IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

VLSI Implementation Of Error Detection And Correction Codes For Space Engineering

D. Mani Ayyappa[1], J. Vishala Devi[2], M. Durga Prasad[3],Y. Abhi Ram[4]

Under the Guidance of Ms.V Jahnavi,Assistant Professor,
Electronics and Communication Engineering,
Sasi Institute of Technology And Engineering , Tadepalligudem , 534101

Abstract

The paper discusses a novel error detection and correction scheme designed for space engineering, aiming to improve on-chip memory reliability under environmental conditions such as cosmic radiation and extreme temperatures. The proposed 2-dimensional code uses a divide-symbol approach, leveraging XOR operations to encode diagonal, parity, and check bits during the encoding process. The decoding process is intricate, involving syndrome calculation and region selection, which efficiently identifies and corrects errors. The effectiveness of the scheme is demonstrated through simulations using Xilinx Vivado, revealing low power consumption and minimal area utilization in comparison to existing methods. This innovative error correction methodology presents a promising solution for enhancing memory reliability in space applications, specifically addressing issues caused by multiple cell upsets induced by radiation. The proposed scheme's benefits include improved memory reliability, reduced power consumption, and minimal area utilization, making it a potentially valuable tool for future space engineering endeavours.

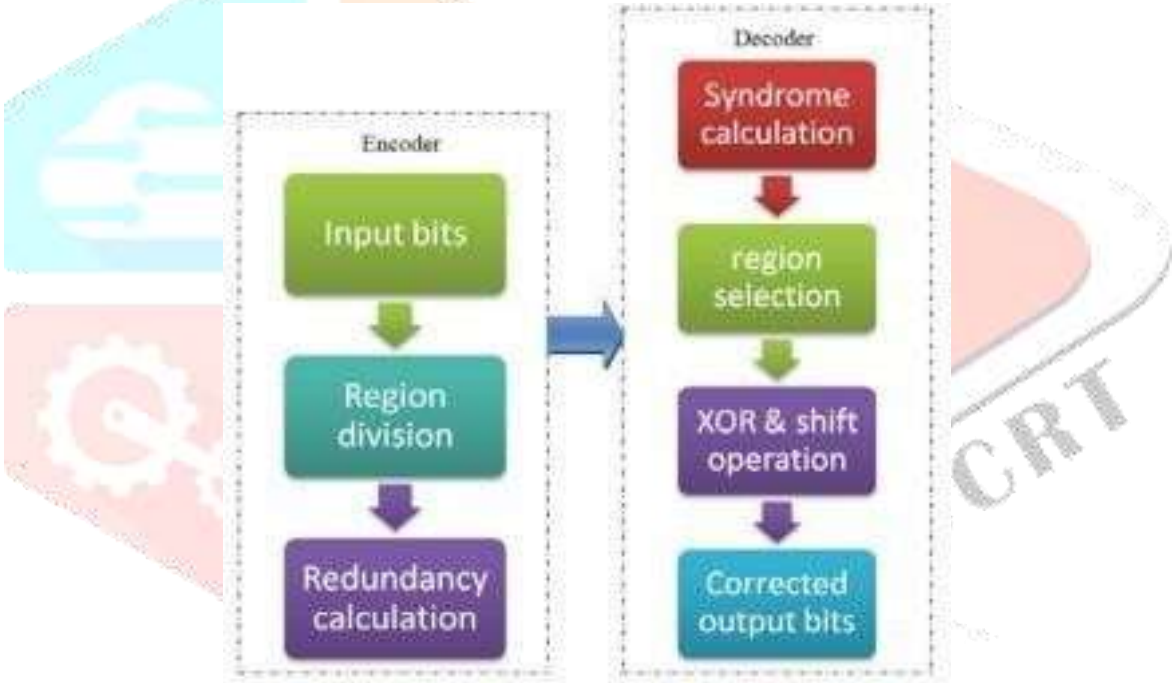
Keywords: Error detection, correction, XOR operations, space engineering.

CHAPTER-I

INTRODUCTION

The paper "VLSI Implementation of Error Detection and Correction Codes for Space Engineering" by R. Jenila, C. Supraja, Dr. C. Kanmani Pappa, and N. Dharani explores the necessity of enhancing on-chip memory reliability in space applications. As technology progresses, on-chip memories in a die are susceptible to bit errors due to single events or multiple cell upsets. These upsets are often triggered by environmental factors such as cosmic radiation, alpha, neutron particles, or extreme temperatures in space, which can lead to data corruption.

Traditional error detection and correction techniques (ECC) are typically employed to recognize and rectify corrupted data over the communication channel. However, these techniques may not always be effective in dealing with multiple cell upsets (MCUs). MCUs occur when two or more bits of the same memory get affected, posing a significant challenge for ECC techniques.



ECC Methodology.

In response to this, the authors propose an advanced error correction 2-dimensional code based on divide-symbol to mitigate radiation-induced MCUs in memory for space applications. This 2- dimensional code employs XOR operations to analyze diagonal bits, parity bits, and check bits during the encoding process. To recover the data, the XOR operation is performed again between the encoded bits and the recalculated encoded bits.

The proposed scheme was simulated and synthesized using Xilinx Vivado implemented in Verilog HDL. The results show that this encoding-decoding process consumes low power and occupies minimum area

and delay, making it a more efficient solution compared to the well-known existing methods.

The authors also discuss various existing techniques for error detection and correction, including the use of parity checks, interleaving techniques, and error correction codes (ECCs). They highlight the limitations of these techniques, such as increased system design complexity, area and power consumption, and the requirement for more parity bits, more time for decoding, and more complex circuits for encoding and decoding operations.

The paper introduces a new encoding-decoding algorithm for error correction and detection in multiple cell upsets (MCUs). The encoding process involves reading the input 16-bit data, dividing the input data into four groups, analyzing diagonal bits, parity bits, and check bits using XOR operation, and calculating the syndrome values for diagonal, parity, and check bits. The decoding process involves syndrome calculation, verification, region selection, and correction.

In conclusion, the proposed scheme can detect and correct multi-bit errors and consumes low power compared to other existing methods. The power consumed for encoding and decoding process is 0.167W and 0.127W respectively. Although the decoder area increases compared to other existing methods, this is mitigated by using advanced region selection criteria.

CHAPTER-II

LITERATURE REVIEW

R. C. Baumann (2005), "Soft errors in advanced computer systems," IEEE Des. Test. Comput., vol. 22, no. 3, pp. 258-266.

The paper "Soft errors in advanced computer systems" by Robert C. Baumann provides a comprehensive analysis of soft-error sensitivity in modern systems. Soft errors, which are transient changes in the state of a digital circuit due to environmental factors, are a significant concern in the design and operation of advanced computer systems. This paper demonstrates that soft-error sensitivity is largely dependent on the application in which the system is used. The paper discusses various mechanisms through which soft errors can occur, including radiation-induced single event upsets, which are a major source of soft errors in silicon. It also presents a model that computes the probability that a strike at the output of a gate has an impact in any output by traversing the circuits backwards from the outputs and gaining information about the logical masking using signal probabilities. The paper also presents a novel hardware architecture that reduces the cost of rollback in any kinds of circuit and detects the error in logic functions already exist, leading to a lot of hardware overhead in non-processor design. Moreover, it introduces a fault-tolerant register latch organization that is able to detect single-bit errors while it is clock gated and can be efficiently reused for offline and general online testing. Overall, the paper emphasizes the importance of understanding and managing soft errors in advanced computer systems. It suggests that soft-error sensitivity is application-

dependent and provides several strategies for managing soft errors, including the use of specialized hardware architectures and the development of fault-tolerant latch organizations.

