ECC Module

October 2012 Reference Design RD1025

Introduction

This reference design implements an Error Correction Code (ECC) module for the LatticeEC™ and LatticeSC™ FPGA families that can be applied to increase memory reliability in critical applications. The ECC module provides Single Error Correction - Double Error Detection (SECDED) capability based on a class of optimal minimum odd-weight error parity codes that provides better performance than typical Hamming-based SECDED codes. Several architecture options are identified that allow the user to optimally tailor the speed, resource utilization, and latency of the module implementation to their specific application requirements.

Features

- SECDED capability implemented using an optimal odd-weight parity matrix that provides better performance than typical Hamming-based codes
- Directly usable code for a (72,64) SECDED module provided. Specifications provided for similar (22,16) and (39,32) modules
- Separate registered encoder and decoder modules to support optimized integration with user logic
- Optional pipelining implementation to provide increased maximum speed of operation
- Error insertion/error indication diagnostic capabilities

Functional Description

ECC Overview

Hamming-based SECDED codes are widely used to increase memory reliability. Hamming encoding involves deriving a set of parity check bits covering different subsets of bits comprising a data word and concatenating or merging the check bits with the original data word. Decoding involves recalculating parity over the encoded data word, including the parity check bits. The parity check bits calculated at the encoder are designated syndrome bits and define what is known as the syndrome vector. The ability to correct single bit errors is provided by specifying encoding parity check relationships such that single bit errors in the encoded data words yield unique and specific syndrome vectors. A detailed scheme for generating effective parity check relationships was originally specified by R. W. Hamming [1].

It is well understood that the minimum number of check bits required for single bit error correction is specified by the relationship:

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

where D is the number of data bits and P is the number of parity check bits.[1] Thus, five check bits are required to implement single bit error correction for a 16-bit data word, six check bits are required for a 32-bit data word, and seven check bits are required for a 64-bit data word. With Hamming-based codes, double error detection is typically provided by an additional parity check bit across all the data and other parity bits. This bit will be 0 in the presence of double bit errors. Double errors essentially result in the mod-2 sum of syndrome vectors and, since single bit errors yield unique syndrome vector states, the mod-2 sum will be non-zero and at least one of the other syndrome

1. The total number of possible syndrome vector states is 2^P. The general requirement for single bit error correction is that any single bit error in the encoded data word must yield a unique syndrome vector. It must also be possible to detect no bit errors; thus, the total number of unique syndrome vectors required is equal to (D + P + 1).

bits will be set. Thus, double bit errors are indicated when the syndrome bit corresponding to the overall parity is 0 and one or more of the other syndrome bits is 1.

This reference design uses an SECDED code originally proposed by M. Y. Hsiao [2] that is simpler to implement and provides faster and better error detection capability than conventional Hamming-based codes. This code focuses on specifying an optimal minimum odd-weight parity checking relationship that reduces implementation logic depth and gate count.

As in a Hamming code, the parity relationships are specified such that any single bit error yields a unique syndrome vector state. In addition, the parity relationships are defined such that any single bit error is indicated by an odd number of syndrome bits being set to 1 (hence odd-weight parity checking), and the minimum odd weight codes (i.e. the codes having the minimum number of 1s) are chosen from the 2P possible codes.

This code provides single error detection/correction capability comparable to a Hamming code. Double error detection is provided by the odd-weight requirement. As for Hamming, double errors essentially result in the mod-2 sum of syndrome vectors. Since all parity vectors are odd weight and unique, the mod-2 sum will be even weight and non-zero. Hence any even weight, non-zero syndrome vector indicates the occurrence of an even number of errors.

The amount of logic required to implement the SECDED parity coding relationships is roughly equal to the number of 1s in the valid syndrome vectors. Logic depth, and hence processing speed, is roughly equal to the number of bits covered by the specific parity relationships. In [2], it is shown that minimum odd weight codes may be constructed that always have fewer 1s than Hamming codes and thus require less logic to implement. Logic depth is also reduced by eliminating the need for a parity bit across all of the data and check bits for double error detection.

