International Journal of Scientific & Engineering Research, Volume 4, Issue 8, August 2013
ISSN 2229-5518

Design and Characterization of ECC IP core using Improved Hamming Code

Arathy S, Nandakumar R

Abstract— Hamming code with additional parity bit can be used for single error correction and double error detection. In conventional hamming code redundancy bits are interspersed in powers-of-two positions at the transmitter end. At the receiver these redundancy bits are to be extracted from the powers-of-two positions. In improved hamming code the redundancy bits are placed at the end of data bits. This eliminates the overhead of interspersing redundancy bits at the sender end and their removal later at the receiver end. Further the overhead bits involved in the process of calculation of redundancy bits is lower in improved hamming code. This paper describes the design of a synthesizable Error Correction Codes (ECC) IP core which uses improved hamming code. The design is described using Verilog HDL, simulated using ModelSim and prototyped in Altera® platform FPGA. Resource utilization and power analysis was done using Altera® Quartus II. Hardware test results are obtained from Signal Tap Logic Analyzer. To make a comparison between ECC using conventional hamming code and ECC using improved hamming code Matlab plots are used.

Index Terms— Error Correction Codes, IP core, Power analysis, Resource utilization.

1 Introduction

DATA that is either transmitted over communication channel (e.g. bus) or stored in memory is not completely error free. Error can be caused by different reasons. Transmission errors are mainly due to signal distortion or attenuation. For example in a transmission system if the clocks are not synchronized, then sender and receiver can be out of synchronisation thereby resulting in error. Storage errors are mainly due to electromagnetic interference in DRAM memory cell and bit flipping in magnetic storage devices.

Error detection is the ability to detect errors and error correction has an additional feature that enables identification and correction of the errors. Error detection always precedes error correction. Both can be achieved by having redundant check bits in addition to data. Original Data is encoded with the redundant bit(s). New data formed is known as code word.

Different error correction methods are there. Hamming code is well known for single error correction and double error detection[2,4]. In conventional Hamming code redundancy bits are to be interspersed in powers-of-two positions at the transmitter end. At the receiver these redundancy bits are to be extracted from the powers-of-two positions. In improved Hamming code the redundancy bits are placed at the end of data bits[1]. This eliminates the overhead of interspersing redundancy bits at the sender end and their removal later at the receiver end. Further the overhead bits involved in the process of calculation of redundancy bits is lower in improved Hamming code[1].

The section II of this paper is a tutorial review of conventional Hamming code error correction. The section III describes the proposed improved Hamming code error correction. The section IV describes the design methodology of ECC IP core. The section V describes the comparison of improved Hamming code ECC and conventional Hamming code ECC. The section VI describes the simulation result. The section V describes the implementation results.

2 Conventional Hamming Code Error Correction

Hamming code with additional parity bit can be used for single error correction and double error detection[2,5]. The principle behind the Hamming code is the addition of 'r' redundancy bits to 'n' data bits. The number of redundancy bits and number of data bits required for single error correction should satisfy equation 1.

$$2^r >= n+r+1 \qquad (1)$$

One more redundancy bit is required for single error correction and double error detection. For example, if the data bit is of 11 bit wide the number of redundancy bits will be 5 for single error correction and double error detection. The total codeword width is 16. These five bits are interspersed in data at locations 0,1,2,4,8. The parity bits are calculated according to the equations derived from the truth table given in table 1.

TABLE 1
TRUTH TABLE

Bit position of data	P[3]	P[2]	P[1]	P[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0

7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

P[0] is having value '1' at bit locations 1, 3, 5, 7, 9, 11, 13 and 15. Therefore P[0] is selected such that there is even parity at these positions (XXX1 <= 15). P[1] is selected such that there is even parity at positions 2, 3, 6, 7, 10, 11, 14 and 15 (XX1X <= 15). P[2] is selected such that there is even parity at positions 4, 5, 6, 7, 12, 13, 14, 15 (X1XX <= 15). P[3] is selected such that there is even parity at positions 8, 9, 10, 11, 12, 13, 14, 15 (1XXX <= 15). P[4] is selected such that there is even parity at all the bit positions including the redundancy bits. These parity bits are interspersed in positions 0, 1, 2, 4 and 8. For the calculation of parity bits at positions 1, 2, 4, 8 & 0, even parity checks were performed on 8, 8, 8, 8 & 16 bits respectively. Thus a total of 48 bits are involved in the process of hamming bits calculation. The codeword format for a sample data 11'b11001100110 is shown in figure 1. Parity bits are shown in bold format.

