ERROR DETECTION AND CORRECTION USING DECIMAL MATRIX CODE FOR MULTIPLE CELL UPSETS

A PROJECT REPORT

Submitted by

SHANMUGA PRIYA.M **Register. No.:1110107086**

SHRUTHILAYA.M **Register. No.:1110107090**

SINDHU.V **Register. No.:1110107093**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION

ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2015

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report titled **" ERROR DETECTION AND CORRECTION USING DECIMAL MATRIX CODE FOR MULTIPLE CELL UPSETS "** is the bonafied work of " **SHANMUGA PRIYA.M, SHRUTHILAYA.M, SINDHU.V "** who carried out the project work under my supervision.

SIGNATURE

Ms.V.Uma Maheswari, M.E.,
Assistant Professor/E.C.E
Kumaraguru College of Technology
Coimbatore.

SIGNATURE

Dr. Rajeswari Mariappan M.E., Ph.D.,
Head Of The Department
Electronics & Communication Engineering
Kumaraguru College of Technology
Coimbatore.

The candidates with Register numbers 1110107086,1110107090 and 1110107093 are examined by us in the project viva-voce examination held on …….

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First we would like to express our praise and gratitude to the Lord, who has showered his grace and blessing enabling us to complete this project in an excellent manner. He has made all things in beautiful in his time.

We express our sincere thanks to our beloved Joint Correspondent, **Shri. Shankar Vanavarayar** for his kind support and for providing necessary facilities to carry out the project work.

We would like to express our sincere thanks to our beloved Principal **Dr. R. S. Kumar M.E., Ph. D.,** who encouraged us with his valuable thoughts.

We would like to express our sincere thanks and deep sense of gratitude to our HOD, **Dr.Rajeswari Mariappan M.E., Ph. D.,** for her valuable suggestions and encouragement which paved way for the successful completion of the project.

We are greatly privileged to express our deep sense of gratitude to the Project Coordinator **Ms.A.KALAISELVI M.E., (Ph. D)**, Assistant Professor, for her continuous support throughout the course.

In particular, We wish to thank and express our everlasting gratitude to the Supervisor **Ms.V.UMA MAHESWARI M.E.,** Assistant Professor for her expert counselling in each and every step of project work and we wish to convey our deep sense of gratitude to all teaching and non-teaching staff members of ECE Department for their help and cooperation.

Finally, we thank our parents and our family members for giving us the moral support in all of our activities and our dear friends who helped us to endure our difficult times with their unfailing support and warm wishes

ABSTRACT

Transient Multiple Cell Upsets (MCUs) are becoming major issues in the reliability of memories exposed to radiation environment. To prevent MCUs from causing data corruption, more complex error correction codes (ECCs) are widely used to protect memory, but the main problem is that they would require higher delay overhead. Recently, Matrix Codes (MCs) based on Hamming codes have been proposed for memory protection. The main issue is that they are double error correction codes and the error correction capabilities are not improved in all cases.

In this project, novel Decimal Matrix Code (DMC) based on divide-symbol is proposed to enhance memory reliability with lower delay overhead in the Xilinx software with the help of Verilog code. The proposed DMC utilizes decimal algorithm to obtain the maximum error detection capability. Moreover, the Encoder Reuse Technique (ERT) is proposed to minimize the area overhead of extra circuits without disturbing the whole encoding and decoding processes. ERT uses DMC encoder itself to be part of the decoder. The proposed DMC is compared to well-known codes such as the existing Hamming, MCs, and Punctured Difference Set (PDS) codes.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER -1

INTRODUCTION

CHAPTER -1

INTRODUCTION

1.1 PROJECT OVERVIEW:

The CMOS technology scales down to nanoscale and memories are combined with an increasing number of electronic systems. The soft error rate in memory cells is rapidly increasing, especially when memories operate in space environments due to ionizing effects of atmospheric neutron , alpha-particle and cosmic rays. Although single bit upset is a major concern about memory reliability , Multiple Cell Upsets (MCUs) have become a serious reliability concern in some memory application.

Now a days to maintain good level of reliability, it is necessary to protect memory cells using protection codes. For this purpose, various error detection and correction methods are being used. In this project 32-bit Decimal Matrix Code was proposed to assure the reliability of memory. Here to detect and correct up to consecutive 16 errors , the proposed protection code utilized decimal procedure to detect errors , so that more errors were detected and corrected . The results showed that the proposed scheme has a protection level against large MCUs in memory.

In this project, at first the binary values of an image is first found out using matlab and these binary values are given as input to the algorithm which is written using verilog code.

After completing the encoder and the decoder part in the algorithm , the errors in the bits are corrected and these corrected binary values are converted into the original image which is error-free.The general flow diagram of the project is expressed by the schematic below:

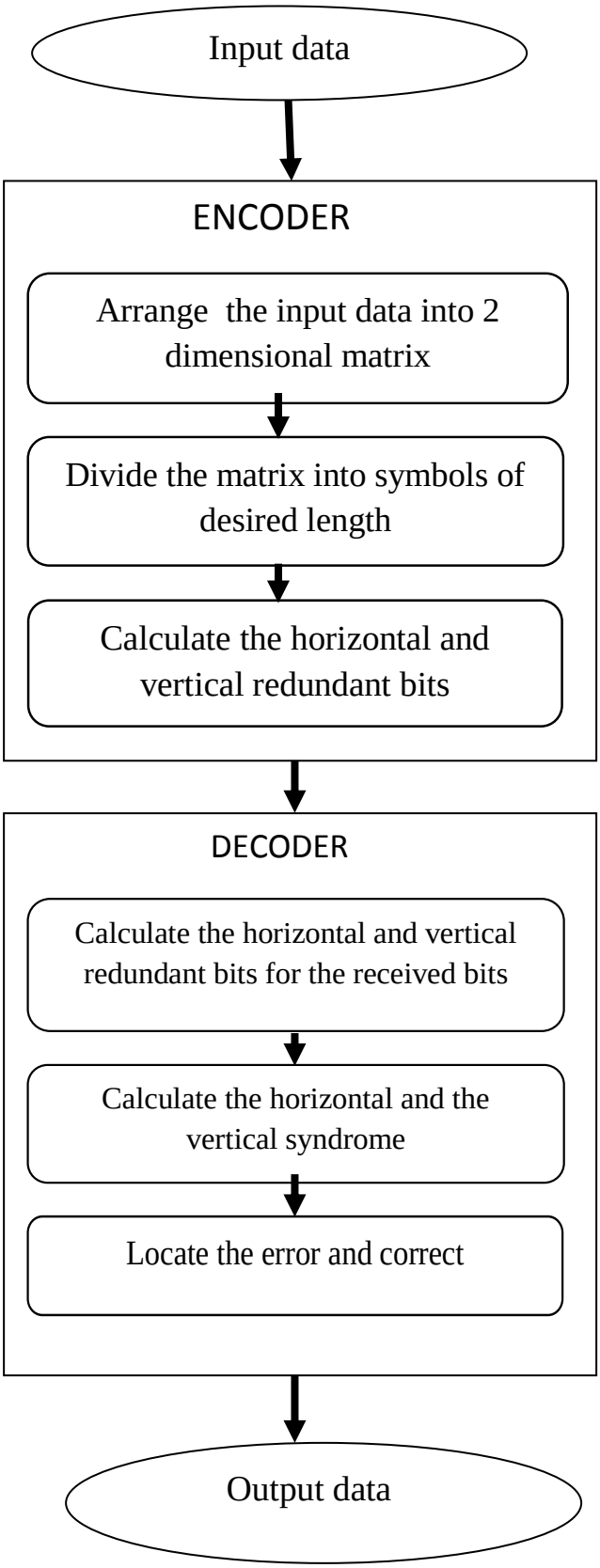


FIG.1.1 GENERAL FLOW DIAGRAM

1.2 DECIMAL MATRIX CODE (DMC):

1.2.1 INTRODUCTION

The general idea for achieving error detection and correction is to add some redundant bits (i.e., some extra data) to a message, which receiver can use to check consistency of the delivered message, and to pick up data determined to be corrupt. Error-detection and correction scheme can be either systematic or non-systematic: In a systematic scheme, the transmitter sends the unique data, and attaches a fixed number of redundant bits, based on particular logic. If only the error detection is required, a receiver can check the same logic to the received data bits and compare its output with the receive check bits; if the values do not match, an error has occurred at some point throughout the transmission. Different types of codes used for Error detection and correction.

In a system to use a non-systematic code, the message is transformed into an encoded message that has atleast as many bits as that message. Error detection and correction is used to reduce the soft errors. Several techniques are used in gate upsets in memories. For example, the Bose– Chaudhuri–Hocquenghem codes, Reed–Solomon codes, punctured difference set (PDS) codes, and matrix codes has been used to contact with MCUs in memories. Which requires more area, power, and delay overheads since the encoding and decoding circuits are more complex in these complicated codes.

Reed-Muller code is another protection code that is able to detect and correct additional error than a Hamming code. The main drawback of this protection code is its more area and power requirements. Hamming Codes are more used to correct Single Error Upsets (SEU's) in memory due to their ability to correct single errors through reduced area and performance overhead. Though brilliant for correction of single errors in a data, they cannot correct double bit errors. One more class of SEC-DED codes

proposed to detect any number of errors disturbing a single byte. These codes are additional suitable than the conventional SEC-DED codes for protecting the byte-organized memories. Though they operate through lesser overhead and are good for multiple error detection, they cannot correct more errors. There are additional codes such as the single-byte-error-correcting, double-byte-error-detecting (SBCDBD) codes that can correct multiple errors.

The Single-error-correcting, Double-error-detecting and Double-adjacent-error-correcting (SEC-DED-DAEC) code provides a low cost ECC methodology to correct adjacent errors. The only drawback through this code is the possibility of miss-correction for a small subset of many errors. As CMOS technology scales down to nanoscale and memories are combined through an increasing number of electronic systems, the soft error rate in memory cells is increase, especially when memories operate in space environments due to ionizing effects of atmosphere.

1.3 SOFT ERRORS:

In electronic computing, a soft error is a type of error where a signal or datum is wrong. Errors may be caused by a defect, usually understood either to beam is take in designer construction, or a broken component. A soft error is also a signal or datum which is wrong, but is not assumed to imply such a mistake or breakage. After observing a soft error ,there is no implication that the system is any less reliable than before. In the space craft industry this kind of error is called a single-event upset. In a computer's memory system, a soft error changes an instruction in a program or a data value. Soft errors typically can be remedied by cold booting the computer. A soft error will not damage a system's hardware; the only damage is to the data that is being processed. There are two types of soft errors, chip-level soft error and system-level soft error. Chip-level soft errors occur when the radioactive atoms in the chip's material

decay and release alpha particles into the chip. Because an alpha particle contains a positive charge and kinetic energy, the particle can hit a memory cell and cause the cell to change state to a different value. The atomic reaction is so tiny that it does not damage the actual structure of the chip.

