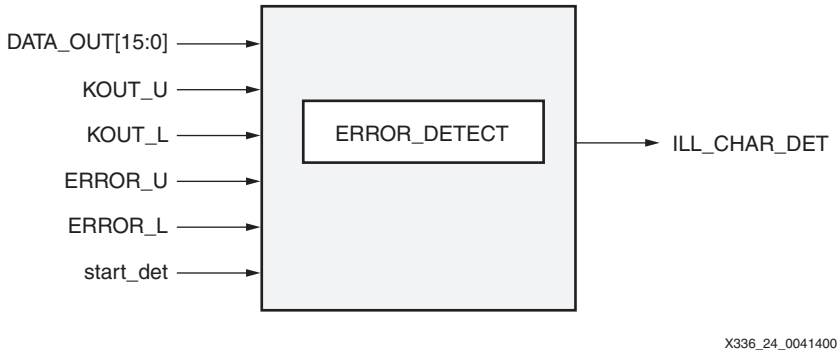

Figure 22: **Error Detect Block Diagram**

Figure 23 shows the main control logic for the error detection module. The start_det signal transitions the state machine from the IDLE state to DETECT, where the output function for the ILL_CHAR_DET signal is generated. Once ILL_CHAR_DET has been determined the state machine then transitions to the DONE state.

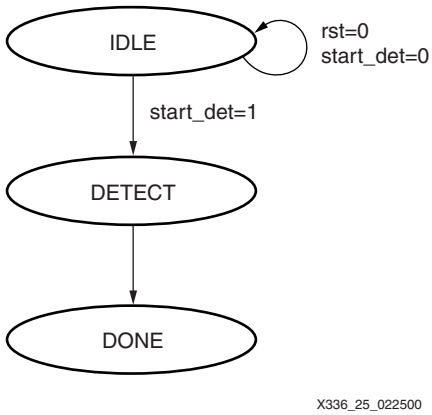

Figure 23: **Error Detection State Machine**

CoolRunner XPLA3 CPLD Implementation

Each 16b/20b encoder and decoder design was implemented in VHDL as described above. Xilinx Project Navigator was used for compilation, fitting, and simulation of the design in a CoolRunner CPLD. Xilinx Project Navigator, which includes the ModelTech simulation tool, is available free-of-charge from the Xilinx website at www.xilinx.com/products/software/webpowered.htm. Each design was targeted separately for a 3.3V, 128 macrocell CoolRunner XPLA3 CPLD (XCR3128XL-10VQ100C). The 16b/20b encoder and decoder utilization is shown in Table 6 and Table 7. This utilization was achieved

using certain fitting parameters, actual results may vary. As shown, there is area remaining in the CPLD for the implementation of other logic in the system.

Table 6: **16b/20b Encoder XPLA3 128-Macrocell Utilization**

Resource	Available	Used	Utilization
Logic Blocks	8	5	62.50%
Macrocells	128	66	51.56%
Product Terms	384	194	50.52%
Foldback Nands	64	13	20.31%
I/O Pins	80	42	52.50%

Table 7: **20b/16b Decoder XPLA3 128-Macrocell Utilization**

Resource	Available	Used	Utilization
Logic Blocks	8	7	87.5%
Macrocells	128	63	49.2%
Product Terms	384	292	76.0%
Foldback Nands	64	8	12.5%
I/O Pins	80	41	51.25%

Design Verification

The encoder and decoder VHDL design verification has been done through simulation using ModelSim XE in Project Navigator. The design has been verified both functionally and with the timing model generated when fitting in a CoolRunner CPLD. A test bench was used as shown in Figure 24, which keeps the data pipeline full when sending to the encoder module and checking data coming from the decoder module. The test bench drove the control, data, and timing needed for each encoder and decoder module. The control in this design is simply a model for understanding the handshaking when using a 16b/20b transmission scheme. Implementation in an actual system may require modification of the control signals used in the source code and test benches provided.

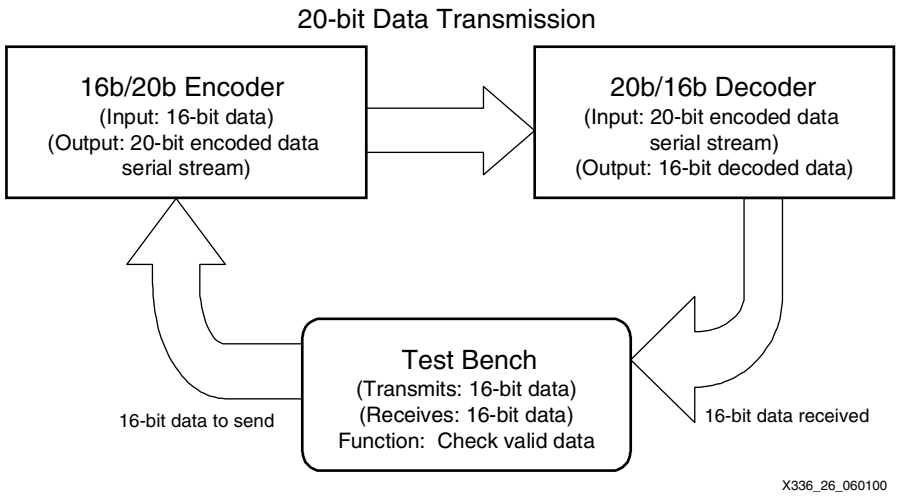

Figure 24: **Encoder and Decoder Testing**

ModelSim Implementation

Note:

