XAPP645 (v2.2) August 9, 2006

Single Error Correction and Double Error Detection

Author: Simon Tam

Summary

This application note describes the implementation of an Error Correction Control (ECC) module in a Virtex™-II, Virtex-II Pro, Virtex-4, or Virtex-5 device. The design detects and corrects all single bit errors (in a codeword consisting of either 64-bit data and 8 parity bits, or 32-bit data and 7 parity bits), and it detects double bit errors in the data. This design utilizes Hamming code, a simple yet powerful method for ECC operations. As a result, this design offers exceptional performance and resource utilization.

Introduction

Error detection and correction is found in many high-reliability and performance applications. For example, in enterprise data storage systems, memory caches are utilized to improve system reliability. The cache is typically placed inside the controller between the host interfaces and the disk array. A robust cache memory design often includes ECC functions to avoid single point of failure losses of customer data. ECC becomes an important feature for many communication applications, such as satellite receivers; it is more performance and cost efficient to correct an error rather than retransmit the data.

The reference design (**XAPP645.zip**) described in this application note implements error detection and correction at the speed of the data read/write rate, up to 144 MHz unpipelined or 313 MHz pipelined in a Virtex-II Pro device with a -6 speed grade. It detects double bit errors and corrects single bit errors anywhere within the codeword. The reference design targets 72-bit double data rate (DDR) DIMM memory. Both 32-bit and pipelined versions are available. The design is easily modified to fit other narrower data widths.

Hamming Code

The ECC functions described in this application note are made possible by Hamming code, a relatively simple yet powerful ECC code. It involves transmitting data with multiple check bits (parity) and decoding the associated check bits when receiving data to detect errors.

The check bits are parallel parity bits generated from XORing certain bits in the original data word. If bit error(s) are introduced in the codeword, several check bits show parity errors after decoding the retrieved codeword. The combination of these check bit errors display the nature of the error. In addition, the position of any single bit error is identified from the check bits.

The Hamming codeword is a concatenation of the original data and the check bits (parity). It is described by an ordered set (d + p,d) where d is the width of the data and p is the width of the parity. The parity matrix [P] can be expressed as:

$$[P] = [D] \bullet [G]$$

where [D] is the data matrix and [G] is the generator matrix. The [G] matrix consists of an identity matrix [I] and a creation matrix [C].

$$[G] = [I:C]$$

For example, the (7,4) Hamming code is:

$$[G] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

The minimum number of check bits required for a single bit error correction is derived from the following equation:

$$D + P + 1 \leq 2^P$$

This reference design uses the (72,64) Hamming code. In other words, the Hamming codeword width is 72 bits, comprised of 64 data bits and eight check bits. The minimum number of check bits needed for correcting a single bit error in a 64-bit word is seven. The extra check bit expands the function to detect double bit errors as well.

To detect errors, the codeword vector multiplies with the transpose of the generator matrix to produce an 8-bit vector [S], known as the syndrome vector.

$$[S] = [D,P] \bullet [G']$$

If all of the elements of the syndrome vector are zeros, no error is reported. Any other non-zero result represents the bit error type and provides the location of any single bit errors. It is then used to correct the original incoming data.

To visualize Hamming code, consider the tables shown in the following figures. Each data bit position as well as the check bits are mapped in a syndrome table as shown in Figure 1. The location of each table cell is given by the row and column position. For example, data bit 60 is at column 100 and row 1000, or position 1000100. By calculating the parity of each position bit, either odd or even, seven check bits can be derived. The check bit equation is composed of XOR operators, denoted as a ⊕. For example, the logical equation for check bit 1(CB1):

CB1 = D0 ⊕ D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D8 ⊕ D10 ⊕ D11 ⊕ D13 ⊕ D15 ⊕ D17 ⊕ D19 ⊕ D21 ⊕ D23 ⊕ D25 ⊕ D26 ⊕ D28 ⊕ D30 ⊕ D32 ⊕ D34 ⊕ D36 ⊕ D38 ⊕ D40 ⊕ D42 ⊕ D44 ⊕ D46 ⊕ D48 ⊕ D50 ⊕ D52 ⊕ D54 ⊕ D56 ⊕ D57 ⊕ D59 ⊕ D61 ⊕ D63

Essentially, all data bits with the table cell position of 1 as the least significant bit are chosen to create CB1 (see Figure 1). CB2 involves the second to the least significant bit and so forth.