System-level soft errors occur when the data being processed is hit with a noise phenomenon, typically when the data is on a databus. The computer tries to interpret the noise as a data bit, which can cause errors in addressing or processing program code. The bad data bit can even be saved in memory and cause problems at a later time. If detected, a soft error may be corrected by rewriting correct data in place of erroneous data. Highly reliable systems use error correction to correct soft errors on the fly. However, in many systems, it may be impossible to determine the correct data, or even to discover that an error is present at all. In addition, before the correction can occur, the system may have crashed, in which case the recovery procedure must include a reboot. Soft errors involve changes to data: the electrons in a storage circuit, but no changes to the physical circuit itself, the atoms. If the data is rewritten, the circuit will work perfectly again. Soft errors can occur on transmission lines, in digital logic, analog circuits, magnetic storage, and elsewhere, but are most commonly known in semiconductor storage**.**

1.3.1 CAUSES OF SOFT ERRORS

1.3.1(a) ALPHA PARTICLES FROM PACKAGE DECAY

Soft errors became widely known with the introduction of dynamic RAM in the 1970s. In these early devices, chip packaging materials contained small amounts of radioactive contaminants. Very low decay rates are needed to avoid excess soft errors, and chip companies have occasionally suffered problems with contamination ever since. It is extremely hard to maintain the material purity needed. Controlling alpha particle

emission rates for critical packaging materials to less than a level of 0.001 counts per hour percm^2 (cph/cm^2) is required for reliable performance of most circuits. For comparison, the count rate of a typical shoe's soles between 0.1 and10cph/cm^2.Package radioactive decay usually causes a soft error by alpha particle emission. The positively charged alpha particle travels through the semiconductor and disturbs the distribution of electrons there. If the disturbance is large enough, a digital signal can change from 0 to 1 or vice versa. In combinational logic, this effect is transient, perhaps lasting a fraction of a nano second and this has led to the challenge of soft errors in combinational logic mostly going unnoticed. In sequential logic such as latches and RAM, even this transient upset can become stored for an indefinite time, to be read out later. Thus, designers are usually much more aware of the problem in storage circuits.

1.3.1(b) COSMIC RAYS

Once the electronics industry had determined how to control package contaminants, it became clear that other causes were also at work. James F. Ziegler led a program of work at IBM which culminated in the publication of a number of papers (ZieglerandLanford,1979) demonstrating that cosmic rays also could cause soft errors. Indeed, in modern devices, cosmic rays may be the predominant cause. Although the primary particle of the cosmic ray does not generally reach the Earth's surface, it creates a shower of energetic secondary particles. At the Earth's surface approximately 95% of the particles capable of causing soft errors are energetic neutrons with the remainder composed of protons and pions. IBM estimated in 1996 that one error per month per 256MiB of RAM was expected for a desktop computer. This flux of energetic neutrons is typically referred to as "cosmic rays" in the soft error literature. Neutrons are uncharged and cannot disturb a circuit on their own, but undergo neutron capture by the nucleus of an atom in a chip. This process may result in the production of charged secondaries, such as alpha particles and oxygen nuclei, which can then cause soft errors.

1.3.1(c) THERMAL NEUTRONS

Neutrons that have lost kinetic energy until they are in thermal equilibrium with their surroundings are an important cause of soft errors for some circuits. At low energies many neutron capture reactions become much more probable and result in fission of certain materials creating charged secondaries as fission byproducts. This nuclear reaction is an efficient producer of an alpha particle, 7Li nucleus and gamma ray. Either of the charged particles (alpha or 7Li) may cause a soft error if produced in very close proximity, approximately 5μm, to a critical circuit node.

For applications in medical electronic devices, this soft error mechanism may be extremely important. Neutrons are produced during high energy cancer radiation therapy using photon beam energies above 10 MeV. These neutrons are moderated as they are scattered from the equipment and walls in the treatment room resulting in a thermal neutron flux that is about 40×10^{6} higher than the normal environmental neutron flux. This high thermal neutron flux will generally result in a very high rate of soft errors and consequent circuit upset.

1.3.1(d) OTHER CAUSES

Soft errors can also be caused by random noise or signal integrity problems, such as inductive or capacitive crosstalk. However, in general, these sources represent a small contribution to the overall soft error rate when compared to radiation effects.

1.4 EXISTING TECHNIQUES FOR ERROR DETECTION AND CORRECTION FOR MULTIPLE CELL UPSETS (MCUs):

Interleaving technique has been used to restrain MCUs. However, interleaving technique may not be practically used in Content-Addressable Memory (CAM), because of the tight coupling of hardware structures from both cells and comparison circuit structures.

More recently, 2-D Matrix Codes (MCs) are proposed to efficiently correct MCUs per word with a low decoding delay, in which one word is divided into multiple rows and multiple columns in logical. The bits per row are protected by Hamming code, while parity code is added in each column. For the MC based on Hamming, when two errors are detected by Hamming, the vertical syndrome bits are activated so that these two errors can be corrected. As a result, MC is capable of correcting only two errors in all cases. In, an approach that combines decimal algorithm with Hamming code has been conceived to be applied at software level. It uses addition of integer values to detect and correct soft errors. The results obtained have shown that this approach have a lower delay overhead over other codes.

In this project, novel Decimal Matrix Code (DMC) based on divide-symbol is proposed to provide enhanced memory reliability. The proposed DMC utilizes decimal algorithm (decimal integer addition and decimal integer subtraction) to detect errors. The advantage of using decimal algorithm is that the error detection capability is maximized so that the reliability of memory is enhanced. Besides, the Encoder-Reuse Technique (ERT) is proposed to minimize the area overhead of extra circuits (encoder and decoder) without disturbing the whole encoding and decoding processes, because ERT uses DMC encoder itself to be part of the decoder

CHAPTER 2

ENCODER AND DECODER OF DECIMAL MATRIX CODE

CHAPTER 2

ENCODER AND DECODER OF DECIMAL MATRIX CODE

2.1 INTRODUCTION:

The decimal matrix code is divided into the encoder and the decoder part for its working. In the decimal matrix code, a special feature called the Encoder Re-Use Technique (ERT) is used. In the encoding (write) process, the DMC encoder is only an encoder to execute the encoding operations. However, in the decoding (read) process, this encoder is employed for computing the syndrome bits in the decoder. These clearly show how the area overhead of extra circuits can be substantially reduced. The detailed explanation of the processes that are taking place in the encoder and the decoder part are explained below:

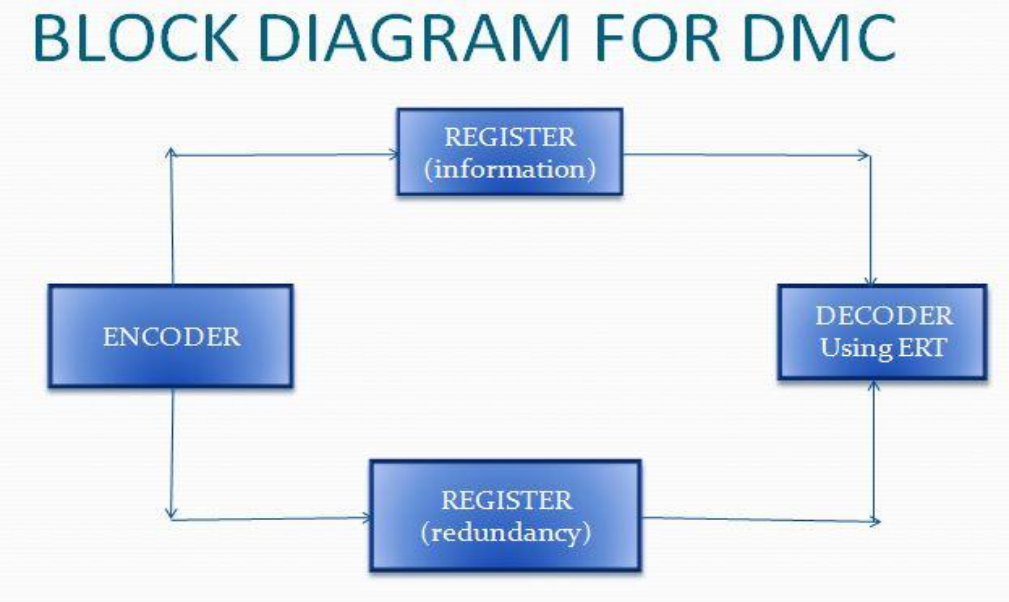


FIG 2.1 BLOCK DIAGRAM OF DMC

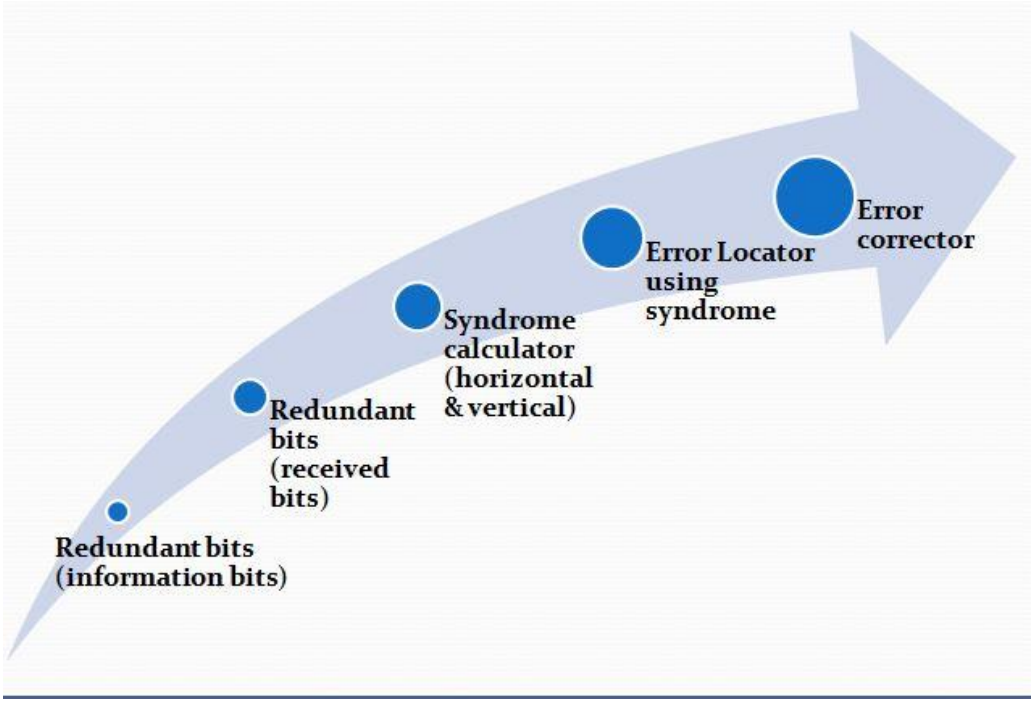
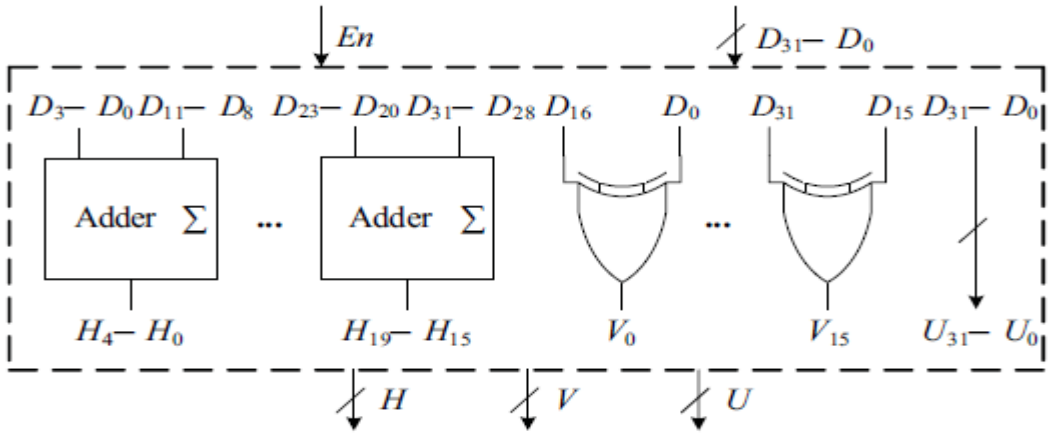


FIG 2.2 PROCESS OF DECIMAL MATRIX CODE:

2.2 DMC ENCODER:

In the proposed DMC, first, the divide-symbol and arrange-matrix ideas are performed, i.e., the N-bit word is divided into 'k' symbols of 'm' bits (N = k × m), and these symbols are arranged in a k1 ×k2 2-D matrix (k = k1 ×k2, where the values of k1 and k2 represent the numbers of rows and columns in the logical matrix respectively). Second, the horizontal redundant bits H are produced by performing decimal integer addition of selected symbols per row. Here, each symbol is regarded as a decimal integer. Third, the vertical redundant bits V are obtained by binary operation among the bits per column. It should be noted that both divide-symbol and arrange-matrix are implemented in logical instead of in physical. Therefore, the proposed DMC does not require changing the physical structure of the memory.