1	1	0	0	1	1	0	**0**	0	1	1	**0**	0	**1**	**1**	**0**
15	14	13	12	11	10	9	**8**	7	6	5	**4**	3	**2**	**1**	**0**

Fig 1: Codeword Format

This codeword is transmitted or stored in the memory. At the receiver end, the parity bits are removed. A parity check is done between the transmitted parity and parity of the received codeword. The result of comparison determines the nature of error. If single bit error has been occurred, then a mask will be generated and the data will be corrected. The error-detection and correction process in hamming code is as illustrated in table 2.

TABLE 2
ERROR DETECTION AND CORRECTION USING HAMMING CODE

Received information including Hamming bits	Status of parity check	Conclusion
1100110001100110	00000	No error
1100110001101110	10011	Error at position 3
1100010001100110	11100	Error at position 12

3 IMPROVED HAMMING CODE ERROR CORRECTION

In improved Hamming code, the redundancy bits are placed at the end of the data bits. This greatly reduces the overhead in interspersing the redundancy bits at the sender end and their removal later at the receiver end. Further the number of overhead bits involved in the process of calculation of redundancy bits is less[1].

The number of redundancy bits in this method is same as that for conventional Hamming code for some values of n. But in some cases, it will be just one more redundancy bit than needed in the Hamming code[1]. The number of redundancy bits, 'r' to be appended to n-bit data to obtain single error correction is according to equation 2.

$$2^{(r-1)} - 1 >= n \qquad (2)$$

For example, if the available space for codeword is only 16 bits, then data bit should be only 10 bit wide and the number of redundancy bits will be 6 to obtain single error correction and double error detection. These six bits are placed at locations 15, 14, 13, 12, 11 and 10. The parity bits are calculated according to the equations derived from the truth table given in table 3.

TABLE 3
TRUTH TABLE

Bit position of data	P[3]	P[2]	P[1]	P[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0

P[0] is having value '1' at bit locations 1, 3, 5, 7 and 9. Therefore P[0] is selected such that there is even parity at these positions (XXX1 <= 10) [1]. P[1] is selected such that there is even parity at positions 2, 3, 6, 7 and 10 (XX1X <= 10). P[2] is selected such that there is even parity at positions 4, 5, 6 and 7 (X1XX <= 10). P[3] is selected such that there is even parity at positions 8, 9 and 10 (1XXX <= 10). P[4] is selected such that there is even parity at the bit positions of redundancy bits

P[3:0]. P[5] is selected such that there is even parity at all the bit positions including the redundancy bits P[4:0]. These parity bits are interspersed in positions 15, 14, 13, 12, 11 and 10. For the calculation of parity bits, even parity checks were performed on 5, 5, 4, 3, 4 & 16 bits respectively. Thus a total of 37 bits are involved in the process of hamming bits calculation. The codeword format for a sample data 10'b1100110011 is shown in figure 2. Parity bits are shown in bold format.

0	**0**	**0**	**0**	**1**	**1**	1	1	0	0	1	1	0	0	1	1
15	**14**	**13**	**12**	**11**	**10**	9	8	7	6	5	4	3	2	1	0

Fig 2: Codeword Format

This codeword is transmitted or stored in the memory. At the receiver end, the parity bits are removed. A parity check is done between the transmitted parity and parity of the received codeword. The result of comparison determines the nature of error. If single bit error has been occurred, then a mask will be generated and the data will be corrected. The error-detection and correction process in hamming code is as illustrated in table 4.