ECC Implementation

The ECC parity check matrix for the code implemented in this design is specified in Table 1. Each data bit is included in three or five parity check bits, implementing minimum odd-weight encoding. Each parity check bit covers 26 data bits, minimizing the parity generation logic depth.

In the ECC encoder, each parity check bit is determined by XORing the corresponding data bits specified in Table 1. For example:

```
cb0 = din[0]^din[1]^din[2]^din[3]^din[4]^din[5]^din[6]^din[7]^din[10]^
      din[13]^din[14]^din[17]^din[20]^din[23]^din[24]^din[27]^din[35]^
      din[43]^din[46]^din[47]^din[51]^din[52]^din[53]^din[56]^din[57]^
      din[58];
```

At the decoder, syndrome bits are calculated over the corresponding data and received check bit. An error has occurred if any of the syndrome bits is set to 1, i.e.

```
error = sb[0]|sb[1]|sb[2]|sb[3]|sb[4]|sb[5]|sb[6]|sb[7];
```

If a single bit error has occurred, an odd number of syndrome bits will be set to 1:

```
one_error = error & (sb[0]^sb[1]^sb[2]^sb[3]^sb[4]^sb[5]^sb[6]^sb[7]);
```

For double bit errors, an even number of check bits will be set to 1:

```
two_error = error & !(sb[0]^sb[1]^sb[2]^sb[3]^sb[4]^sb[5]^sb[6]^sb[7]);
```

For the case of one bit error, the specific errored bit is identified by the specific state of the syndrome vector. For example, an error in data bit 0 is indicated by:

```
eb[0] = sb[0]&sb[1]&!sb[2]&!sb[3]&!sb[4]&sb[5]&!sb[6]&!sb[7];
```

Table 1. Parity Check Matrix for (72, 64) SECDED Code

Byte	Bit	sb0	sb1	sb2	sb3	sb4	sb5	sb6	sb7
0	0	1	1				1		
	1	1	1					1	
	2	1	1						1
	3	1		1	1	1	1		
	4	1		1				1	
	5	1		1					1
	6	1			1				1
	7	1			1			1	
1	8		1	1				1	
	9		1	1					1
	10	1	1	1					
	11		1		1	1	1	1	
	12		1		1				1
	13	1	1		1				
	14	1	1			1			
	15		1			1			1
2	16			1	1				1
	17	1		1	1				
	18		1	1	1				
	19			1		1	1	1	1
	20	1		1		1			
	21		1	1		1			
	22		1	1			1		
	23	1		1			1		
3	24	1			1	1			
	25		1		1	1			
	26			1	1	1			
	27	1			1		1	1	1
	28		1		1		1		
	29			1	1		1		
	30			1	1			1	
	31		1		1			1	
4	32		1			1	1		
	33			1		1	1		
	34				1	1	1		
	35	1	1			1		1	1
	36			1		1		1	
	37				1	1		1	
	38				1	1			1
	39			1		1			1
5	40			1			1	1	
	41				1		1	1	
	42					1	1	1	
	43	1	1	1			1		1
	44				1		1		1
	45					1	1		1
	46	1				1	1		
	47	1			1		1		
6	48				1			1	1
	49					1		1	1
	50						1	1	1
	51	1	1	1	1			1	
	52	1				1		1	
	53	1					1	1	
	54		1				1	1	
	55		1			1		1	
7	56	1				1			1
	57	1					1		1
	58	1						1	1
	59		1	1	1	1			1
	60		1				1		1
	61		1					1	1
	62			1				1	1
	63			1			1		1
Check	64	1							
	65		1						
	66			1					
	67				1				
	68					1			
	69						1		
	70							1	
	71								1

Errors are corrected by generating an error mask word based on the syndrome vector relationships. A bit will be set in the mask corresponding to the location of the errored bit. The error mask word is XORed with the data word, inverting and correcting the identified bit. If there is no error, or there are two bit errors, no bits will be set in the error mask and the data will be unaffected.