D0-D31- 32 bit information

H0-H19- Horizontal redundant bits

V0-V15- Vertical redundant bits

U0-U15- Copied Information bits

FIG 2.3 BLOCK DIAGRAM OF ENCODER OF DMC

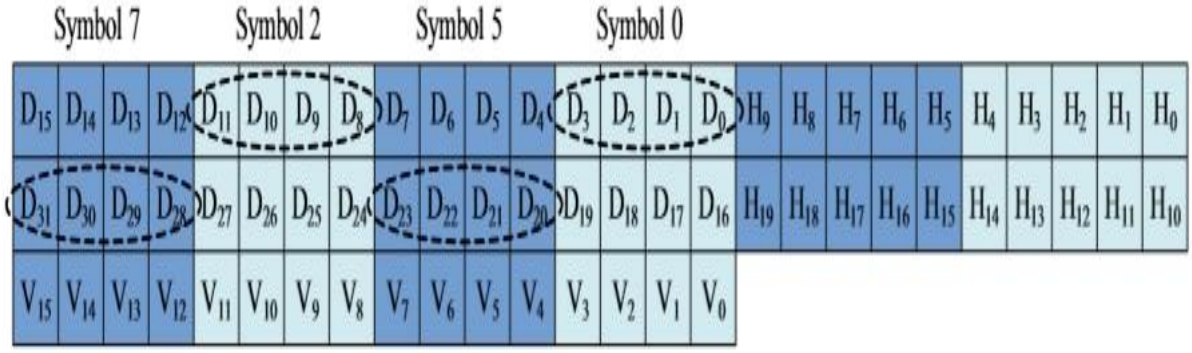


FIG 2.4 FORMAT OF CODE WORD

The horizontal redundant bits H can be obtained by decimal integer addition as follows:

$$H4H3H2H1H0 = D3D2D1D0 + D11D10D9D8 \quad (1)$$

$$H9H8H7H6H5 = D7D6D5D4 + D15D14D13D12 \quad (2)$$

and similarly for the horizontal redundant bits H14H13H12H11H10 and H19H18H17H16H15, where "+" represents decimal integer addition. For the vertical redundant bits V, we have

$$V0 = D0 \oplus D16 \quad (3)$$

$$V1 = D1 \oplus D17 \quad (4)$$

and similarly for the rest vertical redundant bits.

2.3 DMC DECODER:

To obtain a word being corrected, the decoding process is required. For example, first, the received redundant bits H4H3H2H1H0' and V'0−V'3 are generated by the received information bits D'. Second, the horizontal syndrome bits ΔH4H3H2H1H0 and the vertical syndrome bits S3 −S0 can be calculated as follows:

$$\Delta H4H3H2H1H0 = H4H3H2H1H0' - H4H3H2H1H0 \quad (5)$$

$$S0 = V'0 \oplus V0 \quad (6)$$

and similarly for the rest vertical syndrome bits, where "–" represents decimal integer subtraction. When ΔH4H3H2H1H0 and S3 −S0 are equal to zero, the stored codeword has original information bits in symbol 0 where no errors occur. When ΔH4H3H2H1H0 and S3 − S0 are nonzero, the induced errors (the number of errors is 4 in this case) are detected and located in symbol 0, and then these errors can be corrected by

$$D0correct = D0 \oplus S0 \quad (7)$$

The proposed DMC decoder, which is made up of the following submodules, and each executes a specific task in the decoding process: syndrome calculator, error locator,

and error corrector. It can be observed from this figure that the redundant bits must be recomputed from the received information bits D and compared to the original set of redundant bits in order to obtain the syndrome bits ΔH and S. Then error locator uses ΔH and S to detect and locate which bits some errors occur. Finally, in the error corrector, these errors can be corrected by inverting the values of error bits. In the proposed scheme, the circuit area of DMC is minimized by reusing its encoder. This is called the ERT. The ERT can reduce the area overhead of DMC without disturbing the whole encoding and decoding processes. It can be observed that the DMC encoder is also reused for obtaining the syndrome bits in DMC decoder. Therefore, the whole circuit area of DMC can be minimized as result of using the existent circuits of encoder.

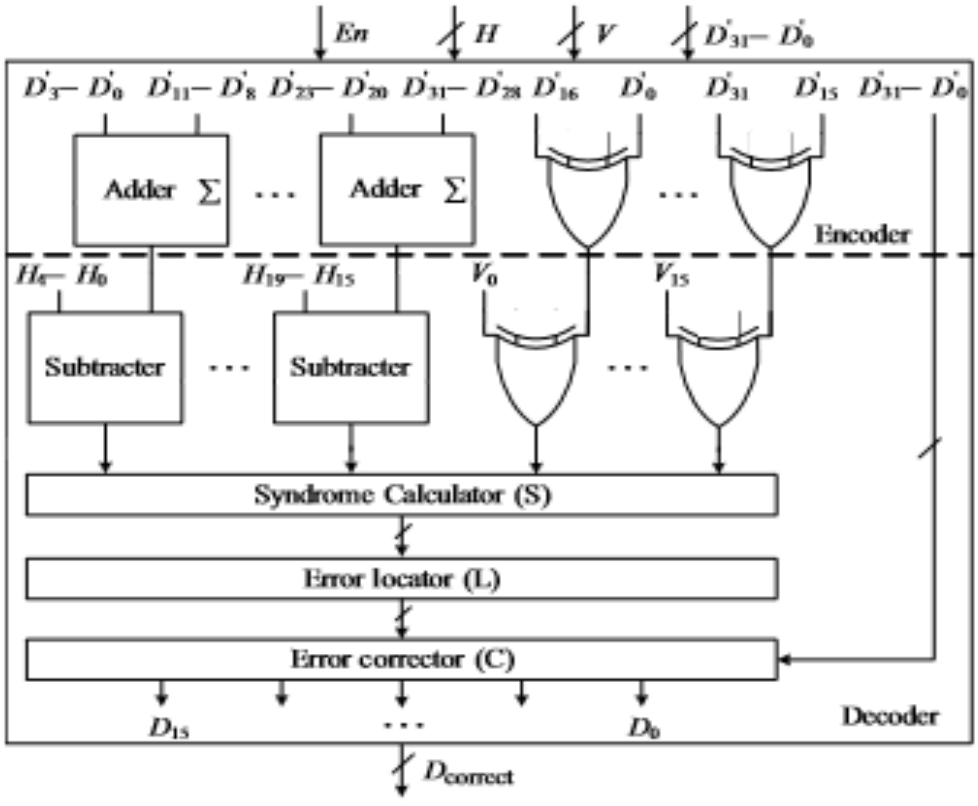


FIG 2.5 BLOCK DIAGRAM OF DECODER OF DMC

CHAPTER -3

HAMMING CODE

CHAPTER -3

HAMMING CODE

3.1 INTRODUCTION

Hamming code is an error correction code that can be used to detect single and double-bit errors and correct single-bit errors that can occur when binary data is transmitted from one device into another.

3.2 DESIGNING (n,k,t) HAMMING CODE

The (n, k, t) code refers to an ‘n’-bit code word having ‘k’ data bits (where n > k) and ‘r’ (=n–k) error-control bits called ‘redundant’ or ‘redundancy’ bits with the code having the capability of correcting ‘t’ bits in the error (i.e., ‘t’ corrupted bits). If the total number of bits in a transmittable unit (i.e., code word) is ‘n’ (=k+r), ‘r’ must be able to indicate at least ‘n+1’ (=k+r+1) different states.

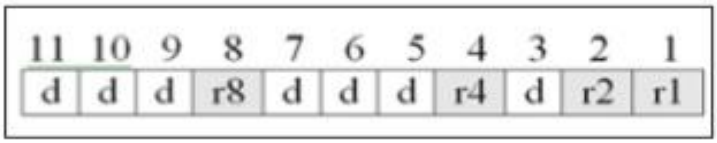


FIG 3.1 POSITION OF REDUNDANT BITS

Of these, one state means no error, and ‘n’ states indicate the location of an error in each of the ‘n’ positions. So ‘n+1’ states must be discoverable by ‘r’ bits; and ‘r’ bits can indicate 2r different states. Therefore, 2r must be equal to or greater than ‘n+1’: 2r e” n +1 or 2r e” k + r +1 The value of ‘r’ can be determined by substituting the value of ‘k’ (the original length of the data to be transmitted). For example, if the value of ‘k’ is ‘7,’ the smallest ‘r’ value that can satisfy this constraint is ‘4’: 24 e” 7+4+1 The (11, 7, 1) Hamming code can be applied to data units of any length. It uses the relationship

between data and redundancy bits discussed above, and has the capability of correcting single-bit errors. For example, a 7-bit ASCII code requires four redundancy bits that can be added at the end of the data unit or interspersed with the original data bits to form the (11, 7, 1) Hamming code. In the figure, these redundancy bits are placed in positions 1, 2, 4 and 8 (the positions in an 11-bit sequence that are powers of '2'). For clarity in the examples below, these bits are referred to as 'r1,' 'r2,' 'r4' and 'r8.' In the Hamming code, each 'r' bit is the parity bit for one combination of data bits as shown below:

r1: bits 1, 3, 5, 7, 9, 11

r2: bits 2, 3, 6, 7, 10, 11

r4: bits 4, 5, 6, 7

r8: bits 8, 9, 10, 11

Each data bit may be included in more than one calculation. In the sequences above, for example, each of the original data bits is included in at least two sets, while the ' r ' bits are included in only one set.

3.3 CALCULATION OF 'r' VALUES.

In the first step, each bit of the original character is placed in its appropriate position in the 11-bit unit. In the subsequent steps, the even parities for the various bit combinations are calculated. The parity value for each combination is the value of the corresponding 'r' bit.

Error detection and correction.

Suppose that by the time the above transmission is received, the seventh bit has changed from '1' to '0.'

