Design and Characterization of SDRAM Controller IP Core with Built In Refined ECC Module

Arathy S, Nandakumar R, Hima Sara Jacob

Abstract— In modern digital systems large capacity and data transfer rate is required. Synchronous DRAM (SDRAM) become the memory of choice due to its speed, burst access and pipeline features. A Controller is required to provide proper commands for SDRAM initialization, read/write accesses and memory refresh. In Synchronous DRAM memories data errors may occur due alpha particles. To ensure reliable data storage, an error correction and detection scheme is required. This paper describes the design and characterization of SDRAM Controller IP core with built in Refined Error Correcting Codes (ECC) module. The refined ECC module uses improved Hamming code which has a better performance than the conventional 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.

Index Terms — Error Correcting Codes, Error Correction, Error Detection, IP core, Power analysis, Resource utilization, Synchronous DRAM.

1 Introduction

Synchronous DRAMs (SDRAMs) become the memory of choice in many digital systems because it provides a significant improvement in bandwidth performance over traditional asynchronous DRAMs such as "FPM" (Fast Page Mode) and "EDO" (Extended Data Out) [6]. In Synchronous DRAMs input address, data, and control signals are typically latched on the positive edge of the clock signal. SDRAMs offer several features such as multiple internal banks, burst mode access, and pipelining of operation executions, that helps to improve bandwidth performance.

There are two popular types of SDRAM in market. The most common single data rate (SDR) SDRAM transfers data typically on the rising edge of the clock. The other is the double data rate (DDR) SDRAM [5] which transfers data on both the rising and falling edge to double the data transfer throughput. Other than the data transfer phase, the different power-on initialization and mode register definitions; these two SDRAMs share the same command set and basic design concepts. This paper describes a design that is targeted for SDR SDRAM. However, due to the similarity of SDR and DDR SDRAM, this design can also be adapted for a DDR SDRAM controller.

As the dimensions and operating voltages of electronic components are reduced, their sensitivity to radiation increases dramatically[9]. The alpha particles emitted by trace uranium and thorium impurities in packaging materials were the dominant cause of soft errors in SDRAM devices. Improved Hamming code Error Correction [7] is used in the proposed controller to provide Single Error Correction and Double Error Detection.

For benchmarking purpose, the Micron® SDR SDRAM MT48LC8M16A2 [4] is chosen as a target for this design. The target memory is a 16-bit memory. But, by the introduction of error correction feature, the width of data bus is reduced to 10 bits. Also, this design has been verified by using memory simulation model.

The section 2 of this paper is a tutorial review of Synchronous DRAMs. The section 3 describes the design of the proposed SDRAM Controller with refined ECC module. The section 4 describes the simulation results. The section 5 describes the implementation results of the proposed controller.

2 Synchronous DRAM Review

SDRAM, or Synchronous Dynamic Random Access Memory is a form of semiconductor memory that can run at faster speeds than conventional DRAM. Since SDRAM has a synchronous interface, it has an internal finite state

- *Arathy S is currently pursuing masters degree program in VLSI and Embedded System in Saintgits College of Engineering,Kottayam, Kerala, India. E-mail: arathysnair89@gmail.com*
- *Nandakumar R is Scientist/Engineer B at National Institute of Electronics and Information Technology, Calicut, Kerala, India. E-mail :* nanda24x7@gmail.com
- *Hima Sara Jacob is currently pursuing masters degree program in VLSI and Embedded System in Saintgits College of Engineering, Kottayam, Kerala, India. E-mail: hima.sara24@gmail.com*

machine that pipelines incoming instructions. Thus the speed of operation is much higher.

As a result of the multiple banks present and pipelining feature, SDRAM is capable of keeping two sets of memory addresses open simultaneously thereby it cuts down the delays associated with asynchronous RAM, which must close one address bank before opening the next. The term pipelining is used to describe the process whereby the SDRAM can accept a new instruction before it has finished processing the previous one. Another important feature of SDRAM is its facility for burst read and burst write[4].

The Micron® 128Mb [4] SDRAM referenced in this paper is a high-speed CMOS, dynamic random access memory containing 134,217,728 bits. It is internally configured as a quad-bank DRAM with a synchronous interface. Each of the 33,554,432-bit banks is organized as 4,096 rows by 512 columns by 16 bits. The 128Mb SDRAM is designed to operate in 3.3V memory systems. All inputs and outputs are LVTTL-compatible.