Note that the error decoding is only guaranteed to operate correctly in the presence of one or two bit errors. Miscorrections (i.e. data corruption) may occur in the presence of three or more errors. According to [2], the SECDED code used in this design has a lower probability of miscorrecting triple errors and a higher probability of correcting quadruple errors than conventional Hamming codes.

Module Architecture

The basic ECC encoder and decoder modules are shown in Figure 1. Block diagrams showing the encoder and decoder architectures are given in Figures 2 and 3, respectively. These figures show the available parameterized pipelining option. Included in the encoder module is an error insertion diagnostic capability.

Figure 1. ECC Modules

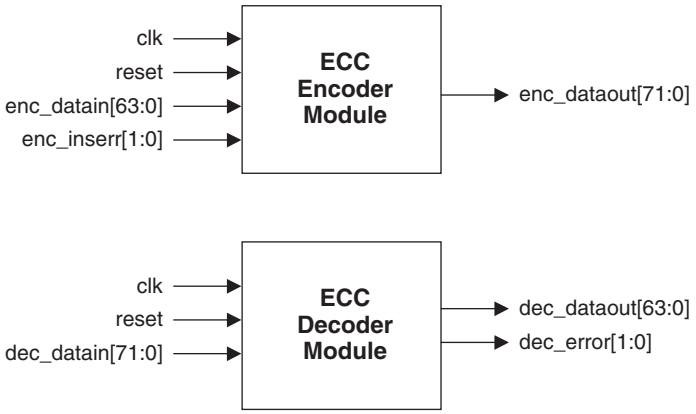


Figure 2. ECC Encoder

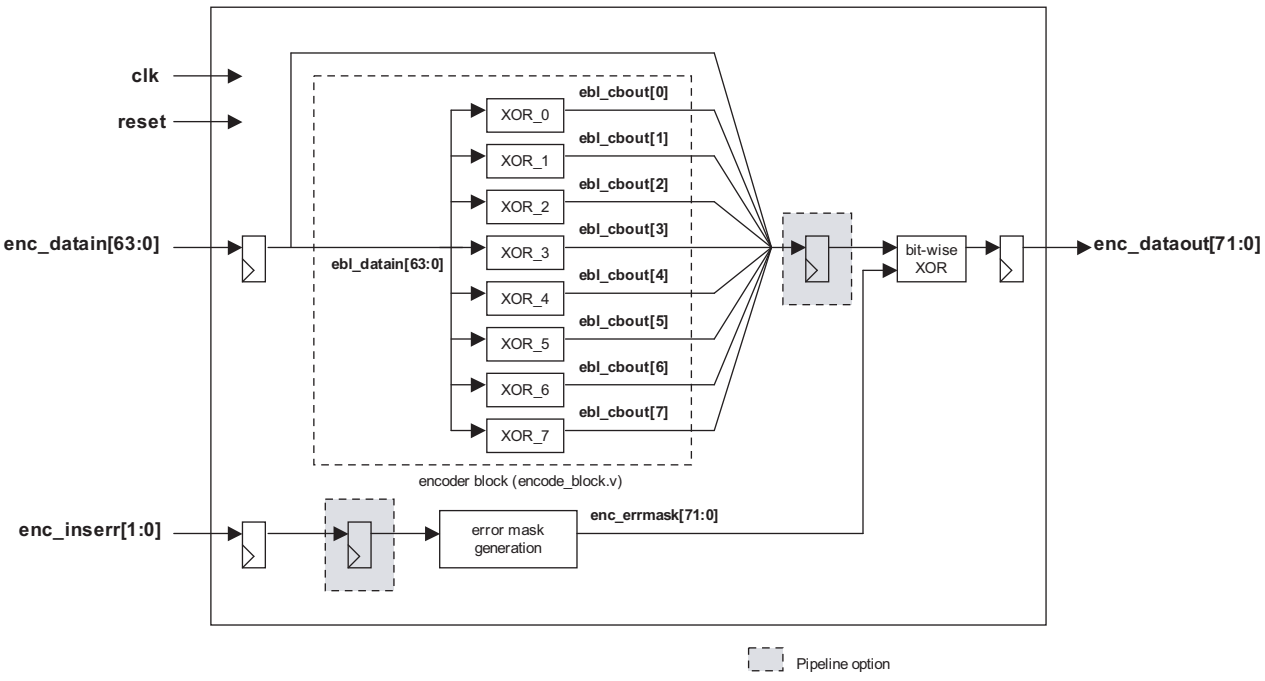
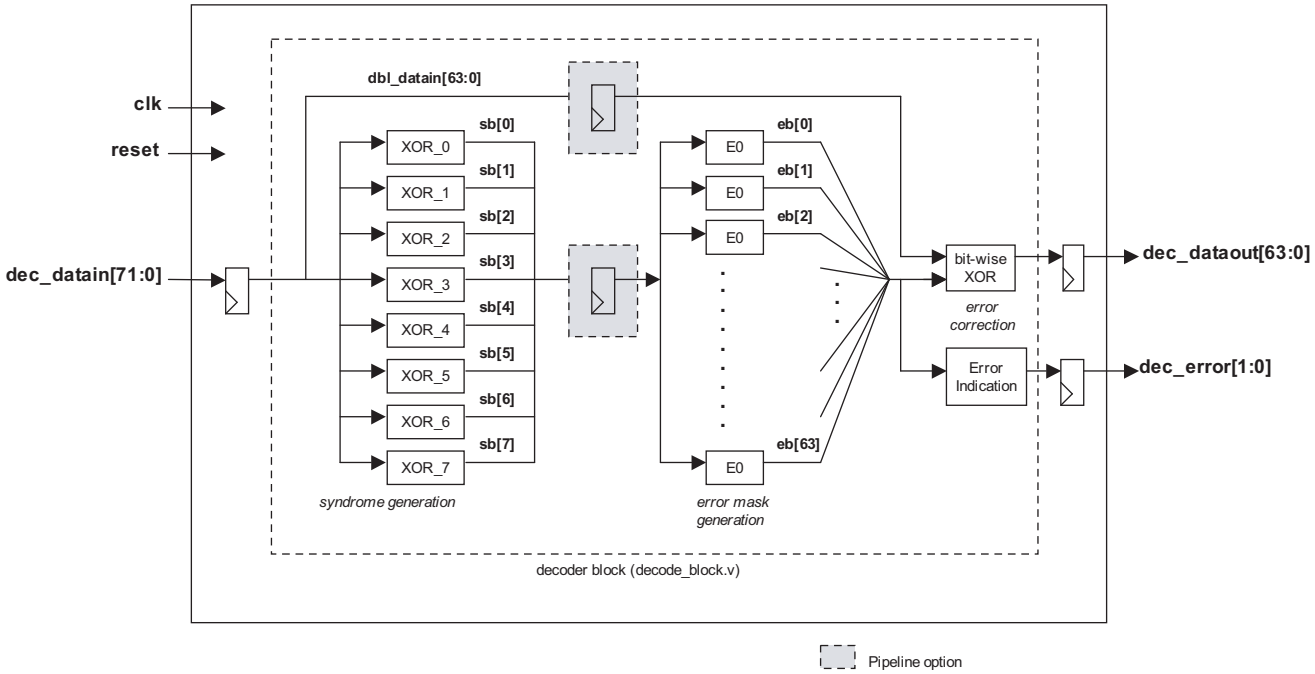


Figure 3. ECC Decoder



ECC Encoder

The ECC encoder generates the parity check bits as specified by the relationships defined in Table 1. An error insertion capability is also implemented that allows the insertion of either one or two bit errors in each data word. Error insertion is performed by creating an error generation bit mask that has either one or two bits set to one and XORing the mask with the output word specified by the input data and generated parity check bits. Error insertion is controlled by enc_inserr[1:0] as specified in Table 2. In the present reference design implementation, when error insertion is enabled, the position of specific bit(s) errored rotates through the encoded data word one bit each clock cycle. This capability could easily be modified to insert errors at a fixed location (or removed completely), reducing the logic utilization.