111	110	101	100	011	010	001	000	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
D56	D55	D54	D53	D52	D51	D50	D49	0111
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	D28	D27	D26	*CB6*	0100
D25	D24	D23	D22	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	0000

x645_01_022103

Figure 1: **Syndrome Table**

When there is no bit error, as shown in Figure 2, the check bits match with the calculated check bits of the data. As a result, all syndrome bits are zero, pointing to the no error position.

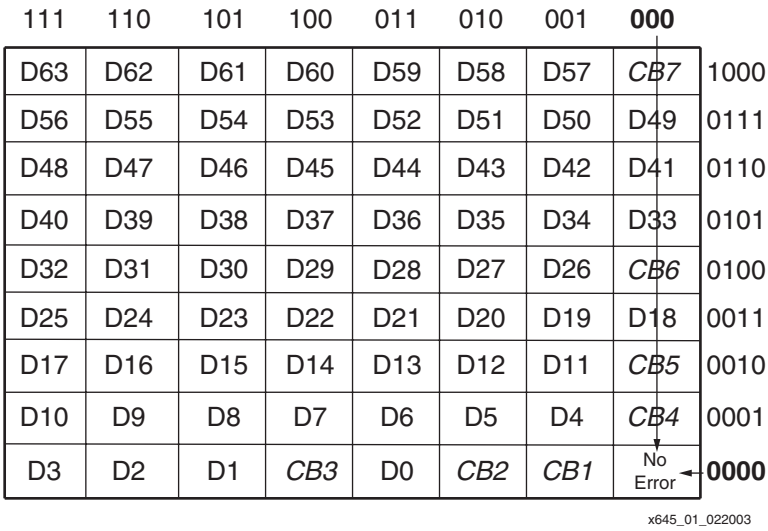


Figure 2: **No Bit Error Detection**

If a single bit error occurs, as shown in Figure 3, several syndromes have odd parity (resulting in a logic one) where the column and row position can be determined. If D28 is incorrect, then CB1, CB2, and CB6 have parity errors. As a result, D28 is identified as the error bit in the syndrome table.

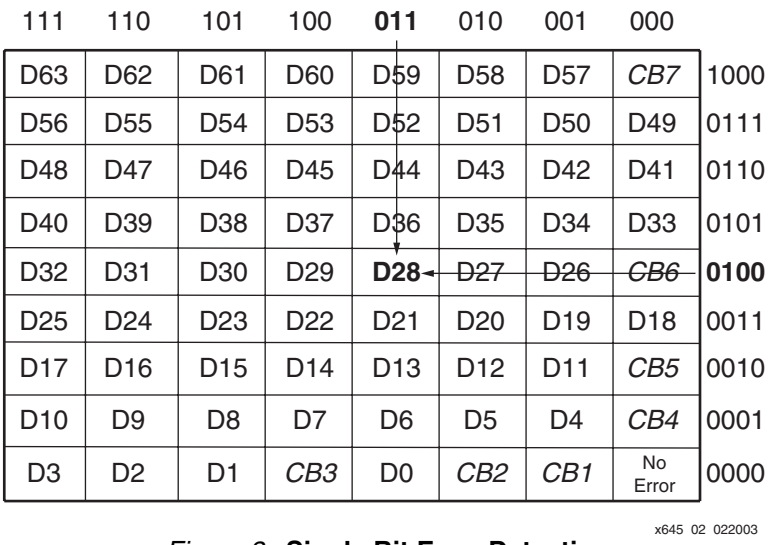


Figure 3: **Single Bit Error Detection**

If a double bit error occurs, as shown in Figure 4, the error bit positions are either not pinpointed or pinpointed incorrectly. For example, if a double bit error occurred at D28 and D22, the resulting syndrome points to column 111 and row 0111. However, it is still possible to detect a double bit error event by adding one additional check bit (CB8) that covers every data bit. If CB1 through CB7 results in a non-zero value while CB8 returns a zero, then a double bit error has occurred.