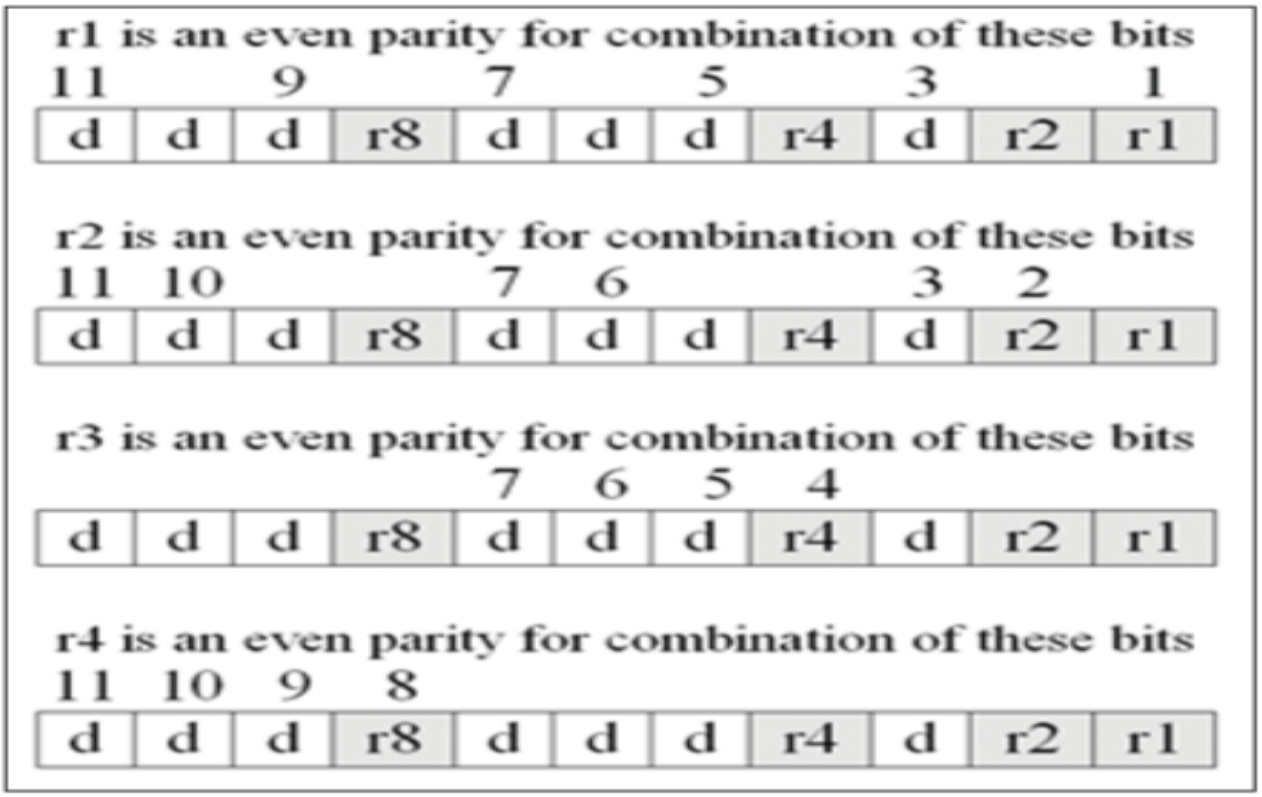


FIG 3.2 REDUNDANT BITS CALCULATION

The receiver takes the transmission and recalculates four new parity bits, using the same sets of bits used by the sender plus the relevant parity 'r' bit for each set .Then it assembles the new parity values into a binary number in the descending order of 'r' position (r8, r4, r2, r1). In the given example, this step gives us the binary number '0111' ('7' decimal), which is the precise location of the corrupted bit. Once the bit is identified, the receiver can complement its value and correct the error. The beauty of the technique is that it can be easily implemented in hardware and the code is corrected before the receiver knows about it**.**

CHAPTER -4

COMPARISON OF DECIMAL MATRIX CODE WITH HAMMING CODE

CHAPTER -4

COMPARISON OF DECIMAL MATRIX CODE WITH HAMMING CODE

4.1 INTRODUCTION:

When comparing the decimal matrix code with the Hamming code, more number of errors can be detected and corrected through Decimal Matrix Code while Hamming code can detect only single and double bit errors and correct only single bit errors.

The area requirement is minimised in the case of Decimal matrix code while it is quite large in the case of Hamming code.The time delay is more in the Hamming code while it is very less with the Decimal Matrix Code.

4.2 TIME DELAY IN DMC AND HAMMING CODE:

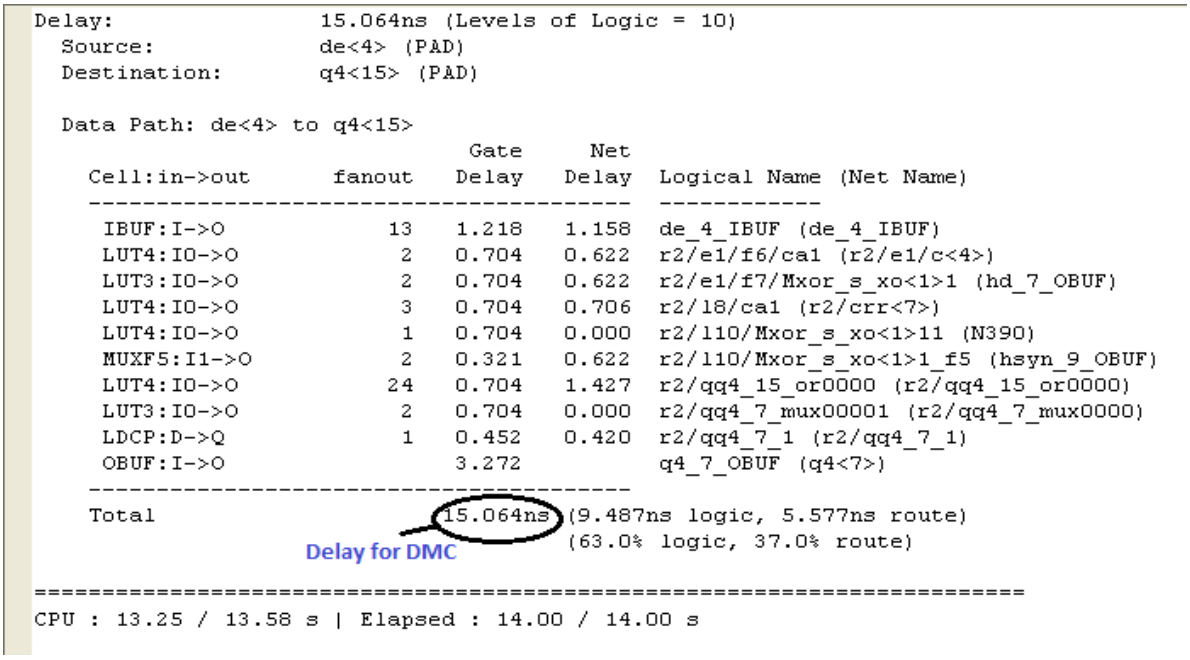


FIG 4.1 DELAY FOR 16 BIT ERROR CORRECTION (32BIT DATA)IN DMC

```
--------------------------------------------------------------------------------
Delay:               11.579ns (Levels of Logic = 7)
  Source:            in1<3> (PAD)
  Destination:       dout<2> (PAD)

  Data Path: in1<3> to dout<2>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             7   1.218   0.787  in1_3_IBUF (in1_3_IBUF)
     LUT3:I1->O            2   0.704   0.526  e1/Mxor_mux0000_xor0001_xo<1>1 (e1/mux0000_xor0001)
     LUT4:I1->O            3   0.704   0.535  e1/out_0_mux00001 (eout_0_OBUF)
     LUT4:I3->O            1   0.704   0.595  e2/Mxor__old_r1_4_xo<4>1_SW0 (N160)
     LUT4:I0->O            3   0.704   0.706  e2/Mxor__old_r1_4_xo<4>1 (e2/_old_r1_4)
     LUT4:I0->O            1   0.704   0.420  e2/out_2_mux0001 (dout_2_OBUF)
     OBUF:I->O                 3.272          dout_2_OBUF (dout<2>)
    ----------------------------------------
    Total                     11.579ns (8.010ns logic, 3.569ns route)
                                       (69.2% logic, 30.8% route)
                      Delay for Hamming
                      Code
================================================================================
```

FIG 4.2 DELAY FOR SINGLE BIT ERROR CORRECTION IN HAMMING CODE

From the two figures (fig 4.1 & 4.2) mentioned above it is observed that the delay for the 32 bit DMC code correcting the 16 bit errors is found to be 15.064ns where for the 7 bit hamming code correcting a single bit is found to be 11.579ns. From these data, it is confirmed that the proposed Decimal Matrix Code is highly efficient than the Hamming code when the delay factor is considered.