3 Proposed Controller Design

3.1 Introduction

The proposed SDRAM Controller is designed to work with a standard memory from Micron Technology® with series MT48LC8M16A2™. It has a user interface end on one side and 128Mb SDRAM on the other end. The design is coded in Verilog HDL. The controller offers facility for programmable burst lengths of 1, 2, 4, and 8, programmable CAS latency of 2 and 3.

Initialization should be done before applying any normal operation. Read and Write should be performed only after the initialization. The proposed controller design automatically performs all the initialization procedures[1,2,4].

The alpha particles emitted by trace uranium and thorium impurities in packaging materials were the dominant cause of soft errors in SDRAM devices. The alpha particle is composed of two neutrons and two protons—a doubly ionized helium atom emitted from the nuclear decay of unstable isotopes [9]. The most common source of alpha particles are from the naturally occurring ^{238}U, ^{235}U, and ^{232}Th. These impurities emit alpha particles at specific discrete energies over a range from 4 to 9 MeV. When an alpha particle travels through a material, it loses its kinetic energy predominantly through interactions with the electrons of that material and thus leaves a trail of ionization in its wake. The higher the energy of the alpha particle, the farther it travels before being "stopped" by the material. Alpha particles from outside the packaged device are clearly not a concern—only alpha particles emitted by the device materials and packaging materials need be considered. Other than alpha particles, the reaction of high-energy cosmic neutrons with silicon and other device materials and the reaction of low-energy cosmic neutrons with high concentrations of ^{10}B in the device can cause data errors leading to failures in electronic devices [9].

Improved Hamming code is used to provide Single Error Correction and Double Error Detection [7]. Refined ECC block consists of encoder and decoder-corrector, which can detect and correct single-bit errors and detect double-bit errors.

3.2 Block Diagram

The block diagram of proposed SDRAM Controller is shown in Fig. 1. The main function of the controller is to convert user commands to commands that can be understandable by the memory [6]. Another is to detect and correct single-bit errors and detect double-bit errors[3]. The main blocks include datapath, finite state machine, initialization control block and ECC block.

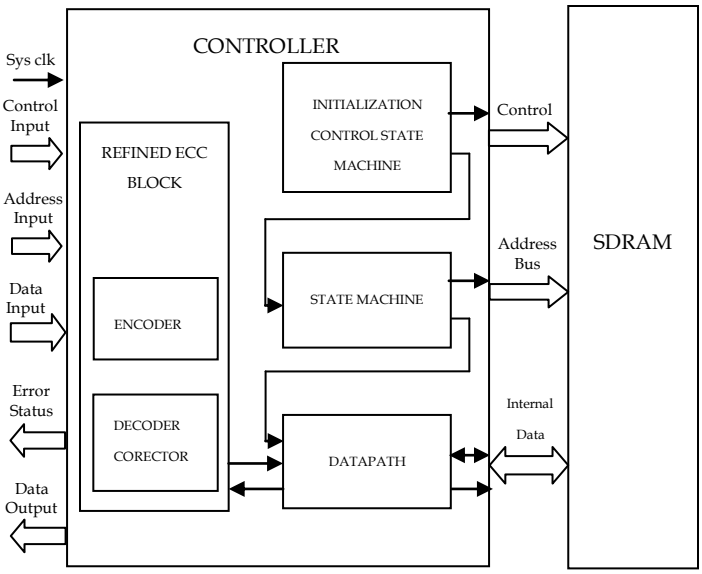

Fig. 1 Controller block diagram

The inputs like command (cmd) and reset are collectively shown as control input. The memory address is given through the address input. The data during write operation is given through the data input. Data during read operation is obtained through data output. Error status is also obtained as an output.

3.3 Pin Description

The input output diagram of controller is shown in Fig. 2.