Please refer to XAPP338: Using Xilinx WebPack and ModelTech ModelSim Xilinx Edition as a guide to using ModelSim with Project Navigator. The ModelSim Quick Start demo provides a good first step for getting familiar with ModelSim.

The test environment provided in this design instantiates both the encoder and decoder module and monitors the transfer of data to/from each module. Figure 25 shows the output generated by each module in the transmission of the data pattern 00000000 01100011 (referenced as $UPPER_{A..H}$ and $LOWER_{A..H}$). The ENCODED_DATA[19:0] in this example is then transmitted serially as the pattern 1001110100 1011010011. Note the disparity into the encoder (based on the transmission of the previous data) when sending is negative (dis_in=1) and the running disparity out is positive (dis_out=0). The test bench provided with this design then waits a latency time to check the data out of the decoder. Note the data is referenced reversed within each lower and upper byte, with the MSB and LSB switched. See Figure 10 on page 14 and Figure 17 on page 19 for the data direction.

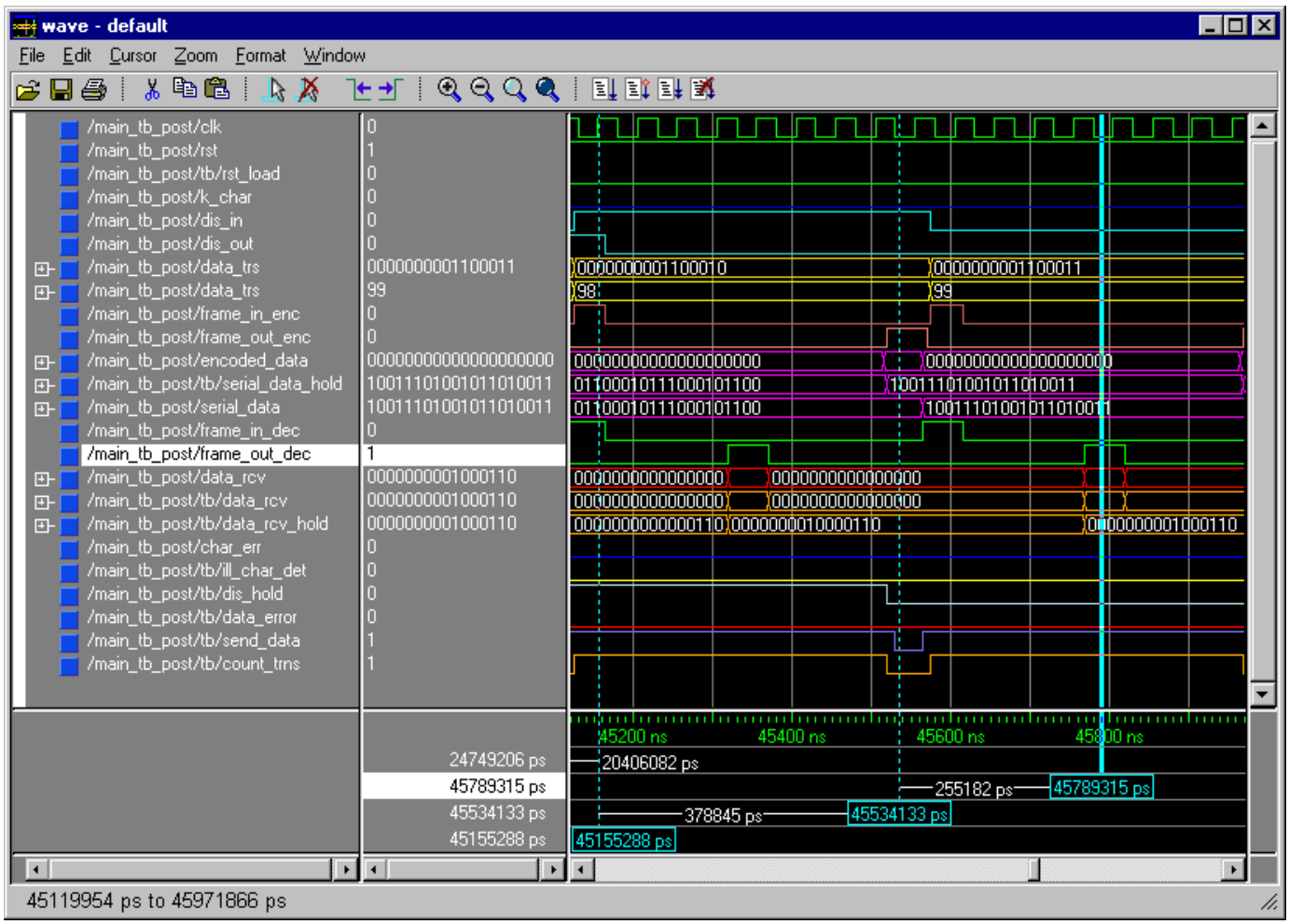

Figure 25: **ModelSim Test Bench Simulation**

VHDL Code Download

VHDL source code and test benches are available for this design. THE DESIGN IS PROVIDED TO YOU “AS IS”. XILINX MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. This design has not been verified on hardware (as opposed to simulations), and it should be used only as an example design, not as a fully functional core. XILINX does not warrant the performance, functionality, or operation of this Design will meet your requirements, or that the operation of the Design will be uninterrupted or error free, or that defects in the Design will be corrected. Furthermore, XILINX does not warrant or make any representations regarding use or the results of the use of the Design in terms of correctness, accuracy, reliability or otherwise.

XAPP336 - http://www.xilinx.com/products/xaw/coolvhdlq.htm

Conclusion

This document has detailed the design of two modules for 8b/10b communication protocol. The 16b/20b encoder design enables 16-bit data transmission serially over any fibre channel using the 8b/10b encoding rules. The 20b/16b decoder module decodes a 20-bit serially encoded stream of data into a 16-bit data word. Both of these designs demonstrate the ability to use CoolRunner CPLDs for communication protocols when low-power, high-performance, and reliable network devices are required. CoolRunner CPLDs are ideal for applications that are portable, handheld, or power-sensitive. CoolRunner CPLDs provide the lowest power consumption available today as well as high system reliability. More information about CoolRunner CPLDs can be found at http://www.xilinx.com/products/xpla3.htm

Glossary

The glossary, defined in Table 8 is intended to identify terms used in 8b/10b transmission.

Table 8: **Glossary**

ATM	Asynchronous Transmission Mode
D Character	Data character.
Decoding Class	Predetermined classification for transmitted data. The decoding class is determined based on the characters of the transmitted data. This is done by essentially counting the number of "1s" and "0s" in each data byte that was sent.