Summary: The paper introduces a novel hardware architecture for rollback reduction and a fault- tolerant register latch organization for error detection. It highlights the importance of understanding and managing soft errors, suggesting that sensitivity varies by application and proposing strategies such as specialized hardware architectures and fault-tolerant latch organizations.

C. L. Chen and M. Y. Hsiao (1984), "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," IBM J. Res. Develop., vol.28, no. 2, pp. 124-134.

The paper "Error-correcting codes for semiconductor memory applications: A state-of-the-art review" by C. L. Chen and M. Y. Hsiao, published in IBM Journal of Research and Development in 1984, provides an extensive review of error-correcting codes (ECCs) used in semiconductor memory applications. ECCs are critical in ensuring the integrity and reliability of data in memory devices, which are increasingly being used in various computing applications. The authors discuss the challenges and solutions associated with implementing ECCs in semiconductor memory applications, including issues related to error detection and correction, power consumption, and area constraints. The paper provides a comprehensive overview of the state-of-the-art in ECC design and implementation, covering various types of ECCs and their applications in semiconductor memory. The authors discuss the advantages and disadvantages of different ECC designs, highlighting the trade-offs between error detection capability, power consumption, and implementation complexity. The authors also explore the future directions for ECC research and development, suggesting potential areas for improvement and innovation in ECC design and implementation. This includes the development of more efficient error detection and correction algorithms, as well as the exploration of new materials and fabrication techniques that could enable more reliable and energy-efficient memory devices. The paper is highly influential in the field of semiconductor memory and ECC design, and it has been cited by numerous subsequent papers and articles. The insights and recommendations provided in the paper continue to guide research and development efforts in this area.

Summary: The paper suggests potential areas for future research and development in ECC design and implementation, including the development of more efficient error detection and correction algorithms and the exploration of new materials and fabrication techniques.

E. Ibe, S. Chung, S. Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto and T.Akioka (2006), "Spreading diversity in multi-cell neutron-induced upsets with device scaling," in Proc. IEEE Custom Integrated Circuit Conf., pp. 437-444

The paper "Spreading diversity in multi-cell neutron-induced upsets with device scaling" by E. Ibe, S. Chung, S. Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto, and T. Akioka, published in the proceedings of the IEEE Custom Integrated Circuit Conference, addresses the issue of multi-cell neutron-induced upsets in advanced computer systems. The study investigates the spreading of diversity in multi-cell upsets with device scaling. Multi-cell upsets are a type of soft error that occurs when two or more bits in the same memory location are affected simultaneously due to external factors such as cosmic rays or radiation. These errors can cause significant problems in digital systems, especially those used in space applications where the environment is harsh and unpredictable. The researchers conducted experiments to observe the spreading of diversity in multi-cell upsets across different device scales. They found that as the device scale increased, the spreading of diversity also increased. This suggests that larger devices are more vulnerable to multi-cell upsets and that measures to mitigate these effects should be taken into account when designing and manufacturing devices. The paper also discusses potential solutions to mitigate the impact of multi-cell upsets. These include techniques such as error detection and correction codes, as well as physical shielding to protect sensitive components from radiation. The authors conclude that further research is needed to develop more effective strategies for dealing with multi-cell upsets in advanced computer systems.

Summary: It finds that larger devices are more vulnerable to these upsets, suggesting that design and manufacturing considerations must take this into account. Potential solutions include error detection and correction codes and physical shielding against radiation. Further research is needed to develop more effective strategies for dealing with multi-cell upsets.

P. Reviriego, J.A. Maestro and C. Cervantes (2007), "Reliability analysis of memories suffering multiple bit upsets," IEEE Trans. Device Mater. Rel., vol. 7, no. 4, pp. 592-601.

The paper "Reliability analysis of memories suffering multiple bit upsets" by P. Reviriego, J.A. Maestro, and C. Cervantes published in IEEE Transactions on Device Materials & Reliability in 2007, addresses the issue of multiple bit upsets in memory systems. The paper explores the problem of multiple bit upsets (MBUs), which occur when two or more bits of the same memory get affected. This is a significant challenge for traditional error detection and correction techniques, as these techniques are typically designed to handle single bit errors. The paper conducts a reliability analysis of memories suffering from MBUs, providing insights into the occurrence and implications of these errors. The authors examine the factors contributing to MBUs, such as environmental conditions like cosmic radiation, alpha, neutron particles, or extreme temperatures. They also investigate the effects of MBUs on memory reliability, demonstrating how these errors can lead to data corruption and compromise the integrity of data storage and retrieval processes. The paper underscores the need for advanced error correction techniques capable of handling MBUs effectively. It contributes to the body of knowledge on memory reliability, offering valuable insights into the challenges posed by MBUs and potential solutions for enhancing memory reliability in the face of such errors.

Summary: This paper identifies environmental factors like cosmic radiation, alpha, neutron particles, or extreme temperatures as contributors to MBUs. It also demonstrates how these errors can lead to data corruption and compromise data integrity. The paper emphasizes the need for advanced error correction techniques capable of handling MBUs effectively.

Z. Ming, X. Li Yi and L. Hong Wei (2011), "New SEC-DED-DAEC codes for multiple bit upsets mitigation in memory," IEEE/IFIP 19th international conference on VLSI and system-on-chip., pp. 254-259.

The paper "New SEC-DED-DAEC codes for multiple bit upsets mitigation in memory" by Z. Ming, X. Li Yi, and L. Hong Wei, presented at the IEEE/IFIP 19th international conference on VLSI and system-on-chip, discusses a new error correction code for the reduction of radiation- induced multiple bit upsets in memories. The paper proposes a new error correction code that detects and corrects adjacent double bit errors, thereby lowering the errors for non-adjacent double bit errors. The experimental results demonstrate that it reduces 40% hardware redundancy and is more efficient compared to other existing ECC codes. Additionally, this method minimizes the errors for non-adjacent DBE by 12% when compared with conventional SEC-DED-DAEC codes, leading to a high reliability memory system design. The paper also discusses the use of different sets of codes (cyclic-linear block codes) as ECC to protect memory from data loss. This scheme exploits the localization of MCU errors, along with the features of DS codes to enhance error correction possibilities and to reduce the decoding time. This is implemented in HDL and the simulation result indicates that this technique is effective in reducing the decoding time and also the area and power consumption. The authors suggest a new code to correct triple adjacent errors (SEC-DAEC-TAEC) and 3-bit burst errors for different data word lengths (16, 32, and 64 data bits). Two optimization criteria have been used; reducing the total number of ones in the parity check matrix minimizes decoding time and the maximum number of ones in its rows optimizes the speed. The paper concludes that the proposed scheme can detect and correct multi-bit errors and consumes low power compared to other existing methods. The power consumed for encoding and decoding process is 0.167W and 0.127W respectively. The decoder area increases compared to other existing methods, but this is reduced by using advanced region selection criteria.

Summary: The proposed code is shown to reduce 40% hardware redundancy and minimize non- adjacent double bit errors by 12%. The paper also discusses the use of cyclic-linear block codes for enhanced error correction and reduced decoding time. The proposed code can detect and correct multi-bit errors with low power consumption.

J. Guo, L. Xiao, Z. Mao and Q. Zhao (2013), "Enhanced memory reliability against multiple cell upsets using decimal matrix code," IEEE Trans. On very large scale integration (VLSI) systems. pp.

1-4.