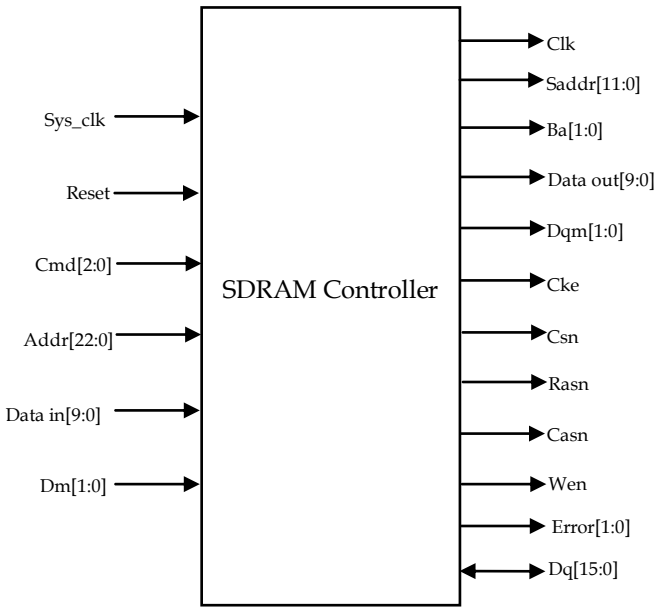

Fig. 2 Pin out diagram of controller

Type and the function of all the pins are described in table I.

TABLE 1
PIN DESCRIPTION

PIN	TYPE	DESCRIPTION
Sys Clk	Input	Input clock to the controller. Used to generate the clock (clk) to the memory.
Reset	Input	System reset - Used to reset the total system.
Cmd [2:0];	Input	Command input which distinguishes between different operations like Pre-charge, refresh, load mode register, active, read and write.
Data in [9:0]	Input	10-bit data input to the controller which is used as data during write operation.
Data out [9:0]	Output	10-bit data output from the controller during read operation.
Addr [22:0]	Input	23-bit Address bus. From this 23-bit address row address, column address and bank address is decoded.
Dm [1:0]	Input	Dm is used to generate the mask signal for the memory (Dqm).
Clk	Output	CLK is generated by the controller. This clock is given as the clock input to the memory. Typical values are 100MHz and 133MHz.
Saddr [11:0]		Address input A0–A11 are sampled during the ACTIVE command (row address A0–A11) and READ/WRITE command (column-address A0–A8).
Ba [1:0]	Output	Bank address is used to select the bank
Cke	Output	CKE activates (HIGH) and deactivates (LOW) the CLK signal.
Csn	Output	CS enables (LOW) and disables (HIGH) the command decoder. All commands are masked when CS is registered HIGH.
Rasn	Output	Command inputs WE, CAS, and RAS
Casn	Output	(along with CS) define the command
Wen	Output	being entered.
Error [1:0]	Output	Error is the output signal used to display the error status. If it is "00", no error has occurred. If it is "10", single-bit error has occurred and corrected. If it is "01", double-bit error has occurred and not corrected. Value "11" is not possible
Dqm [1:0]	Output	DQM is an output mask signal for write accesses and an output enable signal for read accesses. Input data to the memory is masked when DQM is sampled HIGH during a WRITE cycle. The output buffers are placed in a High-Z state (2-clock latency) when DQM is sampled HIGH during a READ cycle. DQM is generated from Dm signal.
Dq [15:0]	Inout	Bidirectional data bus which is connected to the bidirectional data bus of memory. During write operation Dq is directed from controller to the memory. During read operation Dq is directed from memory to the controller

3.4 Initialization Control State Machine

According to architecture and working principle of SDRAM [4], the controller design uses two Finite State Machines to implement timing-logic control [1,2]. The SDRAM must be powered up and initialized in a predefined manner before any normal operation. Automatic initialization is carried out by this state machine. The main processes in initialization is shown in Fig. 3.

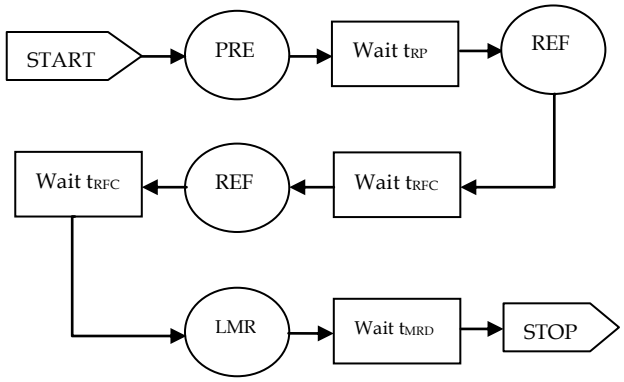

Fig. 3 Initialization Process Flow

Initialization process[4] consist of sequence of operations. First Precharge all banks(PRE) should be done. Precharging is required to deactivate all banks and put them in idle state. After precharging wait for t_{RP} (150 ns) period and execute Auto Refresh(REF). After autorefresh wait for t_{RFC} (495 ns) period and again apply Auto Refresh. Again wait for t_{RFC} period and then Load Mode Register(LMR) command should be issued. With this command