Disparity	Running parity is determined upon sending each byte of data in 8b/10b transmission. Logical value is positive (+) when more "1s" than "0s" are sent. Logical value is negative (–) when more "0s" than "1s" are transmitted. If the parity is neither positive or negative, hence an equal number of "1s" and "0s" transmitted, the logical parity value is zero (0). These values are shown for disparity in (Din) and disparity out (Dout) in Table 1 on page 4. Positive disparity translates to a "0" running disparity in the code. A negative disparity translates to a "1" running disparity in the code. When the parity is neither positive or negative, the running disparity remains unchanged.
Encoding Class	Predetermined classification for transmitting data. The encoding class is determined based on the characters of the data to be sent. This is done by essentially counting the number of "1s" and "0s" in each data byte to send.
K Character	Input parameter to control transmission of special characters. When K is asserted or equal to "1", a special character is encoded and transmitted. When K is equal to "0", data characters are transmitted. For more information on control characters, see "Special Characters" on page 5.

Table 8: **Glossary**

Lxy	Bit classification for encoding. Classification represents the four least significant bits being sent. "x" represents the number of "1s" in transmission, while "y" represents the number of "0s" being sent. For example, L13, is the bit encoding for sending one "1" and three "0s". See Table 1 on page 4 for more examples.
OSI	Open Systems Interconnection
Pxy	Bit classification for decoding. Classification represents the four least significant bits of data being decoded. "x" represents the number of "1s" that were sent, while "y" represents the number of "0s" that were sent. For example, P31, is the decoding classification that three "1s" and one "0" were sent. See Table 4 on page 6 for more examples.
Run length	Length of continuous "1s" or "0s" in an bit stream. The 8b/10b encoding mechanism never allows data to be transmitted with a run length of more than 5. For an example, see "Singular Commas" on page 6.
Special Characters	Special characters are characters used beyond the 8-bit, 256 character limitation. Special characters are denoted by the assertion of the K control character. These characters are used for things such as START, IDLE, RESET, STOP, and link diagnostics. For more information on special characters, see "Special Characters" on page 5.
Synchronization	Way to represent or arrange (events) to indicate coincidence or coexistence.

References

1. *Halsall, Fred, Data Communications, Computer Networks and Open Systems. Fourth Edition. Addison-Wesley Publishing Company. 1996.*
2. *Widmer, A.X. & Franaszek, P.A., A DC-Balanced, Partitioned-Block, 8b/10b Transmission Code. IBM J. Res. Develop., Vol. 27, No. 5. September 1983.*
3. *Gigabit Ethernet And Fibre Channel Technology. Optimized Engineering Technical Compendium Corporation. (http://www.optimized.com)*

Revision History

The following table shows the revision history for this document.

Date	Version #	Revision
07/15/00	1.0	Initial Release