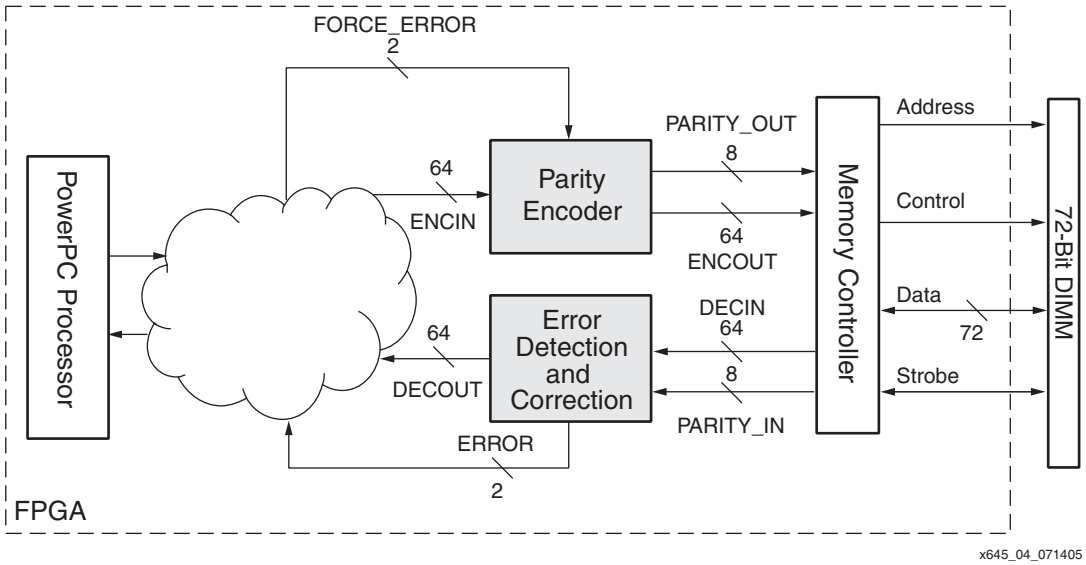
111	110	101	100	011	010	001	000	
D63	D62	D61	D60	D59	D58	D57	*CB7*	1000
~~D56~~	~~D55~~	~~D54~~	~~D53~~	~~D52~~	~~D51~~	~~D50~~	~~D49~~	**0111**
D48	D47	D46	D45	D44	D43	D42	D41	0110
D40	D39	D38	D37	D36	D35	D34	D33	0101
D32	D31	D30	D29	**D28**	D27	D26	*CB6*	0100
D25	D24	D23	**D22**	D21	D20	D19	D18	0011
D17	D16	D15	D14	D13	D12	D11	*CB5*	0010
D10	D9	D8	D7	D6	D5	D4	*CB4*	0001
D3	D2	D1	*CB3*	D0	*CB2*	*CB1*	No Error	0000

x645_03_022003

Figure 4: **Double Bit Error Detection**

Design Overview

Figure 5 shows a block diagram using a DDR memory controller with ECC functions. The DDR DIMM in this example is a Micron MT18VDDT6472G, an ECC configuration module. The reference design has a parity encoder and parity decoder unit. The encoder implements the function of the generator matrix, while the decoder is responsible for error detection and correction. Additionally, the diagnostic functions are supported. These functions are described in the following sections.



x645_04_071405

Figure 5: **ECC in a Memory System**

Parity Encoder

The encoder consists of XORs and a bit-error generator implemented in look-up tables (LUTs). An optional pipeline stage can be added to improve performance further. Figure 6 shows a block diagram of the parity encoder.