some special attributes are set (i.e. burst packet length , access type, CAS latency and other). Then wait for t_{MRD} (15 ns) period. After the initialization, the SDRAM goes into the idle state, which is the initial state of the main FSM which is explained below.

3.5 Main Finite State Machine

Another state machine is the main FSM or the core of the controller which is shown in Fig. 4. Enable signal of this FSM is generated by the initialization block. This ensures that initialization is complete before normal operation. The FSM generates control signals corresponding to the state. Transition between different states is according to the command input.

The controller awakens in the IDLE state and then changes to Precharge All, Precharge Selected, Load Mode Register, AutoRefresh, or Active depending on the system command. The dashed lines in the state machine diagram indicate the automatic transfer. For Read and Write, the controller first goes into Active state. During active state row address and bank address is decoded from the address input. Controller enters write state when command is 110 and read state when command is 111.

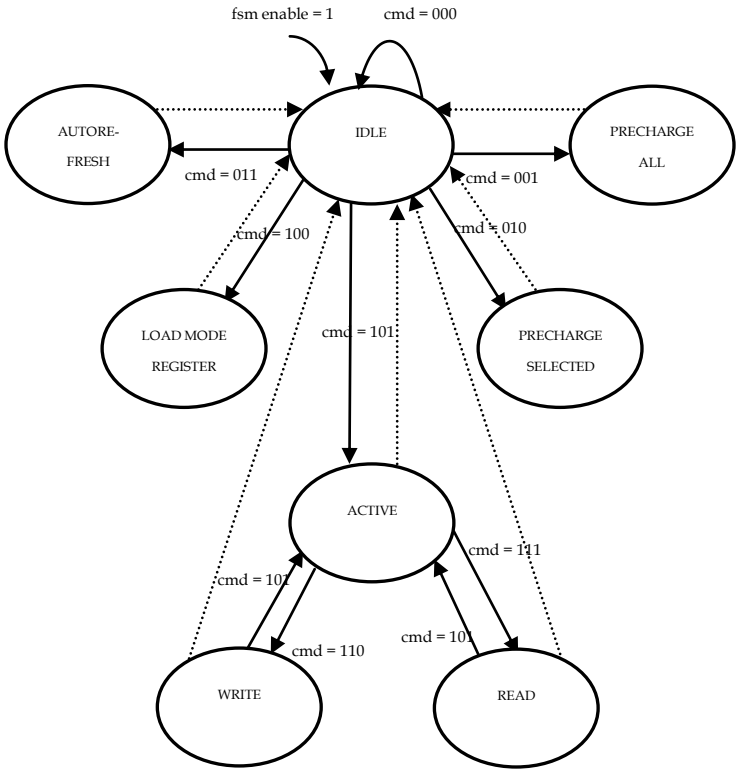

Fig. 4 Main Finite State Machine

3.6 Datapath Circuit

The data flow design between the SDRAM and the system interface is shown in Fig. 5. The datapath circuit consists of four number of 16-bit D flip flops and a tristate buffer. Flip flop is used to shift the data. The data passing through the datapath circuit is either the codeword from the ECC Encoder or codeword from the memory. Tristate buffer is used to determine the direction of bidirectional data bus. Tri state buffer has a control input "oe" which determines whether it is a read or write operation. It is obtained as an output from the main finite state machine.