The paper "Enhanced Memory Reliability Against Multiple Cell Upsets Using Decimal Matrix Code" by J. Guo, L. Xiao, Z. Mao, and Q. Zhao, published in the IEEE Transactions on Very Large Scale Integration (VLSI) Systems, proposes a novel decimal matrix code (DMC) based on a divide-symbol based on decimal algorithm to achieve maximum error detection capability. The paper also introduces an encoder-reuse technique (ERT) to minimize the area overhead of extra circuits without disrupting the entire encoding and decoding processes. The authors argue that the DMC and ERT combination is highly influential in enhancing memory reliability, especially in environments exposed to radiation, where transient multiple cell upsets (MCUs) become major issues affecting the reliability of memories. The DMC minimizes area and delay overheads compared to existing codes such as Hamming, matrix codes, and built-in current sensors. It also improves memory reliability by enhancing the error correction capability. The paper has been cited 89 times according to Semantic Scholar, indicating its influence in the field. It has also been referenced in several other papers, suggesting its relevance and applicability in addressing the problem of multiple cell upsets in memory protection.

Summary: This proposes a novel decimal matrix code (DMC) and an encoder-reuse technique (ERT) to enhance memory reliability, particularly in environments exposed to radiation. The DMC maximizes error detection capability and minimizes area and delay overheads, making it a valuable tool for addressing multiple cell upsets in memory protection.

CHAPTER-III

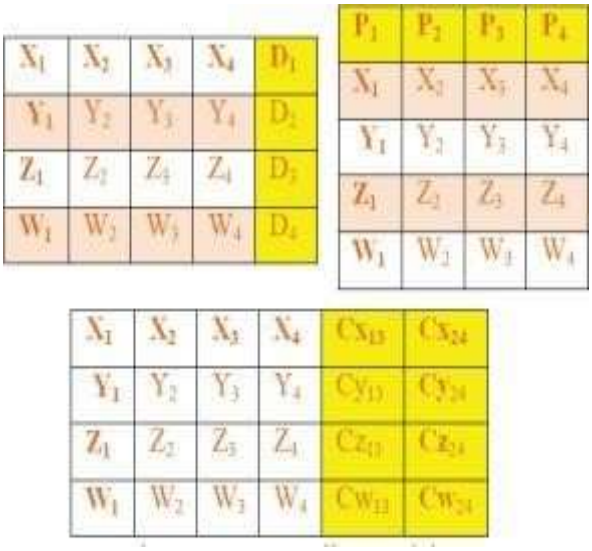
PROPOSED METHOD

In the context of the paper, the proposed encoder and decoder are part of an error correction and detection scheme. The encoder is responsible for transforming the input data into a coded format, while the decoder performs the reverse operation, converting the coded data back into its original form.

Here's how the encoder and decoder work in detail:

Encoding Methodology:

The encoding process, a pivotal component in ensuring data integrity during transmission, involves several intricate steps. Let's delve deeper into the nuances of how the encoder transforms a 16-bit input into a coded format, infusing it with enhanced error detection and correction capabilities.



Encoding model

Data Grouping and Matrix Formation:

The initial phase entails breaking down the 16-bit input into four distinct groups—Xi, Yi, Zi, and Wi. These groups form the basis of subsequent operations. The encoder then arranges these groups into a matrix, setting the stage for further computations.

X1	**Y1**	**Z1**	**W1**
X2	Y2	Z2	W2
X3	Y3	Z3	W3
X4	Y4	Z4	W4

Data Grouping and Matrix Formation.

XOR Operations Unveiling Diagonal, Parity, and Check Bits:

At the core of the encoding process lies a series of XOR operations on these grouped sets.

Diagonal Bits (Di): By XORing bits from Xi and Zi, the encoder introduces diagonal redundancy, crucial for detecting errors affecting bits along the diagonals of the data matrix.

$$D_1 = X_1 \oplus Y_2 \oplus Z_3 \oplus W_4$$
$$D_2 = X_2 \oplus Y_3 \oplus Z_4 \oplus W_1$$

Parity Bits (Pi): Comprehensive XOR operations involving bits from Xi, Yi, Zi, and Wi result in Parity bits. This thorough parity checking enhances the encoder's error detection capabilities.

$$P_1 = X_1 \oplus Y_1 \oplus Z_1 \oplus W_1$$
$$P_2 = X_2 \oplus Y_2 \oplus Z_2 \oplus W_2$$

Check Bits (Ci): Specific XOR operations involving input bits generate Check bits, contributing to error identification and correction during decoding.

$$Cx_{11} = X_1 \oplus X_3$$
$$Cx_{24} = X_2 \oplus X_4$$
$$Cy_{11} = Y_1 \oplus Y_3$$
$$Cy_{24} = Y_2 \oplus Y_4$$

Resulting 32-Bit Coded Data:

The outcome of this encoding journey is a 32-bit coded data sequence. This sequence encapsulates not only the original 16 bits but also the calculated Diagonal, Parity, and Check bits. This extended format fortifies the data against errors, providing a robust foundation for subsequent decoding.

Error Resilience and Redundancy:

The inclusion of Diagonal, Parity, and Check bits significantly boosts the resilience of the coded data. This redundancy, while expanding the data size, ensures the decoder's ability to detect and rectify errors during the decoding process.

Decoding Methodology:

As data reaches its destination, the decoding process becomes paramount in reconstructing the original 16-bit data from the coded format. Let's unravel the intricacies involved in this intricate decoding operation.

Syndrome Calculation:

The decoder kicks off the process by calculating syndrome values—obtained through XOR operations between stored redundancy data and recalculated redundancy bits based on the received message. These syndromes serve as indicators of potential errors within the received coded data.

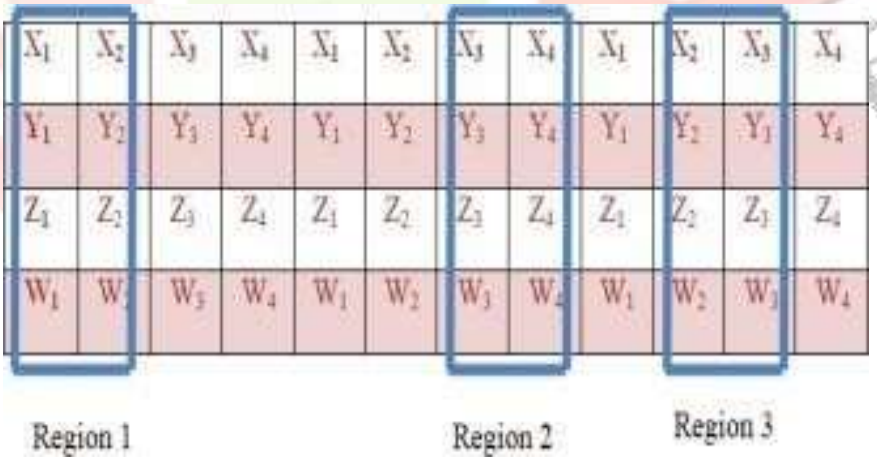
$$SD_i = D_i \oplus RD_i$$
$$SP_i = P_i \oplus RP_i$$
$$SC_i = C_i \oplus RC_i$$

Error Identification and Analysis:

Through a meticulous analysis of calculated syndrome values, the decoder establishes conditions for identifying the existence and location of errors. These values serve as fingerprints of discrepancies, guiding the decoder in subsequent correction steps.

Region Selection and Error Correction:

Armed with information about errors, the decoder strategically selects regions within the coded message that are affected. Applying correction mechanisms specific to these regions, the decoder aims to rectify errors and restore the integrity of the data.



Different regions of date bits.

Original Data Recovery:

The ultimate objective of the decoding process is to recover the original 16-bit data. By eliminating the redundancy bits added during encoding, the decoder meticulously reconstructs the initial data, ensuring its fidelity to the information before encoding.