The check bits are written in the memory along with the associated 64-bit data. During a memory read, the data and the check bits are read simultaneously. Any error(s) introduced during the read/write access between the FPGA and memory are detected.

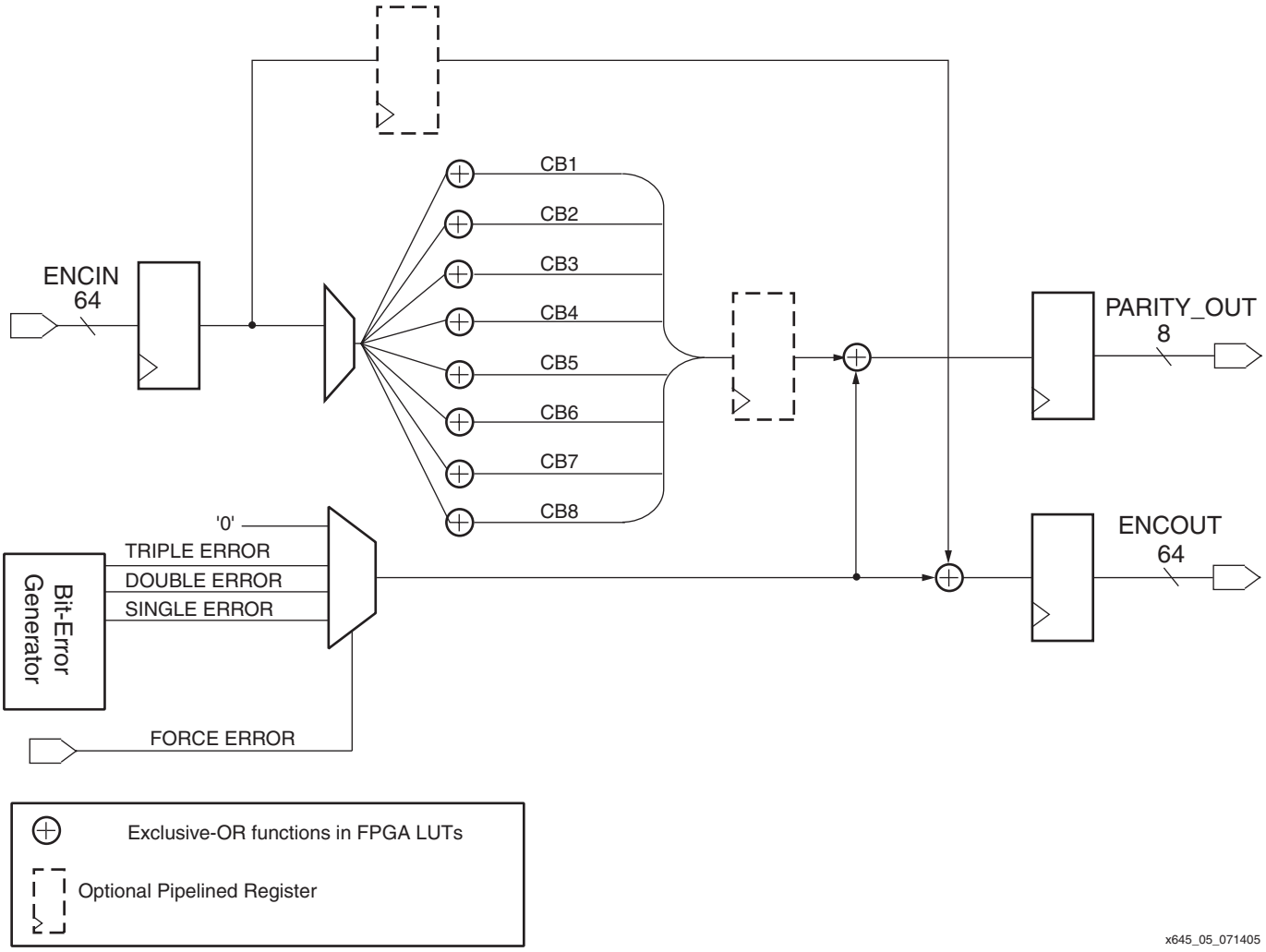


Figure 6: **Parity Encoder Block Diagram**

The parity bits are generated based on an unmodified Hamming code. Table 1 shows the participating bits in the generation of the (72,64) codeword. Table 2 shows the participating bits in the generation of the (39,32) codeword.