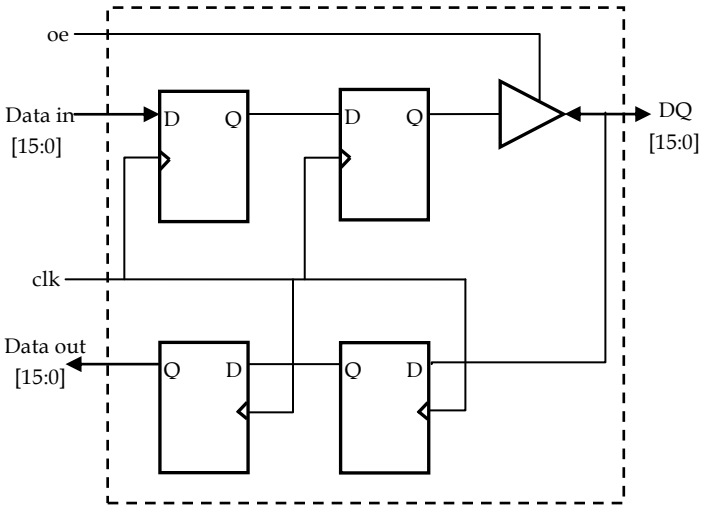

Fig. 5 Datapath Circuit

3.7 Error Correcting Codes (ECC) Block

Hamming Code with additional parity bit can be used to detect and correct single-bit errors and detect double-bit errors [7], [8]. It is 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.

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 [7]. 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 [7]. Refined ECC block comprises an encoder and a decoder-corrector which can detect and correct single-bit errors and detect double-bit errors.

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 [7]. The number of redundancy bits, 'r' to be appended to n-bit data to obtain single error correction is according to equation 1. For example, if the available space for codeword is only 16 bits, then data bit should be only 11 bit wide and the number of redundancy bits will be 5 to obtain single error correction.

$$2^{(r-1)} - 1 \geq n \quad (1)$$

To obtain single error correction and double error detection one extra redundancy bit is required. This bit is all over all parity bit, which checks the parity of all the data bits and the redundancy bits. 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 target memory is 16 bit wide. Therefore sum of data bits and parity bits should not exceed 16 bits. 16-bit codeword can provide error correction and detection to 10-bit data. For an 10-bit message there are 10 possible single-bit errors. Hamming codeword is a concatenation of the original data and the parity bits.

ECC block comprises ECC encoder and ECC decoder-corrector. Encoder converts the 10-bit input message into 16-bit codeword by calculating and placing parity bits at the end positions. This 16-bit codeword is stored in the memory. Decoder- corrector detects any error in the received codeword by calculating syndrome bits and corrects single-bit errors by using a mask. The internal structure of ECC block is given in Fig. 6.

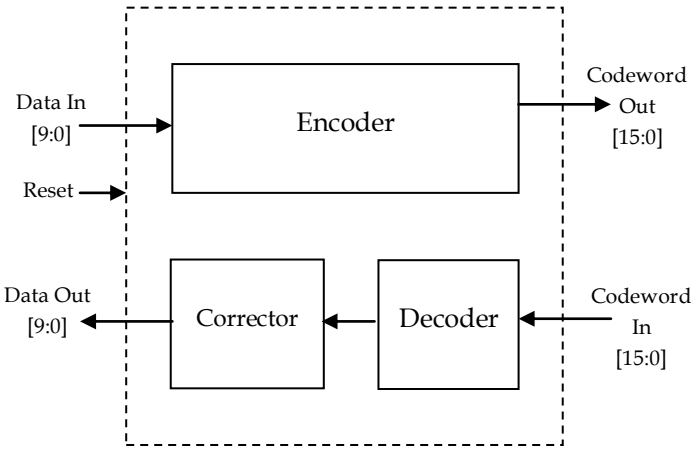

Fig. 6 ECC Block Diagram

ECC Encoder

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 Fig. 7. The parity bits are calculated according to the equations derived from the truth table given in table 2.

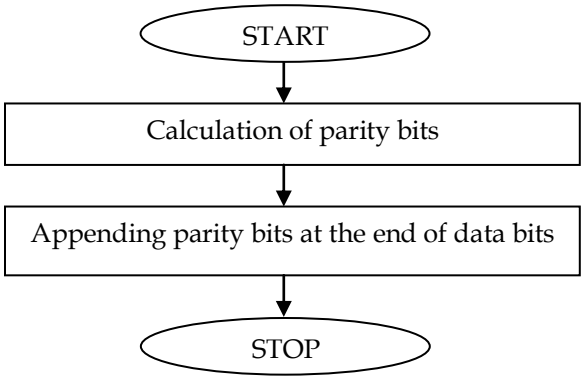

Fig. 7 Encoder Process Flow Chart

TABLE 2
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) [7]. 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 P[5:0] are interspersed in positions 15, 14, 13, 12, 11 and 10 respectively. 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 Fig. 8. Parity bits are shown in bold format. This codeword is transmitted or stored in the memory.