As shown in Figure 2, in this reference design the generated parity check bits [7:0] are appended as bits [71:65] in the encoded data word. Depending on the specific memory architecture implemented, users may want to insert the parity check bits in other bit positions within the encoded data word to avoid conditions where a single hardware failure can result in a degenerative all 0 (no error) state.

Table 2. Error Insertion Coding

enc_inserr[1]	enc_inserr[0]	Condition
0	0	No error insertion
0	1	One error inserted continuously
1	0	Two errors inserted continuously
1	1	Invalid

ECC Decoder

The ECC decoder calculates parity over the specific sets of data bits specified in Table 1 and the corresponding parity check bit in the encoded data word. The generated syndrome vector is then decoded to determine if any errors have occurred and specifically identify any bit errors. An error mask word is generated based on the syndrome vector relationships. A bit will be set in the mask corresponding to the location of an errored bit. The error mask word is XORed with the data word, inverting and correcting the specified bit. If there is no error, or there are

two bit errors, no bits will be set in the error mask and the data will be unaffected. The occurrence of none, one or two errors is indicated by the state of dec_error[1:0] as specified Table 3.

Table 3. Error Detection Coding

dec_error[1]	dec_error[0]	Condition
0	0	No error detected
0	1	One error detected
1	0	Two errors detected
1	1	Invalid

Optional Pipelining

As indicated previously, the depth of the encoding and decoding logic for this ECC implementation is reduced as compared to Hamming-based solutions, providing a higher maximum processing rate. The maximum processing rate for this design is limited by the depth of the syndrome and error mask generation logic in the decoder. The maximum processing speed can be increased by the optional pipelining capability. With pipelining, encoded data is registered before diagnostic error insertion in the encoder as shown in Figure 2, and the output of the syndrome generation circuit is registered before it is input to the mask generation circuit in the decoder as shown in Figure 3. This pipelining has the tradeoff of increasing latency by one clock cycle in both the encoder and decoder.

Parameters

The only explicit parameter for the ECC module reference design is the ability to include optional pipelining. This parameter is set via the ecc_params.v file. Additional modifications may be made by directly editing the source files. The reference design architecture is relatively simple and complete source code is provided.

I/O Descriptions

The I/O for the ECC modules is given in Table 4.