4.3 POWER CONSUMPTION IN DMC AND HAMMING CODE:

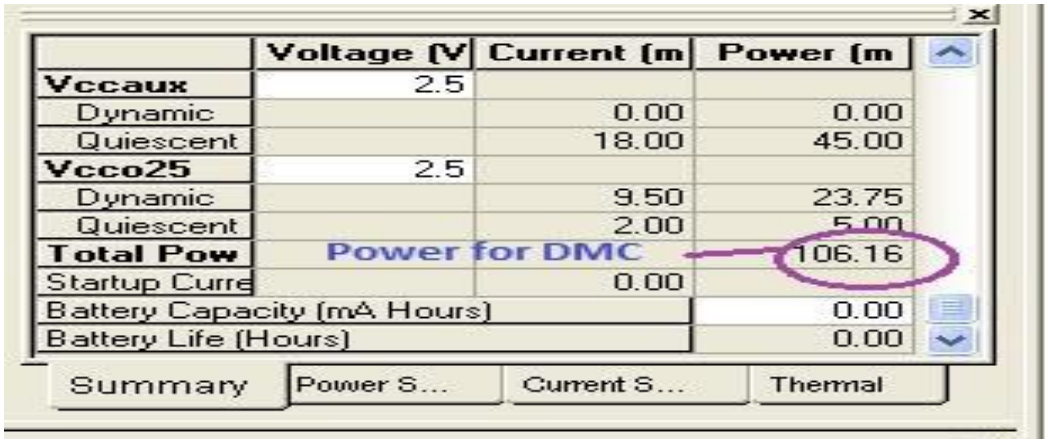


FIG 4.3 POWER CONSUMPTION FOR DMC

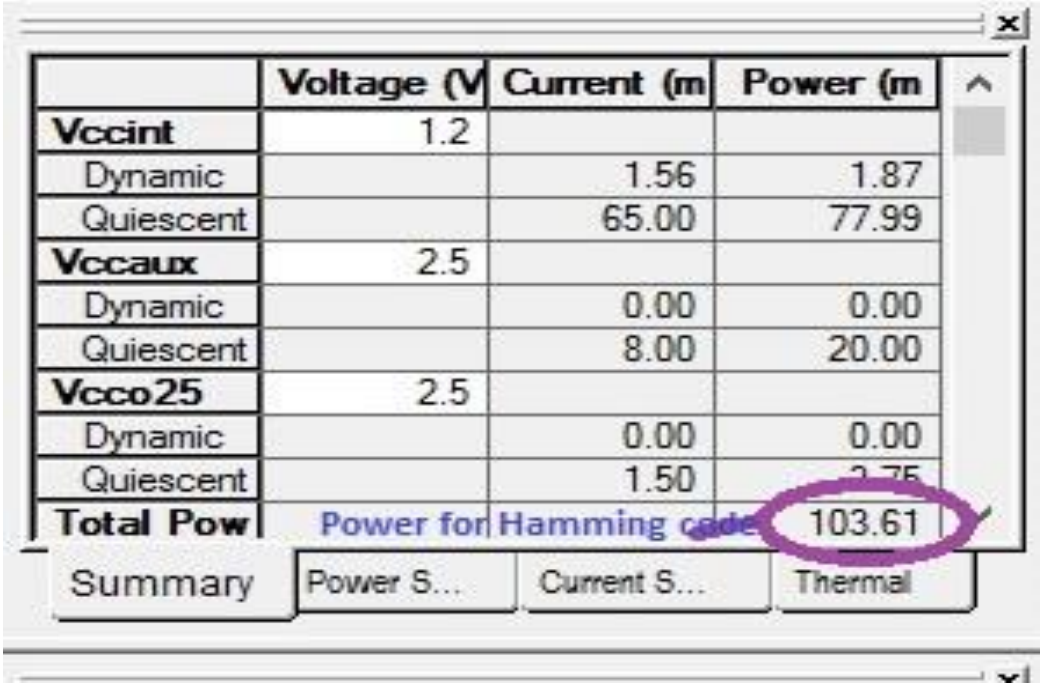


FIG 4.4 POWER CONSUMPTION FOR HAMMING CODE

The power consumption for the DMC Code is found to be 106mW for the 32 bit code correcting 16 bits of errors and at the same time the power consumed by the Hamming code for correcting a single error is 104mW . In this way Decimal Matrix Code(DMC) is more efficient than the Hamming code.

CHAPTER -5

ERROR DETECTION AND CORRECTION IN IMAGE USING MATLAB AND XILINX

CHAPTER -5

ERROR DETECTION AND CORRECTION IN IMAGE USING MATLAB AND XILINX

5.1 PROCESSES INVOLVED IN THE APPLICATION USING MATLAB AND XILINX:

In this project at first an image is converted into its equivalent decimal values using the suitable commands in Matlab.These values are written in an Excel sheet.Then each of the decimal value is converted into its equivalent binary value in the excel sheet which can be read out in the Matlab.These binary values from the excel sheet are given as the input to the verilog code which is meant for error detection and correction.

The input from the excel sheet is given to the encoder part of the decimal matrix code. In the encoder part ,the horizontal and the vertical redundant bits are calculated first and they are stored in a register.

When it comes to the decoder part, for a 32 bit data 16 bits are made corrupted and the horizontal and the vertical redundant bits are calculated for the data with corrupted bits.After calculating the redundant bits ,the syndrome is calculated by using the redundant bits calculated in the encoder part and in the decoder part. In the syndrome calculation both the horizontal and the vertical syndrome is calculated.

Once the syndrome calculated is found to be non-zero,then the occurrence of error is confirmed and the error is corrected by simply locating the position of the corrupted bit and just flipping the bit.

Again these inputs which are corrected are then written in an Excel sheet .This excel sheet is read in Matlab and the original image is shown.