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. 8 Codeword Format

ECC Decoder

Decoder creates syndrome bits through a series of processes. The process flow of decoder is given in Fig. 9.

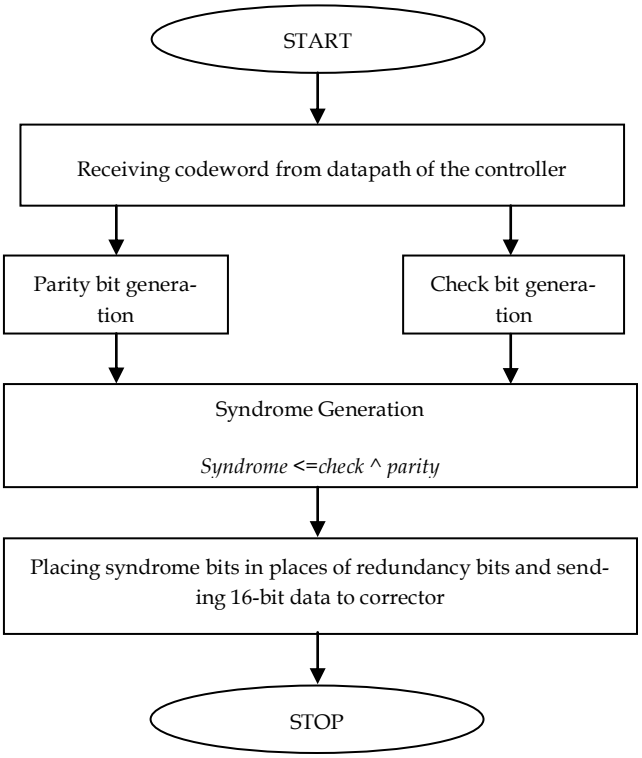

Fig. 9 Decoder Process Flow Chart

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.

During read operation datapath receives data from the memory. Decoder receives this data from datapath. This data represents the codeword corresponding to the actual message. Decoder extracts the redundancy bits (check bits) from the end positions. Then it calculates the parity bits corresponding to the received data. It compares the check bits and the parity bits and generates a syndrome. The syndrome calculation is done according to the equation 2

$$Syndrome = Check \wedge Parity \quad (2)$$

The syndrome bits are placed at the end of the data replacing the parity bits. This data is given to the corrector.

ECC Corrector

The corrector module is responsible for error diagnosis and error correction. Principle of error diagnosis and error correction is explained below [7]. The process flow of corrector is given in Fig. 10.

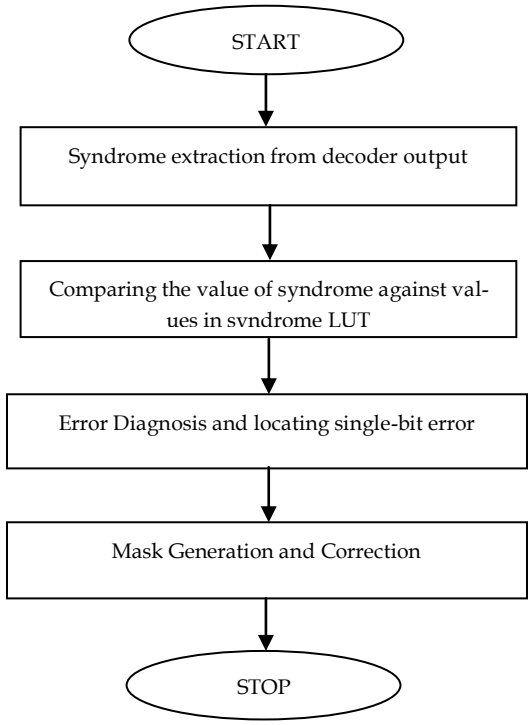

Fig. 10 Corrector Process Flow Chart

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. If S[5:0] is 5'b00000, no error has occurred. If S[5] is "0" and S[4:0] is not equal to "0000", double error has occurred. If S[5] is "1", single error has occurred and the value of S[3:0] indicates the position holding error bit. All the single-bit errors are located and corrected. Double-bit errors are detected and not corrected. Some examples of received codeword and the interpretation of error are illustrated in table 3.