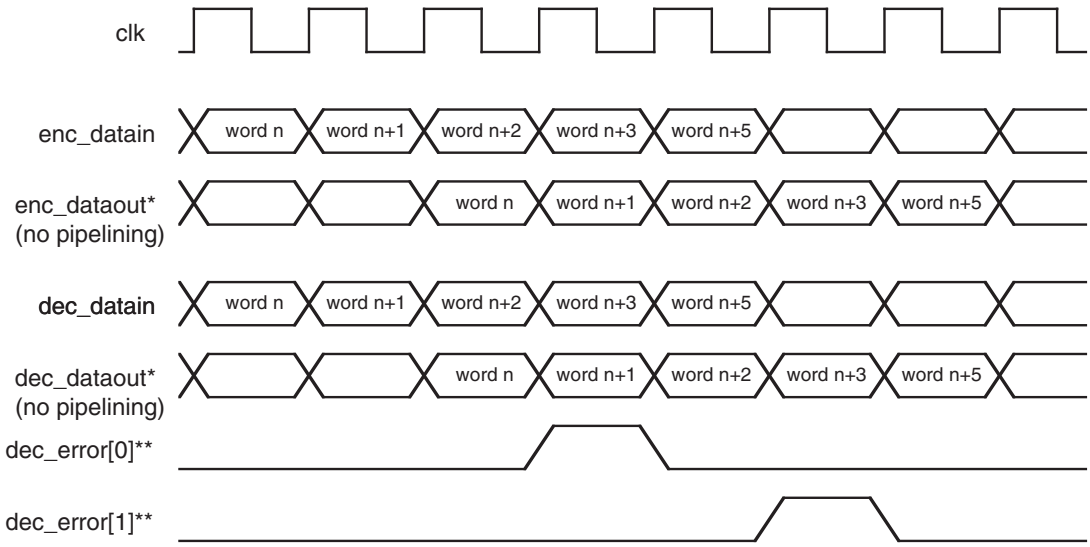
Table 4. ECC Module I/O

Signal Name	Active	I/O	Description
Encoder Module			
clk		I	Clock signal
reset	high	I	Reset signal
enc_datain[63:0]	high	I	Unencoded input data
enc_inserr[1:0]	high	I	Error insertion control
enc_dataout[71:0]	high	O	Encoded output data
Decoder Module			
clk		I	Clock signal
reset	high	I	Reset signal
dec_datain[71:0]	high	I	Encoded input data
dec_dataout[63:0]	high	O	Decoded output data
dec_error[1:0]	high	O	Error indication

Timing Diagrams

The timing latencies through the registered encoder and decoder ECC modules are shown in Figure 4.

Figure 4. ECC Module Timing Diagram



*Pipeline option increases dataout vs. datain latency by 1 byte in both the encoder and decoder.
**Single bit error in word n+1, double bit error in word n+3.

Implementation

As discussed previously, the ECC reference design includes separate registered encoder and decoder modules to facilitate easy integration with user logic. Source code for separate non-registered encoder and decoder blocks are also provided. These non-registered blocks implement only the check bit generation and syndrome-checking logic as indicated in Figures 2 and 3. These non-registered logic blocks may be directly integrated with other logic in the user's data path to implement memory protection with minimal throughput latency, although users will need to manage design parameters such as placement and fanout to ensure signal propagation through the encoding and decoding logic meets the system requirements. The logic utilization for these non-registered blocks is shown in Table 5.

This design is implemented in Verilog. When using this design in a different device, density, speed grade or performance and utilization may vary. Default settings are used during the fitting of the design.

Table 5. Performance and Resource Utilization

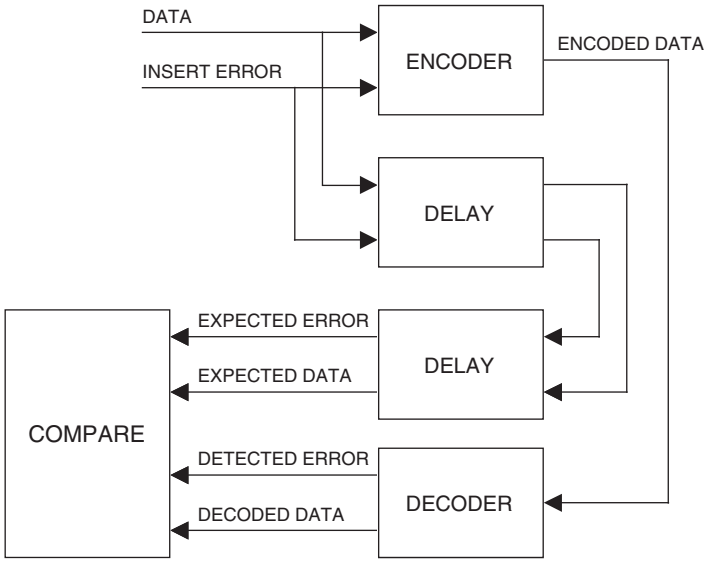
Device Family	Configuration	Language	Speed Grade	Utilization (LUTs)	f_{MAX} (MHz)	I/Os	Architectural Resources
LatticeEC[1]	Pipelined	Verilog	-5	411	>90	278	N/A
	Non- Pipelined	Verilog	-5	440	>90	278	N/A
	Non-registered (logic only)	Verilog	-5	271	—	210	N/A
LatticeSC[2]	Pipelined	Verilog	-5	391	>130	278	N/A
	Non- Pipelined	Verilog	-5	440	>130	278	N/A
	Non-registered (logic only)	Verilog	-5	342	—	210	N/A