CHAPTER -6

SIMULATION RESULTS

CHAPTER -6

SIMULATION RESULTS

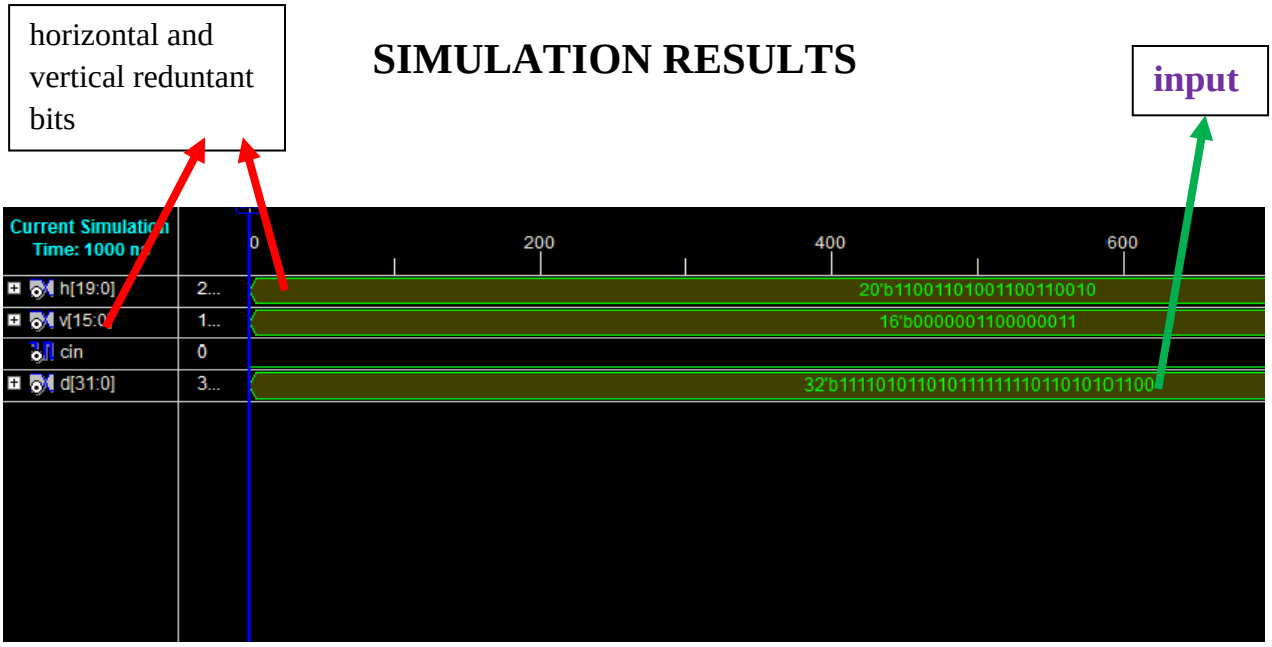


FIG 6.1 32 BIT ENCODER

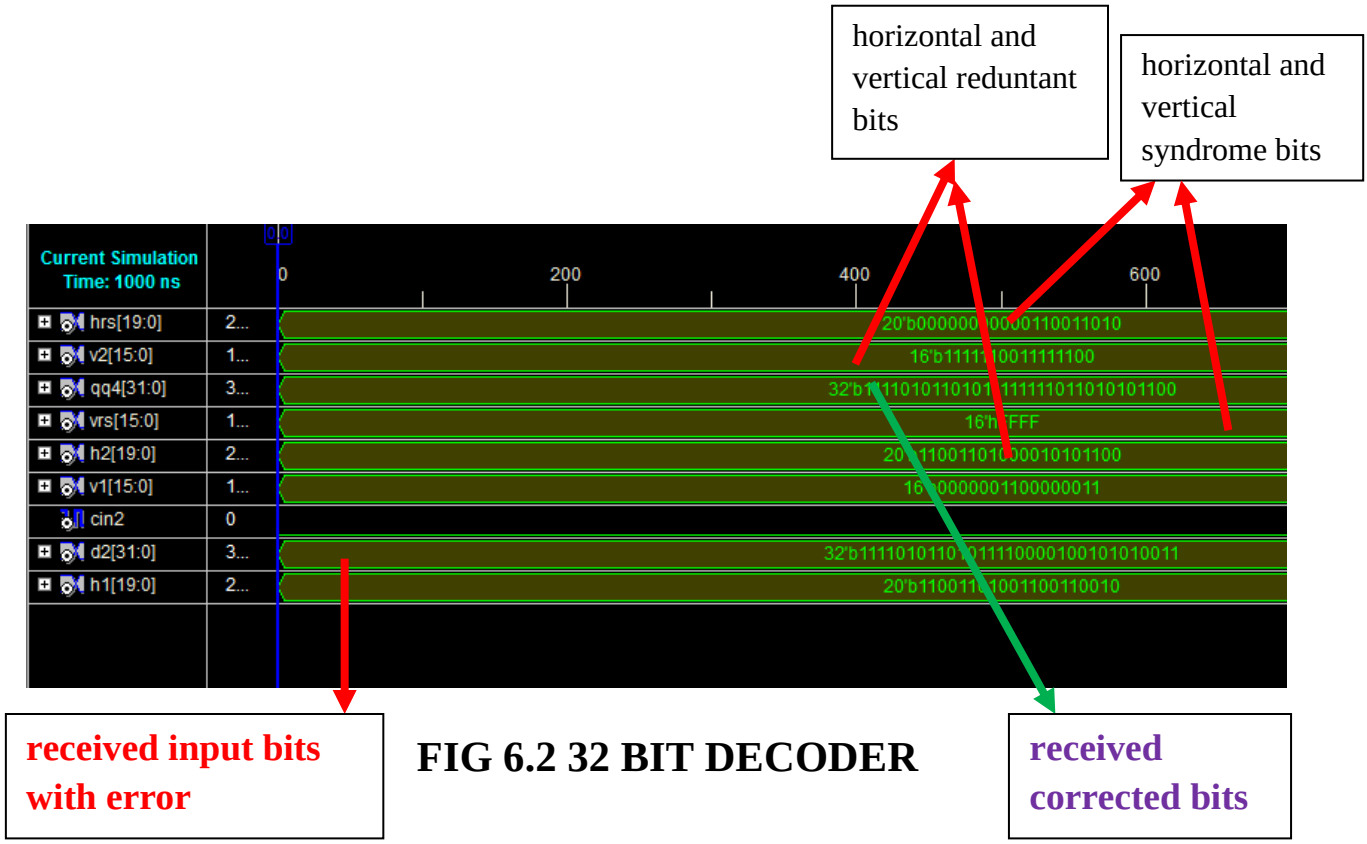


FIG 6.2 32 BIT DECODER

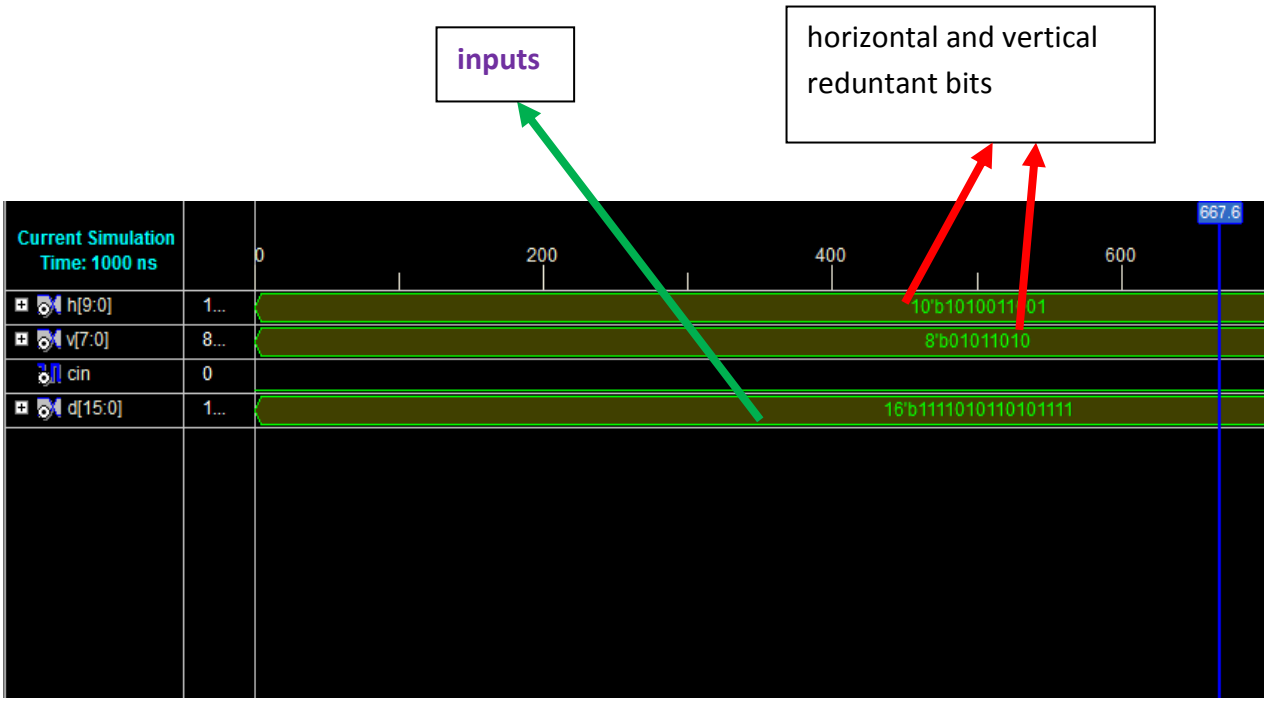


FIG 6.3 16 BIT ENCODER

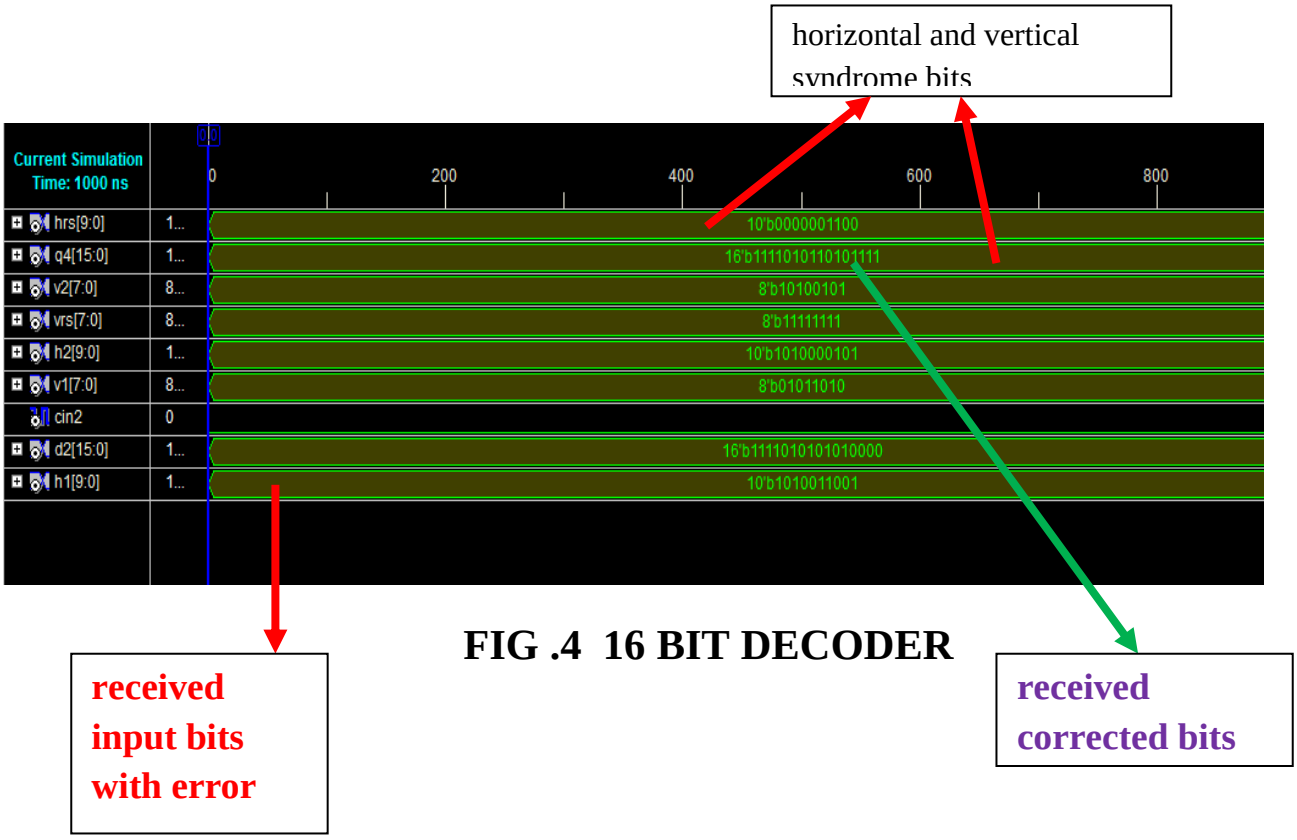


FIG .4 16 BIT DECODER

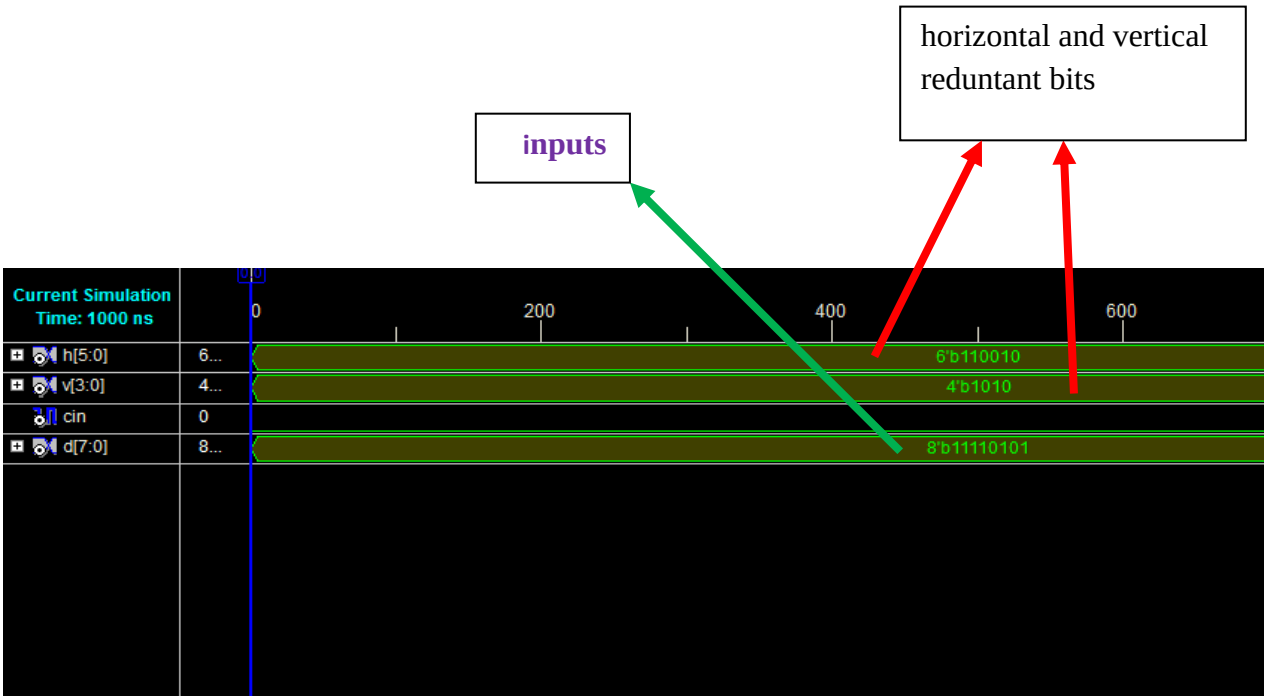


FIG 6.5 8 BIT ENCODER:

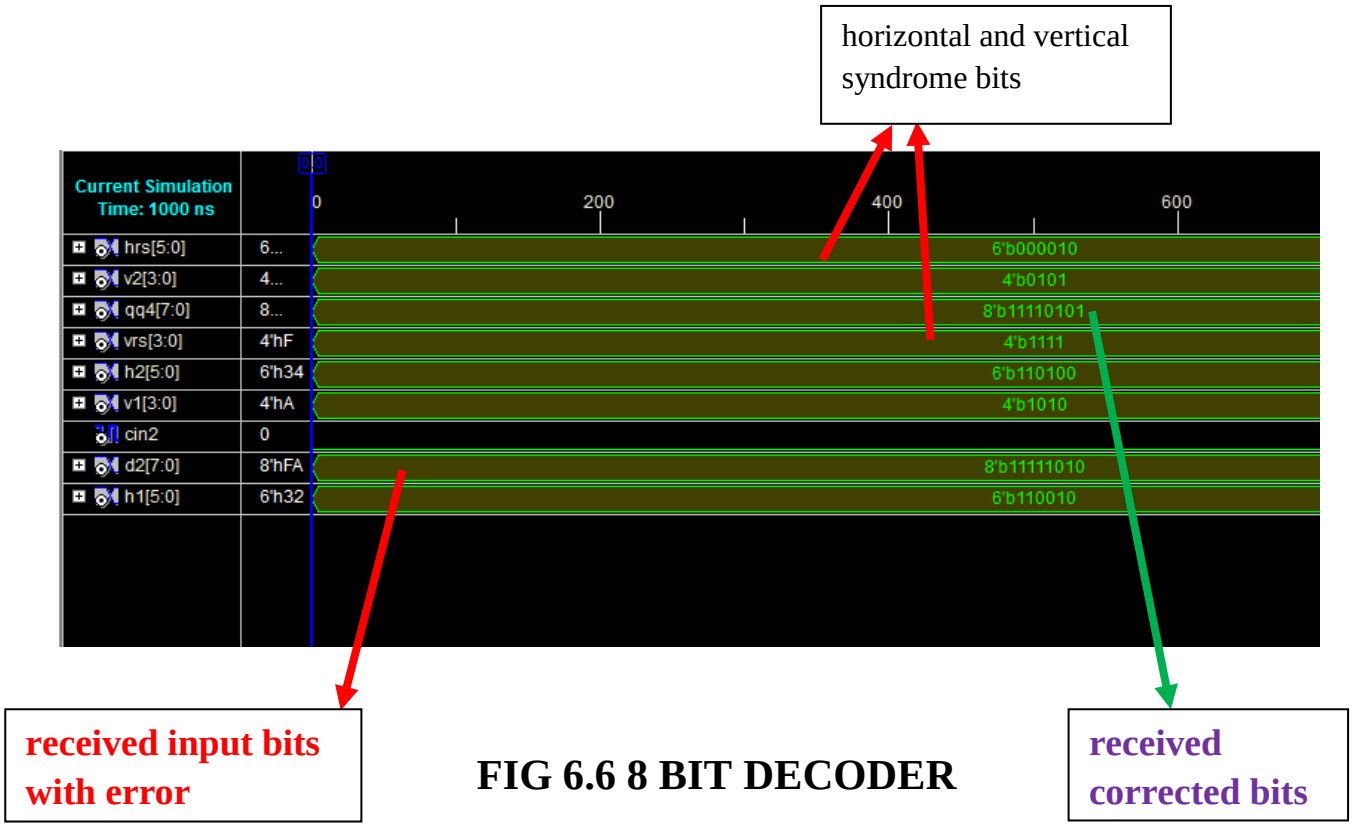


FIG 6.6 8 BIT DECODER

MATLAB IMAGE OUTPUTS:

FIG 6.7 ORIGINAL IMAGE

FIG 6.8 CORRUPTED IMAGE

FIG 6.9 IMAGE BEING ERROR DETECTED AND CORRECTED USING DMC

CHAPTER 7

CONCLUSION AND FUTURE WORK

CHAPTER -7

CONCLUSION AND FUTURE WORK

CONCLUSION

Thus, novel per-word DMC was proposed to assure the reliability of memory. The proposed protection code utilized decimal algorithm to detect errors, so that more errors were detected and corrected. The obtained results showed that the proposed scheme has a superior protection level against large MCUs in memory and it has more error detecting and correcting capability.