Combinational Logic Circuits and Boolean algebra: Both the encoder and decoder operate as combinational

logic circuits, implying that their actions are solely determined by current input states, without reliance on previous states or memory elements. This inherent characteristic makes combinational circuits ideal for tasks requiring immediate responses to input changes, a crucial aspect in error correction and detection. The foundational principles of Boolean algebra underpin the operations of both the encoder and decoder. Boolean algebra deals with binary variables and logical operations, such as AND, OR, and XOR. In the context of error correction coding, XOR operations play a pivotal role in manipulating and analyzing input data. The encoder and decoder leverage these principles to ensure the accurate transformation of data during encoding and decoding phases.

CHAPTER IV ADVANTAGES & APPLICATIONS

Advantages:

1. Space Applications

Space applications, as mentioned earlier, cover a wide array of technologies and services benefiting life on Earth. Some of the most relevant applications include communication, navigation (e.g., GPS), Earth observation, weather forecasting, scientific research, and more. Space missions require a combination of robust systems, efficient error correction, and power efficiency due to the challenging environment of space.

2. Effective Error Correction in Space Communications

Error correction is crucial in space communications because of the long distances and harsh conditions of space. Space-based communications often suffer from noise, interference, and data degradation due to the vast distances that signals travel and the harsh environmental conditions. The following methods are commonly used in space systems:

3. Low Power Consumption in Space Systems

Low power consumption is critical in space systems, especially for satellites, rovers, and other space missions, because power resources (like solar panels or batteries) are limited. Efficient use of energy ensures that the systems can run longer and perform their tasks effectively without relying heavily on power-intensive components.

Solar Power: Many space applications rely on solar panels to generate power.

4. Advanced Error Correction Codes

Advanced error correction techniques are essential in ensuring reliable communication in space. Some of the most advanced techniques include (Low-Density Parity-Check Codes).

Turbo Codes LDPC

Codes Polar Codes

Applications:

1. Space Engineering

Space engineering involves the design, development, and operation of spacecraft, satellites, and other systems for space exploration, communication, Earth observation, and more. Key applications include

2. Digital Electronics

Digital electronics refers to the use of discrete signals (0s and 1s) to process and store data. These systems are the backbone of modern technology, and they play a critical role in both space missions and Earth-based applications

3. Telecommunications

Telecommunications play a critical role in space applications, particularly in satellite communication, broadcasting, and space exploration. Here's how space technology and telecommunications are intertwined:

Satellite Communication (SATCOM): Satellites orbit Earth and provide telecommunication services, including internet, television, and radio. Communication satellites are used for relaying signals over long distances, particularly to remote areas that are difficult to reach with terrestrial networks.

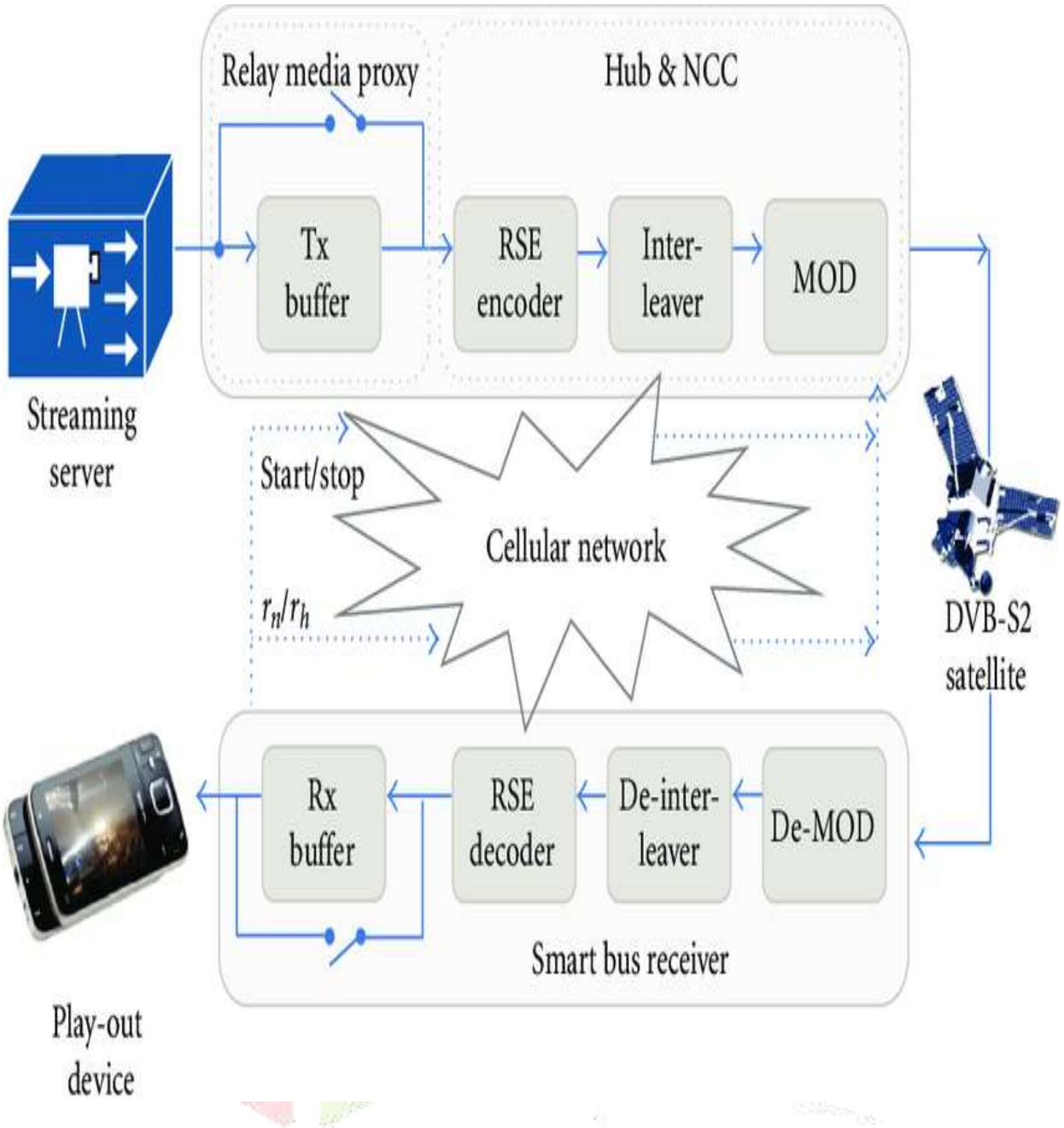
4. Computer Memory

Computer memory, which stores data and program instructions, is essential in space applications where systems need to store large volumes of data for processing and transmission. Memory systems need to be reliable and durable to withstand the harsh conditions of space

Summary:

These four areas—**space engineering**, **digital electronics**, **telecommunications**, and **computer memory**—are deeply interconnected in the context of space technology. Space engineering depends on the efficient use of digital electronics for control and communication, while telecommunications systems enable global connectivity and data transmission from space. At the same time, advanced memory technologies ensure that data is reliably stored, processed, and transmitted, particularly in the challenging environment of space. Together, these fields drive innovation in space exploration, satellite technology, and global communications.

TX/RX CHAIN



CHAPTER V

XILINX AND VERILOG HDL

HISTORY OF VERILOG

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. It is rumored that the original language was designed by taking features from the most popular HDL language of the time, called HiLo, as well as from traditional computer languages such as C. At that time, Verilog was not standardized and the language modified itself in almost all the revisions that came out within 1984 to 1990.

Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog simulator sold by Gateway. The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm" which was a very efficient method for doing gate-level simulation.

The time was late 1990. Cadence Design System, whose primary product at that time included thin film process simulator, decided to acquire Gateway Automation System. Along with other Gateway products, Cadence now became the owner of the Verilog language, and continued to market Verilog as both a language and a simulator.

At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination. In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized the Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event which "opened" the language.