TABLE 3
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

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. For example if single bit error has occurred in 0^{th} position, generated mask will be "0000000001". Separated data bits are XORed with mask to obtain the corrected data according to the equation 3 .

$$Corrected\ Output\ Data = Mask\ \hat{}\ Decoded\ Data \qquad (3)$$

A separate 2-bit output pin is given for the controller to indicate the status of error. Conditions of error output pins and the status of error is given in table 4.

TABLE 4
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 code-word the message is equivalent to the original.
0	1	There are two errors on the code-word no correction have been made.
1	1	Not possible.

Comparison of Improved Hamming Code with Conventional Hamming Code

The error correction using improved Hamming code has reduced the computational complexity [7]. The overhead bits involved in the process of calculation of redundancy bits are lower in improved Hamming code. The table 5 shows the values of overhead bits for both methods and the calculated percentage reduction of overhead bits in improved Hamming code error correction method. Comparison is done using Matlab plot shown in Fig. 11.

TABLE 5
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%

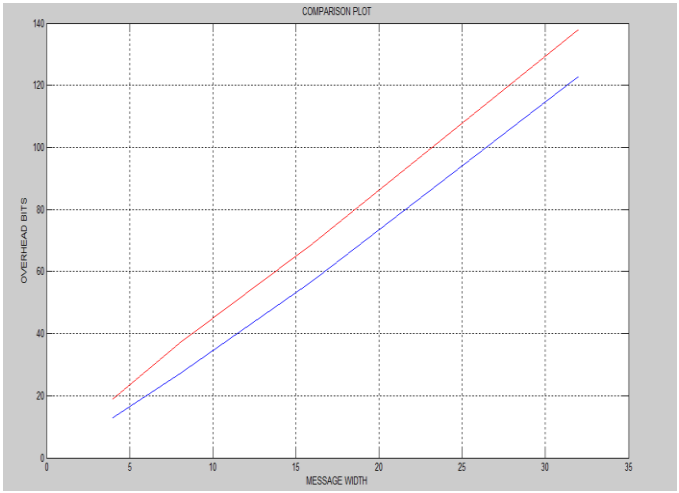

Fig. 11 Comparison Chart

In Fig. 11, 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.

4 SIMULATION RESULTS

The Controller along with memory can be simulated using Modelsim software. The functional simulation result of the wrapper module is shown in Fig. 12. The designed IP core can be further prototyped using Altera DE2 FPGA board[10].

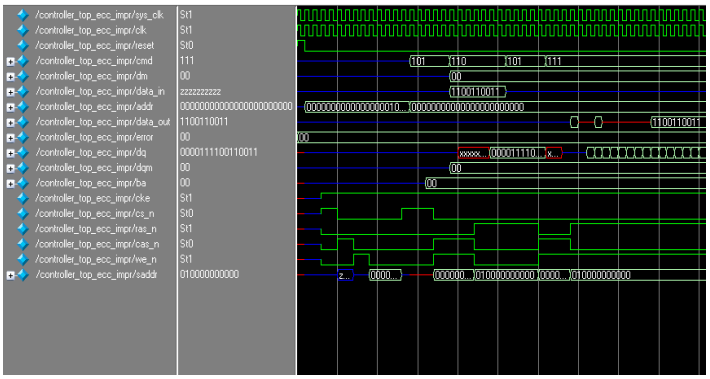

Fig. 12 Simulation result

Note on simulation result :