FUTURE WORK

In this project the binary inputs of the image was taken from an excel sheet and then given as inputs to the decimal matrix code algorithm.Therfore ,the future work will be finding the way of combining the Xilinx and the matlab and reading the binary inputs of the image directly into the code without giving the inputs everytime separately.

CHAPTER -8

REFERENCES

CHAPTER -8

REFERENCES

1. Jing Guo, Liyi Xiao, Member, IEEE, Zhigang Mao, Member, IEEE, and QiangZhao,"Enhanced memory reliability against multiple cell upsets using Decimal Matrix Code" IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst., vol. 22, no. 1, pp.127-135, Jan,2014.

2.D. Radaelli, H. Puchner, S. Wong, and S. Daniel, "Investigation of multi-bit upsets in a 150 nm technology SRAM device," *IEEE Trans.Nucl. Sci., vol. 52, no. 6, pp. 2433–2437, Dec. 2005.*

3.E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact ofscaling on neutron induced soft error in SRAMs from an 250 nm to a 22 nm design rule," *IEEE Trans. Electron Devices, vol. 57, no. 7,* pp. 1527–1538, Jul. 2010.

4. C.Argyrides and D.K.Pradhan, "Improved decoding algorithm for high reliable reed muller coding," in Proc. IEEE Int. Syst. On Chip Conf., Sep. 2007, pp. 95–98.

5. A. Sanchez-Macian, P. Reviriego, and J. A. Maestro, "Hamming SEC-DAED and extended hamming SEC-DEDTAED codes through selective shortening and bit placement," IEEE Trans. Device Mater. Rel., to be published.

6. S. Liu, P. Reviriego, and J. A. Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications," IEEETrans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 1, pp. 148–156, Jan. 2012.

7. M. Zhu, L. Y. Xiao, L. L. Song, Y. J. Zhang, and H. W. Luo, "New mix codes for multiple bit upsets mitigation in fault-secure memories," Microelectron. J., vol. 42, no. 3, pp. 553–561, Mar. 2011.

8. R. Naseer and J. Draper, “Parallel double error correcting code design to mitigate multi-bit upsets in SRAMs,” in Proc. 34th Eur. Solid-StateCircuits, Sep. 2008, pp. 222–225.

9. G. Neuberger, D. L. Kastensmidt, and R. Reis, “An automatic technique for optimizing Reed-Solomon codes to improve fault tolerance in memories,” IEEE Design Test Comput., vol. 22, no. 1, pp. 50–58, Jan.–Feb. 2005.

10. P. Reviriego, M. Flanagan, and J. A. Maestro, “A (64,45) triple error correction code for memory applications,” IEEE Trans. Device Mater. Rel., vol. 12, no. 1, pp. 101–106, Mar. 2012.

11. S. Baeg, S. Wen, and R. Wong, “Interleaving distance selection with a soft error failure model,” IEEE Trans. Nucl. Sci., vol. 56, no. 4, pp. 2111–2118, Aug. 2009. .

12. http://en.wikipedia.org/wiki/Soft_error

13.www.ieeexplore.ieee.org

14. http://cse.iitkgp.ac.in/

15.www.academia.edu

16. http://en.wikipedia.org/wiki/Verilog

CHAPTER -9

APPENDICES

CHAPTER -9

APPENDICES

9.1 XILINX ISE

9.1.1 INTRODUCTION

The Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors. The Xilinx ISE is primarily used for circuit synthesis and design, while the ModelSim logic simulator is used for system-level testing. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and Chip Scope.

9.1.2 USER INTERFACE:

The primary user interface of the ISE is the Project Navigator , which includes the design hierarchy (Sources), a source code editor (Work place) , an output console (Transcript), and a processes tree (Processes). The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure. For single-chip designs there may be one main module, with other modules included by the main module, similar to the main() subroutine in C++ programs. Design constraints are specified in modules, which include pin configuration and mapping. The Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with each function. The Transcript window provides status of currently running operations, and informs engineers on design issues. Such issues may be filtered to show Warnings,

Errors or both. Simulation System-level testing may be performed with the ModelSim logic simulator, and such test programs must also be written in HDL languages. Test bench programs may include simulated input signal waveforms, or monitors which observe and verify the outputs of the device under test.

ModelSim may be used to perform the following types of simulations:

• Logical verification, to ensure the module produces expected results

• Behavioural verification , to verify logical and timing issues

9.1.3 SYNTHESIS:

Xilinx's patented algorithms for synthesis allow designs to run upto 30% faster than competing programs, and allows greater logic density which reduces project costs. Also, due to the increasing complexity of FPGA fabric, including memory blocks as and I/O blocks, more complex synthesis algorithms were developed that separate unrelated modules into slices, reducing post-placement errors. IP Cores are offered by Xilinx and other third-party vendors, to implement system-level functions such as digital signal processing (DSP), bus interfaces, networking protocols, image processing, embedded processors, and peripherals. Xilinx has been instrumental in shifting designs from ASIC-based implementation to FPGA based implementation.

9.2 VERILOG HDL

Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits.

9.2.1 OVERVIEW

Hardware description languages such as Verilog differ from software programming languages because they include ways of describing the propagation time and signal strengths(sensitivity). There are two types of assignment operators; a blocking assignment (=), and a nonblocking(<=)assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in are latively compact and concise form. At the time of Verilog's introduction(1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical schematic capture software and specially written software programs to document and simulate electronic circuits.

The designers of Verilog wanted a language with syntax similar to the C programming language, which was already widely used in engineering software development. Like C, Verilog is case-sensitive and has a basic pre-processor (though less sophisticated than that of ANSI C/C++). Its control flow keywords (if/else, for, while, case, etc.) are equivalent and its operator precedence is compatible with C. Syntactic differences include: required bit-widths for variable declarations, demarcation of procedural blocks (Verilog uses begin end instead of curly braces, and many other minor differences. Verilog requires that variables be given a definite size. In C these sizes are assumed from the 'type' of the variable(for instance an integer type may be 8 bits).

A Verilog design consists of a hierarchy of modules. Modules encapsulated sign hierarchy, and communicate with other modules through a set of declared input, output and bidirectional ports. Internally, a module can contain any combination of the

following: (wire, register, integer, etc.), concurrent and sequential statement blocks and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin /end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a dataflow language.

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating , undefined") and signal strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

A subset of statements in the Verilog language are synthesizable. Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a net list, a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. Further manipulations to the net list ultimately lead to circuit fabrication blueprint (such as a photo mask set for an ASIC or a bit stream file for an FPGA).

9.2.2 HISTORY

Verilog was one of the first modern hardware description languages to be invented. It was created by Prabhu Goel and Phil Moorby during the winter of 1983/1984. The wording for this process was "Automated Integrated Design Systems" (later renamed to Gateway Design Automation in 1985) as a hardware modeling language. Gateway Design Automation was purchased by Cadence Design Systems in 1990. Cadence now has full proprietary rights to Gateway's Verilog and the Verilog-XL, the HDL-simulator that would become the defacto standard (of Verilog logic

simulators) for then extdecade. Originally, Verilog was intended to describe and allow simulation ; only afterwards was support for synthesis added. Verilog is a port manteau of the words “verification” and “logic”.

Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers. The other one is VHDL. HDL’s allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits.

9.3 LEXICAL TOKENS

Verilog source text files consists of the following lexical tokens:

9.3.1 WHITE SPACE

White spaces separate words and can contain spaces, tabs, new-lines and form feeds. Thus a statement can extend over multiple lines without special continuation characters.

9.3.2 COMMENTS

Comments can be specified in two ways (exactly the same way as in C/C++): - Begin the comment with double slashes (//). All text between these characters and the end of the line will be ignored by the Verilog compiler. - Enclose comments between the characters /* and */. Using this method allows you to continue comments on more than one line. This is good for “commenting out” many lines code, or for very brief in-line comments.

9.3.3 NUMBERS

Number storage is defined as a number of bits, but values can be specified in binary, octal, decimal or hexadecimal (Examples are 3'b001, a 3-bit number, 5'd30, (=5'b11110), and 16'h5ED4, (=16'd24276)

9.3.4 IDENTIFIERS

Identifiers are user-defined words for variables, function names, module names, block names and instance names. Identifiers begin with a letter or underscore (Not with a number or $) and can include any number of letters, digits and underscores. Identifiers in Verilog are case-sensitive.

Syntax allowed symbols

ABCDE . . . abcdef. . . 1234567890

_$ not allowed: anything else especially - &#@

9.3.5 OPERATORS

Operators are one, two and sometimes three characters used to perform operations on variables. Examples include >, +, ~, &, .

9.3.6 VERILOG KEYWORDS

These are words that have special meaning in Verilog. Some examples are assign, case, while, wire, register, and, or, nand, and module. They should not be used as identifiers.

9.3.7 CREATING VERILOG TEST FIXTURE

Open the Verilog test fixture in the HDL editor by double-clicking it in the Sources window. If you examine the contents of the new source file you will see that, like a standard VHDL source file, the Xilinx tools automatically generate lines of code in the file to get you started with circuit input definition. This generated code includes:

- a Comment block template for documentation
- a Module statement
- a UUT instantiation
- input initialization

Scroll down to the end of the test fixture file to see the "initial begin" and "end" statements of the module.

The simplest way of defining input stimulus in a Verilog test fixture is to use timing controls and delay, denoted by the pound symbol (#). For example, the statement #100 present in example1_test_verilog.v tells the simulator to delay for 100 ns. Therefore, any statement made after this time scale statement will occur after the 100 ns delay time. It's important to note that the time scale for the delay is defined by the time scale statement at the beginning of the file. By default, the Xilinx tools define the time scale as 1ns/1ps, which indicates that the units are in nano seconds while calculated time precision is 1 pico seconds.

9.4 MODELLING IN VERILOG

Verilog has four levels of modelling:

1) The switch level which includes MOS transistors modelled as switches.

2) The gate level.

3) The Data-Flow level.

4) The Behavioral or procedural level.

In this project we have used Gate-Level Modelling and Behavioral Modelling to write the code and the description of this two level of modelling are explained below.

9.4.1 GATE-LEVEL MODELLING:

Primitive logic gates are part of the Verilog language. Two properties can be specified, drive_strength and delay. Drive_strength specifies the strength at the gate outputs. The strongest output is a direct connection to a source, next comes a connection through a conducting transistor, then a resistive pull-up/down. The drive strength is usually not specified, in which case the strengths defaults to strong1 and strong0. Delays: If no delay is specified, then the gate has no propagation delay; if two delays are specified, the first represent the rise delay, the second the fall delay; if only one delay is specified, then rise and fall are equal. Delays are ignored in synthesis. This method of specifying delay is a special case of “Parameterized Modules” . The parameters for the primitive gates have been predefined as delays.