INTRODUCTION

- HDL is an abbreviation of Hardware Description Language. Any digital system can be represented in a REGISTER TRANSFER LEVEL (RTL) and HDLs are used to describe this RTL.
- Verilog is one such HDL and it is a general-purpose language –easy to learn and use. Its syntax is similar to C.
- The idea is to specify how the data flows between registers and how the design processes the data.
- To define RTL, hierarchical design concepts play a very significant role. Hierarchical design methodology facilitates the digital design flow with several levels of abstraction.
- Verilog HDL can utilize these levels of abstraction to produce a simplified and efficient representation of the RTL description of any digital design.
- For example, an HDL might describe the layout of the wires, resistors and transistors on an Integrated Circuit (IC) chip, i.e., the switch level or, it may describe the design at a more micro level in terms

of logical gates and flip flops in a digital system, i.e., the gate level. Verilog supports all of these levels.

DESIGN STYLES:

Any hardware description language like Verilog can be design in two ways one is bottom-up design and other one is top-down design.

Bottom-Up Design:

The traditional method of electronic design is bottom-up (designing from transistors and moving to a higher level of gates and, finally, the system). But with the increase in design complexity traditional bottom-up designs have to give way to new structural, hierarchical design methods.

Top-Down Design:

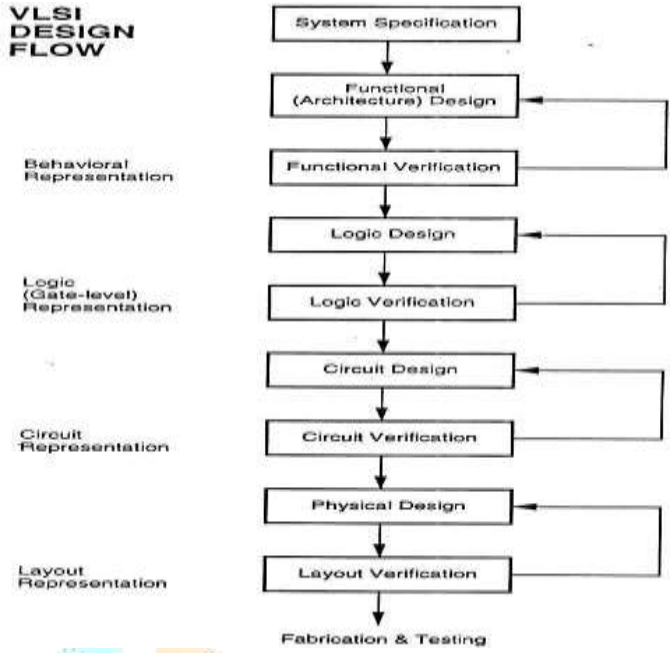
For HDL representation it is convenient and efficient to adapt this design-style. A real top-down design allows early testing, fabrication technology independence, a structured system design and offers many other advantages. But it is very difficult to follow a pure top-down design. Due to this fact most designs are mix of both the methods, implementing some key elements of both design styles.

Features of Verilog HDL

- Verilog is case sensitive.
- Ability to mix different levels of abstract freely.
- One language for all aspects of design, testing, and verification.
- In Verilog, Keywords are defined in lower case.
- In Verilog, Most of the syntax is adopted from "C" language.
- Verilog can be used to model a digital circuit at Algorithm, RTL, Gate and Switch level.
- There is no concept of package in Verilog.
- It also supports advanced simulation features like TEXTIO, PLI, and UDPs.

VLSI DESIGN FLOW

The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip.



System Specification:

The first step of any design process is to lay down the specifications of the system. System specification is a high level representation of the system. The factors to be considered in this process include: performance, functionality, and physical dimensions like size of the chip.

The specification of a system is a compromise between market requirements, technology and economical viability. The end results are specifications for the size, speed, power, and functionality of the VLSI system.

Architectural Design

The basic architecture of the system is designed in this step. This includes, such decisions as RISC (Reduced Instruction Set Computer) versus CISC (Complex Instruction Set Computer), number of ALUs, Floating Point units, number and structure of pipelines, and size of caches among others. The outcome of architectural design is a Micro-Architectural Specification (MAS).

Behavioral or Functional Design:

In this step, main functional units of the system are identified. This also identifies the interconnect requirements between the units. The area, power, and other parameters of each unit are estimated.

Modules. The key idea is to specify behavior, in terms of input, output and timing of each unit, without specifying its internal structure.

The outcome of functional design is usually a timing diagram or other relationships between units.

Logic Design:

In this step the control flow, word widths, register allocation, arithmetic operations, and logic operations of the design that represent the functional design are derived and tested.

This description is called Register Transfer Level (RTL) description. RTL is expressed in a Hardware Description Language (HDL), such as VHDL or Verilog. This description can be used in simulation and verification

Circuit Design:

The purpose of circuit design is to develop a circuit representation based on the logic design. The Boolean expressions are converted into a circuit representation by taking into consideration the speed and power requirements of the original design. Circuit Simulation is used to verify the correctness and timing of each component

The circuit design is usually expressed in a detailed circuit diagram. This diagram shows the circuit elements (cells, macros, gates, transistors) and interconnection between these elements. This representation is also called a netlist. And each stage verification of logic is done.

Physical design:

In this step the circuit representation (or netlist) is converted into a geometric representation. As stated earlier, this geometric representation of a circuit is called a layout.

Layout is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers), which perform the intended logic function of the corresponding component. Connections between different components are also expressed as geometric patterns typically lines in multiple layers.

Layout verification:

Physical design can be completely or partially automated and layout can be generated directly from netlist by Layout Synthesis tools. Layout synthesis tools, while fast, do have an area and performance penalty, which limit their use to some designs. These are verified.

Fabrication and Testing:

Silicon crystals are grown and sliced to produce wafers. The wafer is fabricated and diced into individual chips in a fabrication facility. Each chip is then packaged and tested to ensure that it meets all the design specifications and that it functions properly.

MODULE:

A module is the basic building block in Verilog. It can be an element or a collection of low level design blocks. Typically, elements are grouped into modules to provide common functionality used in places of the design through its port interfaces, but hides the internal implementation.

Syntax:

module<module name> (<module_port_list>);

…..

<module internals> //contents of the module

….

Endmodule

Instances

A module provides a template from where one can create objects. When a module is invoked Verilog creates a unique object from the template, each having its own name, variables, parameters and I/O interfaces. These are known as *instances*.

Ports:

- Ports allow communication between a module and its environment.
- All but the top-level modules in a hierarchy have ports.
- Ports can be associated by order or by name.

You declare ports to be input, output or inout. The port declaration syntax is: *Input* [range_val:range_var] list_of_identifiers; *output*[range_val:range_var] list_of_identifiers;
inout[range_val:range_var] list_of_identifiers;

Identifiers

- Identifiers are user-defined words for variables, function names, module names, and instance names. Identifiers can be composed of letters, digits, and the underscore character.
- The first character of an identifier cannot be a number. Identifiers can be any length.
- Identifiers are case-sensitive, and all characters are significant.

An identifier that contains special characters, begins with numbers, or has the same name as a keyword can

be specified as an escaped identifier. An escaped identifier starts with the backslash character(\) followed by a sequence of characters, followed by white space.

Keywords:

- Verilog uses keywords to interpret an input file.
- You cannot use these words as user variable names unless you use an escaped identifier.
- Keywords are reserved identifiers, which are used to define language constructs.
- Some of the keywords are always, case, assign, begin, case, end and end case etc.
-

Data Types:

Verilog Language has two primary data types:

- ***Nets*** - represents structural connections between components.
- ***Registers*** - represent variables used to store data. Every

signal has a data type associated with it. Data types are:

- ***Explicitly declared*** with a declaration in the Verilog code.
- ***Implicitly declared*** with no declaration but used to connect structural building blocks in the code. Implicit declarations are always net type "wire" and only one bit wide.