TABLE 4
ERROR DETECTION AND CORRECTION USING IMPROVED HAMMING CODE

Received information including Hamming bits	Status of parity check	Conclusion
0000111100110011	000000	No Error
0000111100110111	100011	Single Error at position 2
0000111000110011	101010	Single Error at position 9

4 DESIGN METHODOLOGY

The main blocks include ECC-Encoder and ECC- Decoder – Corrector.

Block Diagram

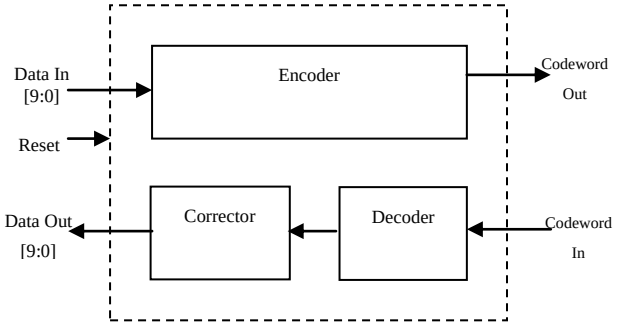


Fig 3: Block diagram

The main blocks are encoder, decoder and corrector. Encoder encodes the input data into a codeword that is either transmitted or stored. The decoder receives the codeword from the channel or memory. Decoder and corrector are responsible for error detection and correction.

Design Flow

The encoder takes the 10-bit input data and encodes the message into a (10 + 6) bit codeword. The process flow of encoder is shown in figure 4 .

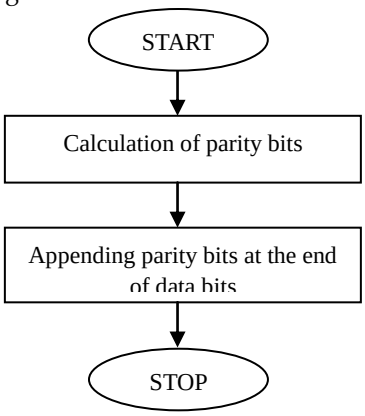


Fig 4: Encoder process flow chart

The parity bits are calculated as explained in section III. The calculated parity bits are placed at the end of data bits thereby reducing the overhead of interspersing redundancy bits in powers-of-two positions. The process flow of decoder is given in figure 5.

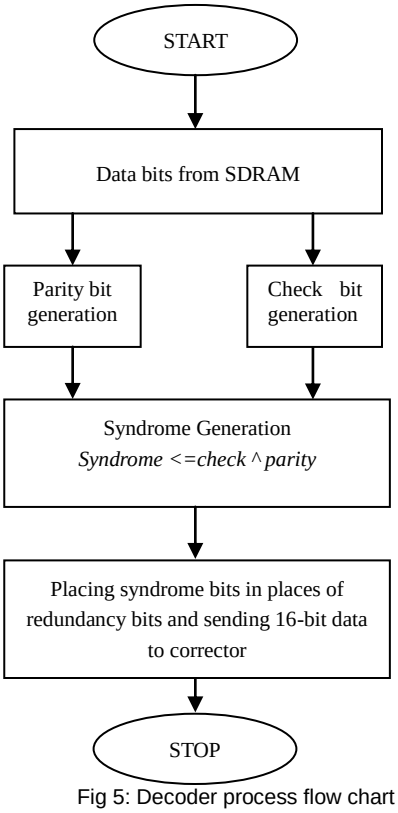


Fig 5: Decoder process flow chart

Decoder receives data bits from memory or channel. The data bits represents the codeword corresponding to the actual message. Decoder first extracts the redundancy bits(check bits) from the end positions. The decoder again calculates the parity bits corresponding to the received data. Decoder compares the check bits and the parity bits and generates a syndrome. The syndrome bits are placed at the end of the data replacing the parity bits. This data is given to the corrector. The process flow of corrector is given in figure 6.