Reset : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b1, cmd = 3'bxxx, dm = 2'bxx, data_in = 10'bxxxxxxxxxx, addr = 23'bxxxxxxxxxxxxxxxxxxxxxxx, data_out = 10'bzzzzzzzzzz, error = 2'b00)
Initialization : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'bxxx, dm = 2'bxx, data_in = 10'bxxxxxxxxxx, addr=23'b00000000000000000100000, data_out = 10'b zzzzzzzzzz, error = 2'b00)
Activate : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b101, dm = 2'b00, data_in = 10'bxxxxxxxxxx, addr = 23'b00000000000000000000000, data_out = 10'b zzzzzzzzzz, error = 2'b00)
Write : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b110, dm = 2'b00, data_in = 10'b1100110011, addr = 23'b00000000000000000000000, data_out = 10'b zzzzzzzzzz, error = 2'b00)
Activate : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b101, dm = 2'b00, data_in = 10'b zzzzzzzzzz, addr = 23'b00000000000000000000000, data_out = 10'bxxxxxxxxxx, error = 2'b00)
Read : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b111, dm = 2'b00, data_in = 10'bzzzzzzzzzz, addr = 23'b00000000000000000000000, data_out = 10'b1100110011, error = 2'b00)

5 Implementation Results

The controller module is tested by using Altera® DE2 FPGA board. Sys_clk is taken from the board itself. Reset is given as trigger input. Altera Quartus® II is used to synthesis the design. Signal Tap® II Logic Analyzer is used to take hardware test result from FPGA which is shown in Fig. 13. Power analysis and Resource utilization result is listed in table 6 and table 7 respectively.

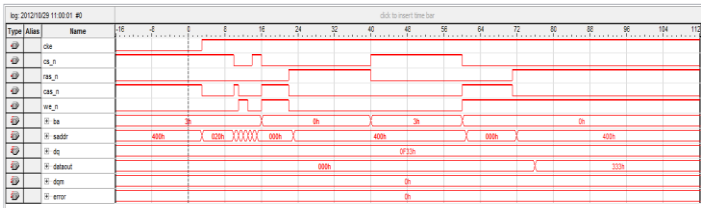

Fig. 13 Hardware test result

Power Analysis table in table 6 gives details about the core dynamic thermal power dissipation, core static thermal power dissipation, I/O thermal power dissipation and total thermal power dissipation.

TABLE 6
Power Analysis Table

PARAMETER	RESULT
Total Thermal Power Dissipation	115.89 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	79.94 mW
I/O Thermal Power Dissipation	35.94 mW

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

TABLE 7
Resource Utilization Table

RESOURCE	USAGE
Estimated Total logic elements	1,376
Total combinational functions	686
Logic element usage by number of LUT inputs	
-- 4 input functions	374
-- 3 input functions	180
-- 2 input functions	132
Logic elements by mode	
-- normal mode	646
-- arithmetic mode	40

Total registers	1117
-- Dedicated logic registers	1117
-- I/O registers	0
I/O pins	51
Total memory bits	6272
Maximum fan-out node	clock
Maximum fan-out	785
Total fan-out	6338
Average fan-out	3.32

CONCLUSION

This paper describes the the design, simulation and characterization of synthesizable SDRAM Controller IP core with built in refined ECC module. Improved Hamming code error correction which is used in this design reduces the overhead bits in the process of calculation of redundancy bits when compared with the conventional Hamming code. The design was developed using Verilog HDL. It can be easily modified for different system design requirements. The proposed design has been tested by implementing the design on Altera DE2 board which uses Cyclone-II device.

REFERENCES

[1] Tomasz Szymanski, Rafak Kiekbik, Andrzej Napieralski, "SDRAM controller for real time digital image processing systems." CAD Systems in Microelectronics, 2001. CADSM 2001. Proceedings of the 6th International Conference.

[2] Qiu Daqiang, Hu Bing, Li Dandan " Design of SDRAM Controller in High-Speed Data Acquisition Based on PCI Bus." The Eighth International Conference on Electronic Measurement and Instruments 2007.

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

[4] MICRON Technology Inc., "Synchronous DRAM: MT48LC8M16A2 Data Sheet."

[5] MICRON Technology Inc., "DOUBLE DATA RATE (DDR) SDRAM: MT46V32M16 Data Sheet."

[6] Xilinx Inc., XAPP134 "Synthesizable High Performance SDRAM Controller." 2000.

[7] 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.

[8] 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.

[9] Baumann, R.C.; , "Radiation-induced soft errors in advanced semi conductor technologies," *Device and Materials Reliability, IEEE Transactions on* , vol.5, no.3, pp. 305- 316, Sept. 2005 doi: 10.1109/TDMR.2005.853449

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