Register Data Types

- Registers store the last value assigned to them until another assignment statement changes their value
- Registers represent data storage constructs.
- Register arrays are called memories.
- Register data types are used as variables in procedural blocks.
- A register data type is required if a signal is assigned a value within a procedural block
- Procedural blocks begin with keyword initial and always.

The data types that are used in register are register, integer, time and real.

MODELING CONCEPTS:

Abstraction Levels:

- Behavioral level
- Register-Transfer Level
- Gate Level
- Switch level

Behavioral or algorithmic Level

- This level describes a system by concurrent algorithms (Behavioral).
- Each algorithm itself is sequential meaning that it consists of a set of instructions that are executed one after the other.
- The blocks used in this level are 'initial', 'always' ,'functions' and 'tasks' blocks
- The intricacies of the system are not elaborated at this stage and only the functional description of the individual blocks is prescribed.
- In this way the whole logic synthesis gets highly simplified and at the same time more efficient.

Register-Transfer Level:

- Designs using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers.
- An explicit clock is used. RTL design contains exact timing possibility, operations are scheduled to occur at certain times.
- Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".

Gate Level:

- Within the logic level the characteristics of a system are described by logical links and their timing properties.
- All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates).
- It must be indicated here that using the gate level modeling may not be a good idea in logic design.
- Gate level code is generated by tools like synthesis tools in the form of netlists which are used for gate level simulation and for backend.

OPERATORS

Verilog provided many different operators types. Operators can be,

- Arithmetic Operators
- Relational Operators
- Bit-wise Operators
- Logical Operators
- Reduction Operators
- Shift Operators
- Concatenation Operator
- Conditional Operator

Arithmetic Operators

- These perform arithmetic operations. The + and - can be used as either unary (-z) or binary (x-y) operators.
- Binary: +, -, *, /, % (the modulus operator)
- Unary: +, - (This is used to specify the sign)
- Integer division truncates any fractional part
- The result of a modulus operation takes the sign of the first operand
- If any operand bit value is the unknown value x, then the entire result value is x
- Register data types are used as unsigned values (Negative numbers are stored in two's complement form).

Relational Operators

Relational operators compare two operands and return a single bit 1or 0. These operators synthesize into comparators. Wire and reg variables are positive Thus (-3'b001) = = 3'b111 and (- 3d001)>3d1 10, however for integers -1<>

Operator	Description
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

- The result is a scalar value
- 0 if the relation is false (a is bigger than b)
- 1 if the relation is true (a is smaller than b)
- x if any of the operands has unknown x bits (if a or b contains X)

Note: If any operand is x or z, then the result of that test is treated as false (0)

Bit-wise Operators

Bitwise operators perform a bit wise operation on two operands. This take each bit in one operand and perform the operation with the corresponding bit in the other operand. If one operand is shorter than the other, it will be extended on the left side with zeroes to match the length of the longer operand

Operator	Description
~	negation
&	and
|	inclusive or
^	exclusive or
^~ or ~^	exclusive nor (equivalence)

Computations include unknown bits, in the following way:

-> ~x = x

-> 0&x = 0

-> 1&x = x&x = x

-> 1|x = 1

-> 0|x = x|x = x

-> 0^x = 1^x = x^x = x

-> 0^~x = 1^~x = x^~x = x

When operands are of unequal bit length, the shorter operand is zero-filled in the most significant bit positions.

Logical Operators

Logical operators return a single bit 1 or 0. They are the same as bit-wise operators only for single bit operands. They can work on expressions, integers or groups of bits, and treat all values that are nonzero as "1". Logical operators are typically used in conditional (if ... else) statements since they work with expressions.

Operator	Description
!	logic negation
&&	logical and
\|\| | logical or |

Expressions connected by && and || are evaluated from left to right Evaluation stops

as soon as the result is known

The result is a scalar value:

- 0 if the relation is false
- 1 if the relation is true
- x if any of the operands has x (unknown) bits

Reduction Operators

Reduction operators operate on all the bits of an operand vector and return a single-bit value. These are the unary (one argument) form of the bit-wise operators.

Operator	Description
&	and
~&	nand
\| | or |
~\| | nor |
^ | xor |
^~ or ~^ | xnor |

- Reduction operators are unary.
- They perform a bit-wise operation on a single operand to produce a single bit result.
- Reduction unary NAND and NOR operators operate as AND and OR respectively, but with their outputs negated.

Shift Operators

Shift operators shift the first operand by the number of bits specified by the second operand. Vacated positions are filled with zeros for both left and right shifts (There is no sign extension).

Operator	Description
<<	left shift
>>	right shift

- The left operand is shifted by the number of bit positions given by the right operand.
- The vacated bit positions are filled with zeroes

Concatenation Operator

- The concatenation operator combines two or more operands to form a larger vector.
- Concatenations are expressed using the brace characters { and }, with commas separating the expressions within.
- ->Example: + {a, b[3:0], c, 4'b1001} // if a and c are 8-bit numbers, the results has 24 bits
- Unsized constant numbers are not allowed in concatenations

Operator Precedence

Operator	Symbols
Unary, Multiply, Divide, Modulus	!, ~, *, /, %
Add, Subtract, Shift	+, - , <<, >>
Relation, Equality	<,>,<=,>=,==,!=,===,!==
Reduction	&, !&,^,^~,|,~|
Logic	&&, ||
Conditional	? :

Switch Level:

This is the lowest level of abstraction. A module can be implemented in terms of switches, storage nodes and interconnection between them. However, as has been mentioned earlier, one can mix and match all the levels of abstraction in a design. RTL is frequently used for Verilog description that is a combination of behavioral and dataflow while being acceptable for synthesis.

Xilinx Verilog HDL Tutorial

Getting started

Frist we need to download and install Xilinx and ModelSim. These tools both have free student versions. Please accomplish Appendix B, C, and D in that order before continuing with this tutorial. Additionally if you wish to purchase your own Spartan3 board, you can do so at Digilent's Website. Digilent offers academic pricing. Please note that you must download and install Digilent Adept software. The software contains the drivers for the board that you need and also provides the interface to program the board.

Introduction

Xilinx Tools is a suite of software tools used for the design of digital circuits implemented using Xilinx Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). The design procedure consists of (a) design entry, (b) synthesis and implementation of the design, (c) functional simulation and (d) testing and verification. Digital designs can be entered in various ways using the above CAD tools: using a schematic entry tool, using a hardware description language (HDL) – Verilog or VHDL or a combination of both. In this lab we will only use the design flow that involves the use of Verilog HDL.

The CAD tools enable you to design combinational and sequential circuits starting with Verilog HDL design specifications. The steps of this design procedure are listed below:

1. Create Verilog design input file(s) using template driven editor.
2. Compile and implement the Verilog design file(s).
3. Create the test-vectors and simulate the design (functional simulation) without using a PLD (FPGA or CPLD).
4. Assign input/output pins to implement the design on a target device.
5. Download bitstream to an FPGA or CPLD device.
6. Test design on FPGA/CPLD device

A Verilog input file in the Xilinx software environment consists of the following segments:

Header: module name, list of input and output ports.

Declarations: input and output ports, registers and wires.

Logic Descriptions: equations, state machines and logic functions.