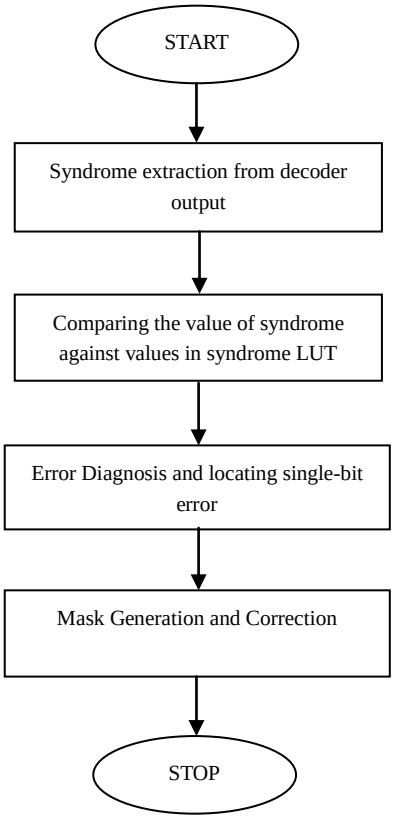


Fig 6: Corrector process flow chart

The first step of the ECC-Corrector is the extraction of syndrome bits from the received data. By extracting syndrome bits, separation of redundancy bits and data bits takes place. The decoder then compares the syndrome value against values in the syndrome Look Up Table (LUT). The status of error depends upon the value of syndrome. All the single-bit errors are located and corrected. Double-bit errors are detected and not corrected. The next step in the corrector process flow is the mask generation and correction. Mask is generated only when the error is diagnosed as single-bit error. Depending upon the location of error mask varies. Separated data bits is XORed with mask to obtain the corrected data. Separate pin is given for indicating the status of error. Conditions of error output pins and the status of error is given in table 5.

Table 5
Error Status Table

Error[1]	Error[0]	Diagnosis
0	0	There is no error on the message on the output.
1	0	There was one error on the codeword the message is equivalent to the original.
0	1	There are two errors on the codeword no correction have been made.
1	1	Not possible.

5 Comparison Of Improved Hamming Code With Conventional Hamming Code

The proposed method of error correction has reduced the computational complexity[1]. The overhead bits involved in the process of calculation of redundancy bits are lower in improved Hamming code. The table 6 shows the values of overhead bits for both methods and the calculated percentage reduction of overhead bits in the proposed method. Comparison is done using Matlab plot shown in figure 7.

TABLE 6
COMPARISON TABLE

Data bits	Parity bits	Overhead bits		% reduction in overhead bits
		Conventional Hamming code	Improved Hamming code	
4	4	19	13	31.57%
8	5	37	27	27.02%
16	6	69	57	17.39%
32	7	138	123	10.87%

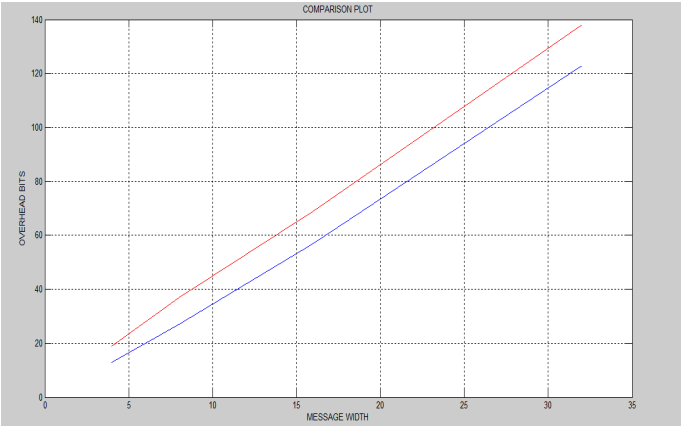


Fig 7: Comparison Plot

In figure 7, comparison of overhead bits in case of conventional Hamming code and improved Hamming code is shown. Overhead bits are plotted against message width. Upper line indicates the overhead bits for conventional Hamming code and the lower line indicates the overhead bits for improved Hamming code.