Table 1: **64-Bit Hamming Code**

Participating Data Bits	Generated Check Bits							
	CB1	CB2	CB3	CB4	CB5	CB6	CB7	CB8
0	√	√						√
1	√		√					√
2		√	√					√
3	√	√	√					√
4	√			√				√
5		√		√				√
6	√	√		√				√
7			√	√				√
8	√		√	√				√
9		√	√	√				√
10	√	√	√	√				√
11	√				√			√
12		√			√			√
13	√	√			√			√
14			√		√			√
15	√		√		√			√
16		√	√		√			√
17	√	√	√		√			√
18				√	√			√
19	√			√	√			√
20		√		√	√			√
21	√	√		√	√			√
22			√	√	√			√
23	√		√	√	√			√
24		√	√	√	√			√
25	√	√	√	√	√			√
26	√					√		√
27		√				√		√
28	√	√				√		√
29			√			√		√
30	√		√			√		√
31		√	√			√		√
32	√	√	√			√		√
33				√		√		√
34	√			√		√		√

Table 1: **64-Bit Hamming Code *(Continued)***

Participating Data Bits	Generated Check Bits							
	CB1	CB2	CB3	CB4	CB5	CB6	CB7	CB8
35		√		√		√		√
36	√	√		√		√		√
37			√	√		√		√
38	√		√	√		√		√
39		√	√	√		√		√
40	√	√	√	√		√		√
41					√	√		√
42	√				√	√		√
43		√			√	√		√
44	√	√			√	√		√
45			√		√	√		√
46	√		√		√	√		√
47		√	√		√	√		√
48	√	√	√		√	√		√
49				√	√	√		√
50	√			√	√	√		√
51		√		√	√	√		√
52	√	√		√	√	√		√
53			√	√	√	√		√
54	√		√	√	√	√		√
55		√	√	√	√	√		√
56	√	√	√	√	√	√		√
57	√						√	√
58		√					√	√
59	√	√					√	√
60			√				√	√
61	√		√				√	√
62		√	√				√	√
63	√	√	√				√	√

Table 2: **32-Bit Hamming Code**

Participating Data Bits	Generated Check Bits						
	CB0	CB1	CB2	CB3	CB4	CB5	CB6
0	√	√					√
1	√		√				√
2		√	√				√
3	√	√	√				√
4	√			√			√
5		√		√			√
6	√	√		√			√
7			√	√			√
8	√		√	√			√
9		√	√	√			√
10	√	√	√	√			√
11	√				√		√
12		√			√		√
13	√	√			√		√
14			√		√		√
15	√		√		√		√
16		√	√		√		√
17	√	√	√		√		√
18				√	√		√
19	√			√	√		√
20		√		√	√		√
21	√	√		√	√		√
22			√	√	√		√
23	√		√	√	√		√
24		√	√	√	√		√
25	√	√	√	√	√		√
26	√					√	√
27		√				√	√
28	√	√				√	√
29			√			√	√
30	√		√			√	√
31		√	√			√	√

Parity Decoder

The decoder unit shown in Figure 7 consists of three blocks:

- Syndrome generation
- Syndrome LUT and mask generation
- Data correction

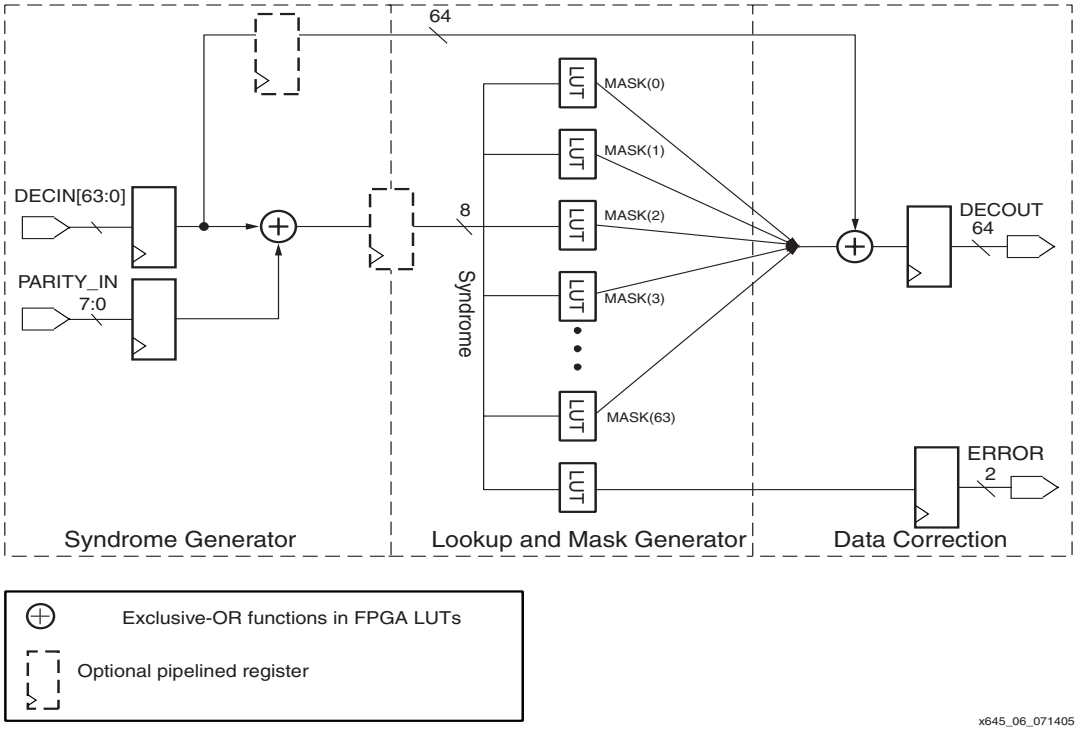


Figure 7: **ECC Functional Block Diagram**

Syndrome Generation

The incoming 64-bit data along with the 8-bit parity are XOR'd together to generate the 8-bit syndrome (S1 through S8). This is very similar to check bit generation, for example:

S1 = DECIN0 ⊕ DECIN1 ⊕ DECIN3 ⊕ DECIN4 ⊕ DECIN6 ⊕ DECIN8 ⊕ DECIN10 ⊕ DECIN11 ⊕ DECIN13 ⊕ DECIN15 ⊕ DECIN17 ⊕ DECIN19 ⊕ DECIN21 ⊕ DECIN23 ⊕ DECIN25 ⊕ DECIN26 ⊕ DECIN28 ⊕ DECIN30 ⊕ DECIN32 ⊕ DECIN34 ⊕ DECIN36 ⊕ DECIN38 ⊕ DECIN40 ⊕ DECIN42 ⊕ DECIN44 ⊕ DECIN46 ⊕ DECIN48 ⊕ DECIN50 ⊕ DECIN52 ⊕ DECIN54 ⊕ DECIN56 ⊕ DECIN57 ⊕ DECIN59 ⊕ DECIN61 ⊕ DECIN63 ⊕ PARITY_IN(1)

Then the next stage uses the syndrome to look for the error type and the error location. An optional pipeline stage can be added here to improve performance further.

Syndrome LUT and Mask Generation

In order to correct a single bit error, a 64-bit correction mask is created. Each bit of this mask is generated based on the result of the syndrome from previous stage. When no error is detected, all bits of the mask become zero. When a single bit error is detected, the corresponding mask masks out the rest of the bits except for the error bit. The subsequent stage then XORs the mask with the original data. As a result, the error bit is reversed (or corrected) to the correct state. If a double bit error is detected, all mask bits become zero. The error type and corresponding correction mask are created during the same clock cycle.