End: endmodule

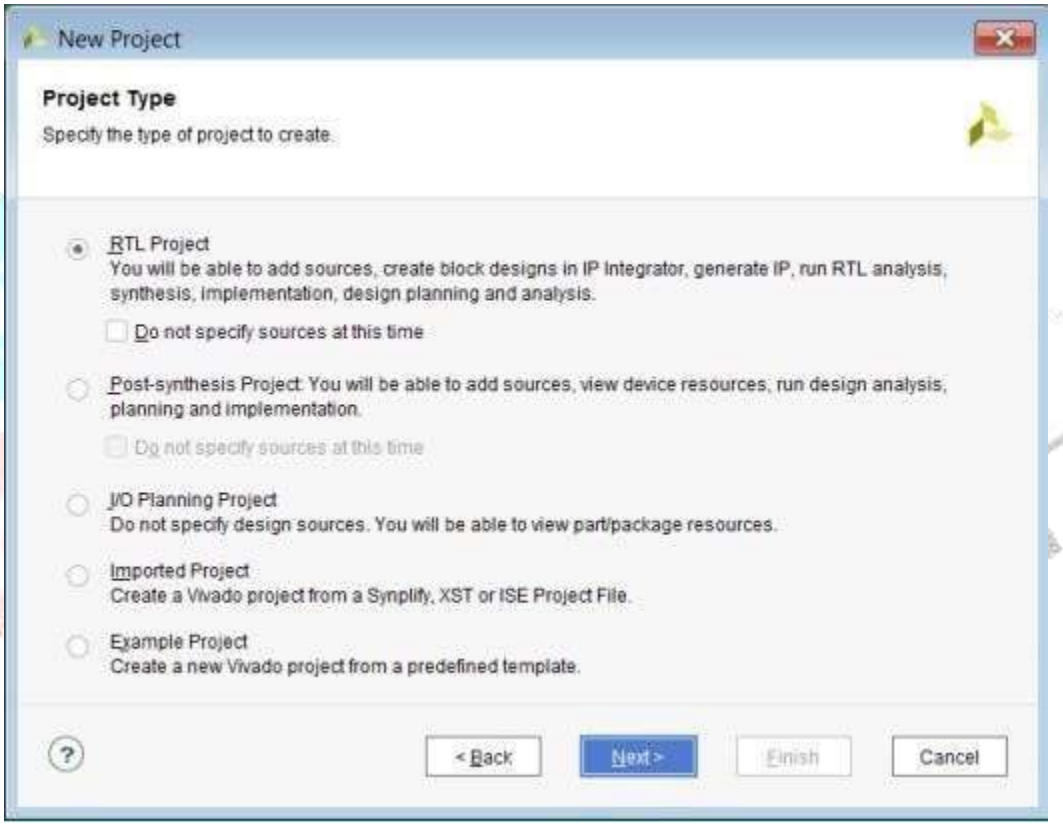
All your designs for this lab must be specified in the above Verilog input format. Note that the

state diagram segment does not exist for combinational logic designs.

Programmable Logic Device: FPGA

In this lab digital designs will be implemented in the Basys2 board which has a Xilinx Spartan3E – XC3S250E FPGA with CP132 package. This FPGA part belongs to the Spartan family of FPGAs. These devices come in a variety of packages. We will be using devices that are packaged in 132 pin package with the following part number: XC3S250E-CP132.

Creating a New Project



Creating Projects You can use the New Project wizard to easily create different types of projects in the Vivado IDE. To open the New Project wizard, select File > New Project. This wizard enables you to specify a project location and name and create the types of projects shown in below figure

New Project Wizard—Project Type Page

Project Name: Write the name of your new project which is user defined.

Project Location: The directory where you want to store the new project in the specified location in one of your drive. In above window they are stored in location c drive which is not correct, the location of software and code should not be same location and Clicking on NEXT. For each of the properties given below, click on the '**value**' area and select from the list of values that appear.

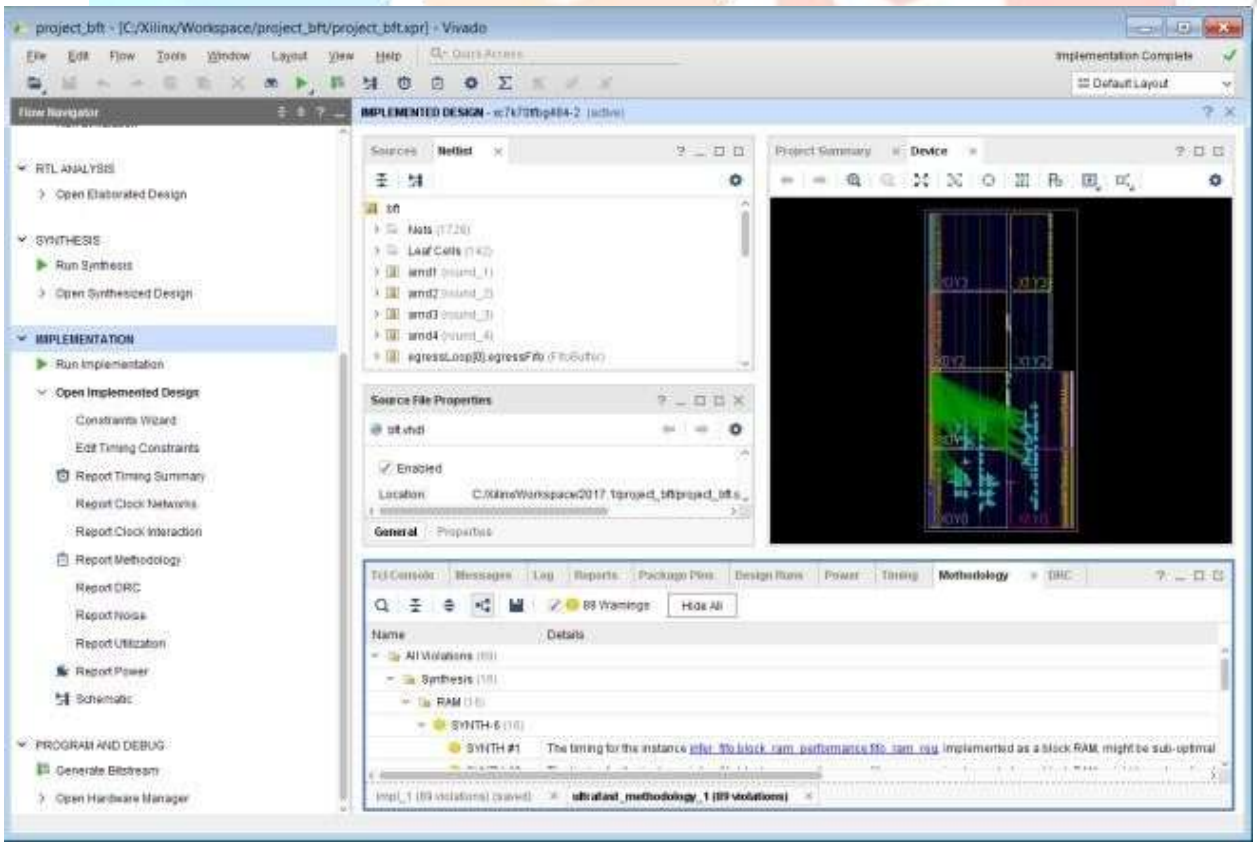
- **Device Family**: Family of the FPGA/CPLD used. In this laboratory we will be using the Spartan3E FPGA's.
- **Device**: The number of the actual device. For this lab you may enter **XC3S250E** (this can be found on the attached prototyping board)
- **Package**: The type of package with the number of pins. The Spartan FPGA used in this lab is packaged in CP132 package.
- **Speed Grade**: The Speed grade is "-4".
- **Synthesis Tool**: **XST** [VHDL/Verilog]
- **Simulator:** The tool used to simulate and verify the functionality of the design. Then click on **NEXT** to save the entries.