6 Simulation Results

The simulation result of encoder, decoder with no error, decoder with single error and decoder with double error are shown in figure 8 , figure 9, figure 10 and figure 11 respectively. The designed IP core is further prototyped using Altera DE2 FPGA board[3].

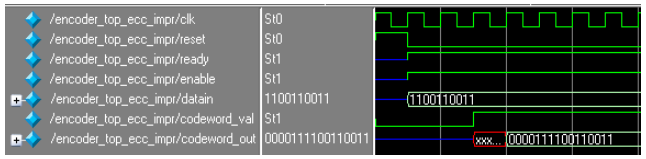


Fig 8: Simulation Result of Encoder

Note on simulation result :

Reset:(clk=1, reset=1, ready = z, enable = z, datain = zzzzzzzzzz, codeword_val = 0, codeword_out = zzzzzzzzzzzzzzzz)

Encoding:(clk=1, reset=0, ready = 1, enable = 1, datain = 1100110011, codeword_val = 1, codeword_out = 0000111100110011)

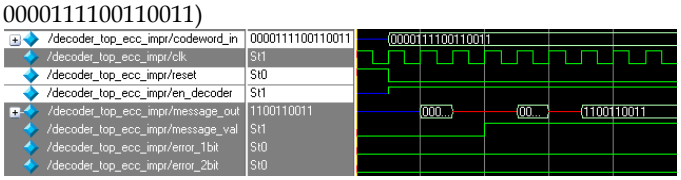


Fig 9: Simulation Result of Decoder-corrector with no error

Note on simulation result :

Reset:(clk=1, reset=1, en_decoder = z, codeword_in = zzzzzzzzzzzzzzzz, message_out = zzzzzzzzzz, message_val = 0, error_1bit = 0, error_2bit = 0)

Decoding:(clk=1, reset=0, en_decoder = 1, codeword_in = 0000111100110011, message_out = 1100110011, message_val = 1, error_1bit = 0, error_2bit = 0)

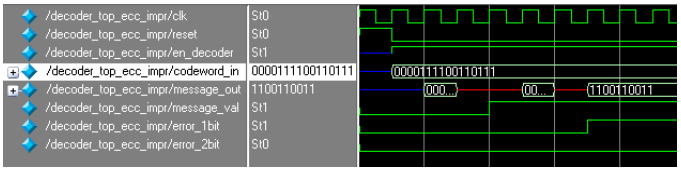


Fig 10: Simulation Result of Decoder with single error

Note on simulation result :

Reset:(clk=1, reset=1, en_decoder = z, codeword_in = zzzzzzzzzzzzzzzz, message_out = zzzzzzzzzz, message_val = 0, error_1bit = 0, error_2bit = 0)

Decoding:(clk=1, reset=0, en_decoder = 1, codeword_in = 0000111100110111, message_

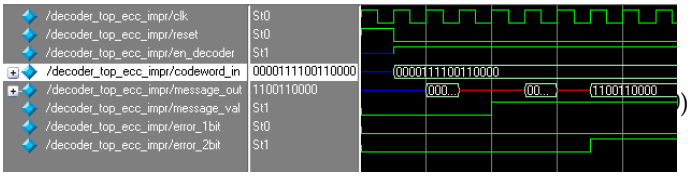


Fig 11: Simulation Result of Decoder with double error

Note on simulation result :

Reset:(clk=1, reset=1, en_decoder = z, codeword_in = zzzzzzzzzzzzzzzz, message_out = zzzzzzzzzz, message_val = 0, error_1bit = 0, error_2bit = 0)

Decoding:(clk=1, reset=0, en_decoder = 1, codeword_in = 0000111100110000, message_out = 1100110000, message_val = 1, error_1bit = 0, error_2bit = 1)

7 Implementation Results

The ECC IP core is tested by using Altera® DE2 FPGA board. Altera Quartus® II is used to synthesis the design. Signal Tap® II Logic Analyzer is used to take hardware test result from FPGA. Resource utilization summary is given in table 7 and power analysis summary is given in table 8.