9.4.1 (a) BASIC GATES

These implement the basic logic gates. They have one output and one or more inputs. In the gate instantiation syntax, GATE stands for one of the keywords and, nand, or, nor, xor, xnor.

9.4.1(b) Buf, NOT GATES

These implement buffers and inverters, respectively. They have one input and one or more outputs. In the gate instantiation syntax, GATE stands for either the keyword buf or not.

9.4.1(c) THREE-STATE GATES:

bufif1, bufif0, notif1, notif0 These implement 3-state buffers and inverters. They propagate z (3-state or high-impedance) if their control signal is de-asserted. These can have three delay specifications: a rise time, a fall time, and a time to go into 3-state.

9.4.1(d) DATA TYPES:

Value Set Verilog consists of only four basic values. Almost all Verilog data types store all these values: 0 (logic zero, or false condition) 1 (logic one, or true condition) x (unknown logic value) x and z have limited use for synthesis. z (high impedance state).

9.4.1(e) WIRE :

A wire represents a physical wire in a circuit and is used to connect gates or modules. The value of a wire can be read, but not assigned to, in a function or block. A wire does not store its value but must be driven by a continuous assignment statement or by connecting it to the output of a gate or module. Other specific types of wires include: 1.wand (wired-AND): the value of a wand depend on logical AND of all the drivers connected to it.2. wor (wired-OR): the value of a wor depend on logical OR of all the drivers connected to it.3. tri (three-state;): all drivers connected to a tri must be z, except one (which determines the value of the tri).

Syntax

wire [msb:lsb]

wire_variable_list;

wand [msb:lsb]

wand_variable_list;

wor [msb:lsb]

wor_variable_list;

tri [msb:lsb] tri_variable_list;

9.4.1(f) REGISTER:

Declare type register for all data objects on the left hand side of expressions in inital and always procedures, or functions. A register is the data type that must be used for latches, flip-flops and memories. However it often synthesizes into leads rather than storage. In multi-bit registers, data is stored as unsigned numbers and no sign extension is done for what the user might have thought were two's complement numbers.

Syntax

register [msb:lsb] register_variable_list;

9.4.1(g) INPUT, OUTPUT, INOUT :

These keywords declare input, output and bidirectional ports of a module or task. Input and inout ports are of type wire. An output port can be configured to be of type wire, register, wand, wor or tri. The default is wire.

Syntax

input [msb:lsb]

input_port_list;

output [msb:lsb]

output_port_list;

inout [msb:lsb]

inout_port_list;

9.4.1(h) INTEGER :

Integers are general-purpose variables. For synthesis they are used mainly loops-indices, parameters, and constants. They are of implicitly of type register. However they store data as signed numbers whereas explicitly declared register types store them as unsigned. If they hold numbers which are not defined at compile time, their size will default to 32-bits. If they hold constants, the synthesizer adjusts them to the minimum width needed at compilation.

Syntax:

integer integer_variable_list; ... integer_constant ... ;

9.4.2 BEHAVIORAL MODELLING:

Verilog procedural statements are used to model a design at a higher level of abstraction than the other levels. They provide powerful ways of doing complex designs. However small changes in coding methods can cause large changes in the hardware generated. Procedural statements can only be used in procedures.

9.4.2(a) PROCEDURAL ASSIGNMENTS:

Procedural assignments are assignment statements used within Verilog procedures (always and initial blocks). Only reg variables and integers (and their bit/part-selects and concatenations) can be placed left of the "=" in procedures. The right hand side of the assignment is an expression which may use any of the operator types.

Delay in Assignment (not for synthesis):

In a delayed assignment, Δt time units pass before the statement is executed and the left-hand assignment is made. With intra-assignment delay, the right side is evaluated immediately but there is a delay of Δt before the result is place in the left hand assignment. If another procedure changes a right-hand side signal during Δt, it does not effect the output. Delays are not supported by synthesis tools.

Syntax for Procedural Assignment

variable = expression

Delayed assignment

#Δt variable = expression;

Intra-assignment

delay variable = #Δt expression;

9.4.2(b) BLOCKING ASSIGNMENTS:

Procedural (blocking) assignments (=) are done sequentially in the order the statements are written. A second assignment is not started until the preceding one is complete.

Syntax

Blocking variable = expression;

variable = #Δt expression;

grab inputs now, deliver ans. later.

#Δt variable = expression;

grab inputs later, deliver ans. Later

9.4.2(c) NON-BLOCKING (RTL) ASSIGNMENTS :

RTL (non-blocking) assignments (<=), which follow each other in the code, are started in parallel. The right hand side of non-blocking assignments is evaluated starting from the completion of the last blocking assignment or if none, the start of the procedure. The transfer to the left hand side is made according to the delays. An intra-assignment delay in a non-blocking statement will not delay the start of any subsequent statement blocking or non-blocking. However normal delays are cummulative and will delay the output.

For synthesis:

• One must not mix "<=" or "=" in the same procedure.

• “<=” best mimics what physical flip-flops do; use it for “always @ (posedge clk ..) type procedures.

• “=” best corresponds to what c/c++ code would do; use it for combinational procedures.

Syntax

Non-Blocking

variable <= expression;

variable <= #Δt expression;

#Δt variable <= expression;

Blocking

variable = expression;

variable = #Δt expression;

#Δt variable = expression;

begin ... end:

begin ... end block statements are used to group several statements for use where one statement is syntactically allowed. Such places include functions, always and initial blocks, if, case and for statements. Blocks can optionally be named.

Syntax

begin block_name

reg [msb:lsb] reg_variable_list;

integer [msb:lsb] integer_list;

parameter [msb:lsb] parameter_list; ... statements ...

end

9.4.2(d) FOR LOOPS:

Similar to for loops in C/C++, they are used to repeatedly execute a statement or block of statements. If the loop contains only one statement, the begin ... end statements may be omitted.

Syntax

for (count = value1; count </<=/>/>= value2; count = count +/- step)

begin

... statements ...

End

9.4.2(e) WHILE LOOPS:

The while loop repeatedly executes a statement or block of statements until the expression in the while statement evaluates to false. To avoid combinational feedback during synthesis, a while loop must be broken with an @(posedge/negedge clock) statement . For simulation a delay inside the loop will suffice. If the loop contains only one statement, the begin ... end statements may be omitted.

Syntax

while (expression)

begin

... statements ...

End

9.4.2(f) if ... else if ... else:

The if ... else if ... else statements execute a statement or block of statements depending on the result of the expression following the if. If the conditional expressions in all the if's evaluate to false, then the statements in the else block, if present, are executed. There can be as many else if statements as required, but only one if block and one else block. If there is one statement in a block, then the begin .. end statements may be omitted. Both the else if and else statements are optional. However if all possibilities are not specifically covered, synthesis will generated extra latches.

Syntax

if (expression)
begin
... statements ...
End
else if (expression)
begin
... statements ...
end
... more else if blocks ...
else
begin
... statements ...
end

9.4.2(g) CASE:

The case statement allows a multipath branch based on comparing the expression with a list of case choices. Statements in the default block executes when none of the case choice comparisons are true. With no default, if no comparisons are true, synthesizers will generate unwanted latches. Good practice says to make a habit of

putting in a default whether you need it or not. If the defaults are don't cares, define them as 'x' and the logic minimizer will treat them as don't cares and save area. Case choices may be a simple constant, expression, or a comma-separated list of same.

Syntax

case (expression)
case_choice1:
begin
... statements ...
end
case_choice2:
begin
... statements ...
End
... more case choices blocks ...
default:
begin
... statements ...
End
Endcase

9.5 MATLAB:

MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, you can analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions enable you to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java.

You can use MATLAB for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called tool boxes. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

9.5.1 HISTORY:

Cleve Moler, the chairman of the computer science department at the University of New Mexico, started developing MATLAB in the late 1970s. He designed it to give his students access to LINPACK and EISPACK without them having to learn Fortran. It soon spread to other universities and found a strong audience within the applied

mathematics community. Jack Little, an engineer, was exposed to it during a visit Moler made to Stanford University in 1983. Recognizing its commercial potential, he joined with Moler and Steve Bangert. They rewrote MATLAB in C and founded MathWorks in 1984 to continue its development. These rewritten libraries were known as JACKPAC. In 2000, MATLAB was rewritten to use a newer set of libraries for matrix manipulation, LAPACK.

MATLAB was first adopted by researchers and practitioners in control engineering, Little's specialty, but quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra, numerical analysis, and is popular amongst scientists involved in image processing.

9.5.2 KEY FEATURES OF MATLAB:

• High-level language for numerical computation, visualization, and application development.

• Interactive environment for iterative exploration, design, and problem solving.

• Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration, and solving ordinary differential equations.

• Built-in graphics for visualizing data and tools for creating custom plots.

• Development tools for improving code quality and maintainability and maximizing performance.

• Tools for building applications with custom graphical interfaces.

• Functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET(Dot NET), and Microsoft Excel.

9.5.3 THE MATLAB SYSTEM

The MATLAB system consists of five main parts:

9.5.4 THE MATLAB LANGUAGE

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

9.5.5 THE MATLAB WORKING ENVIRONMENT

This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

9.5.6 HANDLE GRAPHICS

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces on your MATLAB applications.

9.5.7 THE MATLAB MATHEMATICAL FUNCTION LIBRARY

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigenvalues, Bessel functions, and fast Fourier transforms.

9.5.8 THE MATLAB APPLICATION PROGRAM INTERFACE (API)

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

9.5.9 DESCRIPTION OF THE COMMANDS USED FOR THE APPLICATION:

9.5.9(a) xlsread

Read Microsoft Excel spreadsheet file

Syntax

[num,txt,raw] = xlsread(filename)

Description

[num,txt,raw] = xlsread(filename) reads data from the first worksheet in the Microsoft Excel spreadsheet file named filename and returns the numeric data in array num. Optionally, returns the text fields in cell array text, and the unprocessed data (numbers and text) in cell array raw. If your system does not have Excel for Windows, xlsread operates in basic import mode, and reads only XLS or XLSX files.

9.5.9(b) Imshow

Display image

Syntax

imshow(X,map)

Description

imshow(X,map) displays the indexed image X with the colormap map. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0.

9.5.9(c) dec2bin

Convert decimal to binary number in string

Syntax

str = dec2bin(d,n)

Description

returns the str = dec2bin(d,n) produces a binary representation with at least n bits. The output of dec2bin is independent of the endian settings of the computer you are using.

9.5.9(d) bin2dec

Convert binary number string to decimal number

Syntax

bin2dec(binarystr)

Description

bin2dec(binarystr) interprets the binary string binarystr and returns the equivalent decimal number.

bin2dec ignores any space (' ') characters in the input string

9.5.9(e) num2str

Convert number to string

Syntax

str = num2str(A)

Description

str = num2str(A) converts array A into a string representation str. When A contains floating-point numbers, the output format depends upon the size of the original values and sometimes includes exponents.

- num2str is useful for labeling and titling plots with numeric values.
- Unlike fprintf, the num2str function trims any leading spaces from a string, even when used with the space character flag.