Data Correction

In the data correction stage, the mask is XOR'd together with the original incoming data to flip the error bit to the correct state, if needed. When there are no bit errors or double bit errors, all the mask bits are zeros. As a result, the incoming data goes through the ECC unit without changing the original data.

Error Diagnostics

In addition to displaying the error type, the reference design also supports diagnostic mode. Single, multiple, and triple bit errors can be introduced to the output codeword.

When the ERROR port is 00, no single, two, or greater bit error is detected. In other words, the examined data has no parity error. When the ERROR port is 01, it indicates single bit error occurred within the 72-bit codeword. In addition, the error is corrected, and the data is error free. When the ERROR port is 10, a two bit error has occurred within the codeword. In this case, no error correction is possible in this case. When the ERROR port is 11, errors beyond the detection capability can occur within the codeword and no error correction is possible. This is an invalid error type.

A deliberate bit error can be injected in the codeword at the output of the encoder as a way to test the system. Force_error provides several types of error modes.

Force_error = 00

This is the normal operation mode. No bit error has been imposed on the output of the encoder.

Force_error = 01

Single bit error mode. One bit is reversed (0 to 1 or 1 to 0) in the codeword at every rising edge of the clock. The single bit error follows the sequence moving from bit 0 of the codeword to bit 72. The sequence is repeated as long as this error mode is active.

Force_error = 10

Termed double bit error mode. Two consecutive bits are reversed (0 becomes 1 or 1 becomes 0) in the codeword at every rising edge of the clock. The double bit error follows the sequence moving from bit (0,1) of the codeword to bit (71, 72). The sequence repeats as long as this error mode is active.

Force_error = 11

Termed triple-error mode. Three-bits are reversed (0 becomes 1 or 1 becomes 0) in a codeword generated at every rising edge of the clock. The double bit error follows the sequence moving from bit (0,1, 2) of the codeword together to bit (70, 71, 72) sequentially. The sequence repeats as long as this error mode is active.

Utilization and Performance

The reference design utilizes a minimum amount of resources and has high performance. Table 3 provides a performance and utilization summary. The design was synthesized using the Xilinx Synthesis Tool (XST). The performance summary is based on ISE 8.2i speed specifications and reflects the 64-bit version ECC reference design only.

Table 3: **Performance Utilization Summary**

Device	Utilizaton(1)	Performance	
		Unpipelined	1-Stage Pipeline
XC2VP4 -6	16%	144 MHz	313 MHz
XC2VP7 -6	10%	136 MHz	298 MHz
XC2VP20 -6 or XC2VPX20 -6	5%	132 MHz	232 MHz
XC2VP50 -6	2%	127 MHz	176 MHz
XC4VLX15 -11	9%	197 Mhz	295 Mhz
XC4VFX20 -11	7%	204 Mhz	256 Mhz
XC4VFX60 -11	3%	158 Mhz	253 Mhz

Table 3: **Performance Utilization Summary *(Continued)***

Device	Utilizaton(1)	Performance	
		Unpipelined	1-Stage Pipeline
XC4VSX35 -11	4%	172 Mhz	293 Mhz
XC5VLX30 -2	4%	251 MHz	302 MHz
XC5VLX110 -2	1%	239 MHz	301 MHz

Latency

Although not required, the I/Os of the module are registered. For the encoder, the latency from the time the input data is presented at ENCIN to encoded data becomes available at ENCOUT is two clock cycles unpipelined or three clock cycles pipelined. For the decoder, the latency from the time the input data is presented to DECIN to the processed data becomes available at DECOUT is two clock cycles unpipelined or three clock cycles pipelined. The status signal ERROR is synchronous to DECOUT. Figure 8 illustrates the timing latencies.