TABLE 7
RESOURCE UTILIZATION TABLE

RESOURCE	***USAGE***
Estimated Total logic elements	*180*
Total combinational functions	*87*
Logic element usage by number of LUT inputs	
-- 4 input functions	*24*
-- 3 input functions	*6*
-- 2 input functions	*57*
Logic elements by mode	
-- normal mode	*87*
-- arithmetic mode	*0*
Total registers	*159*
-- Dedicated logic registers	*159*
-- I/O registers	*0*
I/O pins	*60*
Maximum fan-out node	*clk*
Maximum fan-out	*159*
Total fan-out	*748*
Average fan-out	*2.44*

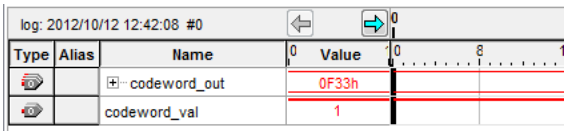
Resource Utilization table summarizes usage statistics for resources including logic elements, registers, I/O pins, memory blocks, interconnect usage, and fan-out.

TABLE 8
POWER ANALYSIS TABLE

PARAMETER	*RESULT*
Total Thermal Power Dissipation	*113.94 mW*
Core Dynamic Thermal Power Dissipation	*0.00 mW*
Core Static Thermal Power Dissipation	*79.94 mW*
I/O Thermal Power Dissipation	*34.01 mW*

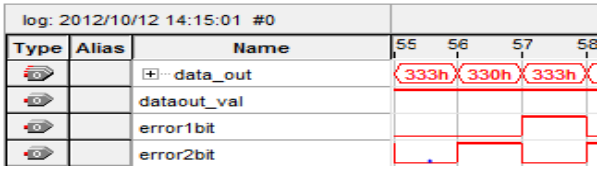
Power Analysis table gives details about the core dynamic thermal power dissipation, core static thermal power dissipation, I/O thermal power dissipation and total thermal power dissipation.

Hardware test result is obtained from Signal Tap Logic Analyzer. Encoder and decoder-corrector is tested and the result is given in figure 12 and figure 13 respectively.



Datain	Codeword_out	Codeword_val
1100110011	0000111100110011	1

Fig 12: Hardware Test Result of Encoder



Codeword_in	Data_out	Dataout_ val	Error1bit	Error2bit
0000111100110011	1100110011	1	0	0
0000111100110000	1100110000	1	0	1
0000111100110111	1100110011	1	1	0

Fig 13: Hardware Test Result of Decoder-Corrector

The hardware test results tabulated above are subjected to vis-à-vis comparison for the function table, for verifying compliance.

8 CONCLUSION

This paper describes the design, simulation and characterization of synthesizable ECC IP core using improved Hamming code. This eliminated the overhead of interspersing the redundancy bits at the sender end and their removal at the receiver end. Further the effort needed in identifying the values of the redundancy bits is lower. The design was developed using Verilog HDL. The proposed design has been tested by implementing the design on Altera DE2 board which uses Cyclone-II device.

REFERENCES

[1] Kumar, U.K.; Umashankar, B.S.; , "Improved Hamming Code for Error Detection and Correction," *Wireless Pervasive Computing, 2007. ISWPC '07. 2nd International Symposium on* , vol., no., 5-7 Feb. 2007 doi: 10.1109/ISWPC.2007.342654

[2] Altera ., "DDR and DDR2 SDRAM ECC Reference Design". 2006.

[3] Altera DE2 Development Board User Manual available online at ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

[4] W. Gao and S. Simmons, "A study on the VLSI implementation of ECC for embedded DRAM," Electrical and Computer Engineering,2003. IEEE CCECE 2003. Canadian Conf., Vol. 1, pp. 203-206, May 2003.

[5] 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.