1. Performance and utilization characteristics are generated using LFEC20E-5F484C with Lattice Diamond® 1.3 design software.
2. Performance and utilization characteristics are generated using LFSC3GA15E-5F900C with Lattice Diamond 1.3 design software.

Verification Environment

The test bench environment for verifying the ECC core is shown in Figure 5.

Figure 5. ECC Module Timing Diagram



The test bench inputs random data into the encoder. The same data is also input to shift registers which match the delays of the encoder and decoder circuits. The encoded data is applied to the input of the decoder. This is repeated for three scenarios: no errors, one error inserted and two errors inserted. The errors are inserted using a diagnostic capability built into the encoder section. The outputs of the decoder are compared to the expected outputs. In all cases the number of errors detected should match the number of errors inserted and in the cases where zero or one error is inserted the decoder output data should match the encoder input data.

References

[1] R. W. Hamming, “Error Detecting and Error Correcting Codes,” Bell System Tech. J., Vol. 26, No. 2, pp. 147-160, April 1950.

[2] M. Y. Hsiao, “A Class of Optimal Minimum Odd-weight-column SEC-DED Codes,” IBM J. RES. Develop., Vol. 14, pp. 395-401, July 1970.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2005	01.0	Initial release.
October 2012	01.1	Updated document with new corporate logo.
		Added support Lattice Diamond 1.3 design software.
		Added support for the LatticeSC device family.
		Updated equation in the ECC Overview section.

Appendix A. Additional SECDED Codes

The ECC module architecture may be easily modified to support other data sizes. Comparable optimal minimum odd-weight parity check matrices for (22,16) and (39,32) codes are given in Tables 6 and 7, respectively.

Table 6. Parity Check Matrix for (22, 16) SECDED Code

Byte	Bit	sb0	sb1	sb2	sb3	sb4	sb5
	0	1	1	1			
	1	1	1			1	
	2	1	1				1
0	3	1				1	1
	4	1		1			1
	5	1			1		1
	6		1	1	1		
	7		1	1		1	
	8		1	1			1
	9		1		1	1	
	10	1			1	1	
1	11				1	1	1
	12		1			1	1
	13			1	1	1	
	14	1		1	1		
	15			1	1		1
	16	1					
	17		1				
	18			1			
Check	19				1		
	20					1	
	21						1

Table 7. Parity Check Matrix for (39, 32) SECDED Code

Byte	Bit	sb0	sb1	sb2	sb3	sb4	sb5	sb6
0	0	1					1	1
	1	1				1		1
	2	1			1	1		
	3	1		1				1
	4	1	1					1
	5	1				1	1	
	6	1			1		1	
	7	1	1			1		
1	8		1				1	1
	9		1			1		1
	10		1		1			1
	11		1	1				1
	12		1			1	1	
	13		1		1		1	
	14	1	1				1	
	15		1		1	1		
2	16			1	1		1	
	17			1			1	1
	18		1	1			1	
	19	1		1			1	
	20			1		1	1	
	21		1	1		1		
	22	1		1		1		
	23			1		1		1
3	24	1	1		1			
	25				1	1		1
	26			1	1	1		
	27			1	1			1
	28				1	1	1	
	29		1	1	1			
	30	1		1	1			
	31	1			1			1
Check	32	1						
	33		1					
	34			1				
	35				1			
	36					1		
	37						1	
	38							1