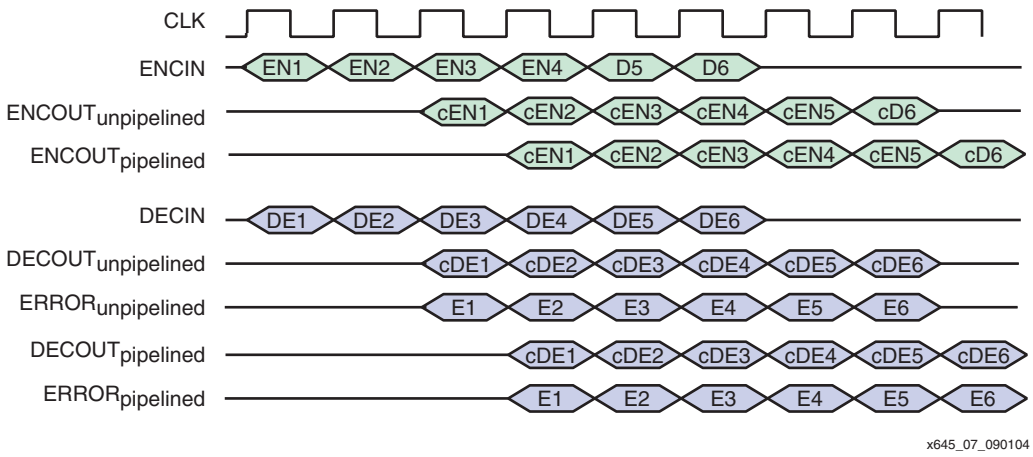


Figure 8: **Timing Diagram**

Latency Description

ENx = Write data before being encoded.

cENx = Write data after the encoder. Check bits are available for write.

DEx = Read data before the ECC unit.

cDE = Corrected read data after the ECC unit.

Ex = Error status generated from the ECC unit.

Pin Descriptions

Table 4 lists all of the user interface pins of the ECC module rising clock edge.

Table 4: **ECC Module Pin Description**

Pin Name	In/Out	Width (64-bit)	Width (32-bit)	Description
CLK	In			Clock input.
RESET	In			Active Low reset.
ENCIN	In	63:0	31:0	Original data input to the encoder.
ENCOUT	Out	63:0	31:0	Registered original data through the encoder.

Table 4: **ECC Module Pin Description *(Continued)***

Pin Name	In/Out	Width (64-bit)	Width (32-bit)	Description
PARITY_OUT	Out	7:0	6:0	Parity bits generated from the encoder based on the data (encin) registered at the same clock edge.
DECIN	In	63:0	31:0	Incoming data to the decoder.
DECOUT	Out	63:0	31:0	Corrected data from DECIN.
PARITY_IN	In	7:0	6:0	Parity bits associated with the incoming data (DECIN) registered at the same rising clock edge
FORCE_ERROR	In	1:0	1:0	Introduce bit error in the encoded dataword for test purpose. 00 – Normal operation 01 – Inject single bit error 10 – Inject double bit error 11 – Inject triple bit error
ERROR	Out	1:0	1:0	Error status 00 – No error 01 – Single bit error detected and corrected 10 – Double bit error detected. No correction 11 – Invalid bit error detected

Reference Design Files

The VHDL and Verilog reference design files are posted on the Xilinx website at:

http://www.xilinx.com/bvdocs/appnotes/xapp645.zip

Conclusion

This application note shows a simple method of encoding and looking up Hamming code in Virtex-II, Virtex-II Pro, and Virtex-4 devices.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/03/03	1.0	Initial Xilinx release.
09/17/03	1.1	Updated with Error Detection and Correction (EDC) functions for 32-bit data. Performance now reflects Speed Files version 1.81.
02/03/04	1.2	Expanded document to include pipelined applications.
09/01/04	2.0	Updated to include Virtex-4 FPGAs.
07/20/05	2.1	Updated Performance Utilization Summary Table (Table 3).
08/09/06	2.2	Updated **Utilization and Performance** section and Performance Utilization Summary Table (Table 3).