Opening Designs:

Use the Flow Navigator or Flow menu to select the following commands:

- Open Elaborated Design
- Open Synthesized Design
- Open Implemented Design

The Flow > Open Implemented Design command populates the Vivado IDE as shown in below figure.



Implemented design

All project files such as schematics, netlists, Verilog files, VHDL files, etc., will be stored in a subdirectory

with the project name.

In order to open an existing project in Xilinx Tools, select **File->Open Project** to show the list of projects on the machine. Choose the project you want and click **OK**.

If creating a new source file, click on the NEW SOURCE.

Creating a Verilog HDL input file for a combinational logic design:

In this lab we will enter a design using a structural or RTL description using the Verilog HDL. You can create a Verilog HDL input file (**.v**file) using the HDL Editor available in the Xilinx Vivado Tools (or any text editor).

In the previous window, click on the NEW SOURCE

(Note: "**Add to project**" option is selected by default. If you do not select it then you will have to add the new source file to the project manually.)

Select **Verilog Module** and in the "File Name:" area, enter the name of the Verilog source file you are going to create. Also make sure that the option **Add to project** is selected so that the source need not be added to the project again. Then click on **Next** to accept the entries.

In the **Port Name** column, enter the names of all input and output pins and specify the **Direction** accordingly. A Vector/Bus can be defined by entering appropriate bit numbers in the **MSB/LSB** columns. Then click on **Next>**to get a window showing all the new source information above window. If any changes are to be made, just click on **<Back** to go back and make changes. If everything is acceptable, click on **Finish > Next > Next > Finish** to continue.

Once you click on **Finish**, the source file will be displayed in the sources window in the **Project Navigator.** If a source has to be removed, just right click on the source file in the **Sources in Project** window in the **Project Navigator** and select **remove** in that. Then select **Project -> Delete Implementation Data** from the Project Navigator menu bar to remove any related files.

Editing the Verilog source file

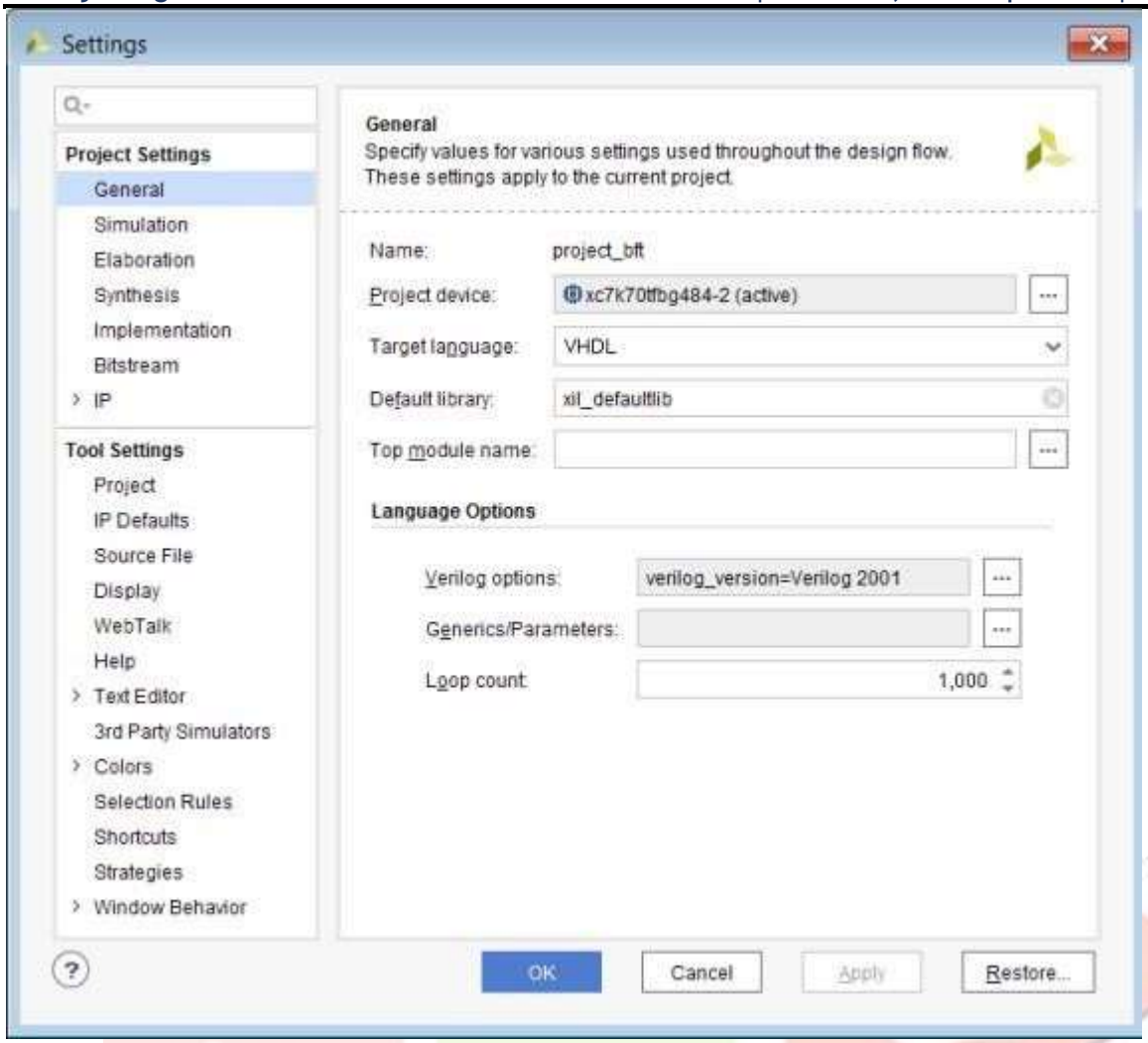
The source file will now be displayed in the **Project Navigator** window (Figure 8). The source file window can be used as a text editor to make any necessary changes to the source file. All the input/output pins will be displayed. Save your Verilog program periodically by selecting the **File->Save** from the menu. You can also edit Verilog programs in any text editor and add them to the project directory using "Add Copy Source".

Here in the above window we will write the Verilog programming code for specified design and algorithm in the window.

After writing the programming code we will go for the synthesis report.

Configuring Project Settings

You can configure the Project Settings in the Settings dialog box to meet your design needs. These settings include general settings, related to the top module definition and language options, as well as simulation, elaboration, synthesis, implementation, bitstream, and IP settings.



Settings Dialog Box—Project Settings General Category

To open the Settings dialog box, use any of the following methods:

- In the Flow Navigator Project Manager section, click Settings.
- Select Tools > Settings.
- In the main toolbar, click the Settings toolbar button .
- In the Project Summary, click the Edit link next to Settings.

Synthesis and Implementation of the Design:

The design has to be synthesized and implemented before it can be checked for correctness, by running functional simulation or downloaded onto the prototyping board. With the top-level Verilog file opened (can be done by double-clicking that file) in the HDL editor window in the right half of the Project Navigator, and the view of the project being in the **Module view** , the **implement design** option can be seen in the **process view**. **Design entry utilities** and **Generate Programming File** options can also be seen in the process view.

To synthesize the design, double click on the **Synthesize Design** option in the **Processes window**. To implement the design, double click the **Implement design** option in the **Processes window**. It will go through steps like **Translate, Map and Place & Route**. If any of these steps could not be done or done with errors,

it will place a **X** mark in front of that, otherwise a tick mark will be placed after each of them to indicate the successful completion

After synthesis right click on synthesis and click view text report in order to generate the report of our design.

XILINX VIVADO SIMULATION PROCEDURE:

After completion of synthesis we will go simulation in order to verify the functionality of the implemented design.

Click on **Run Simulation** and set the module that is need to Run

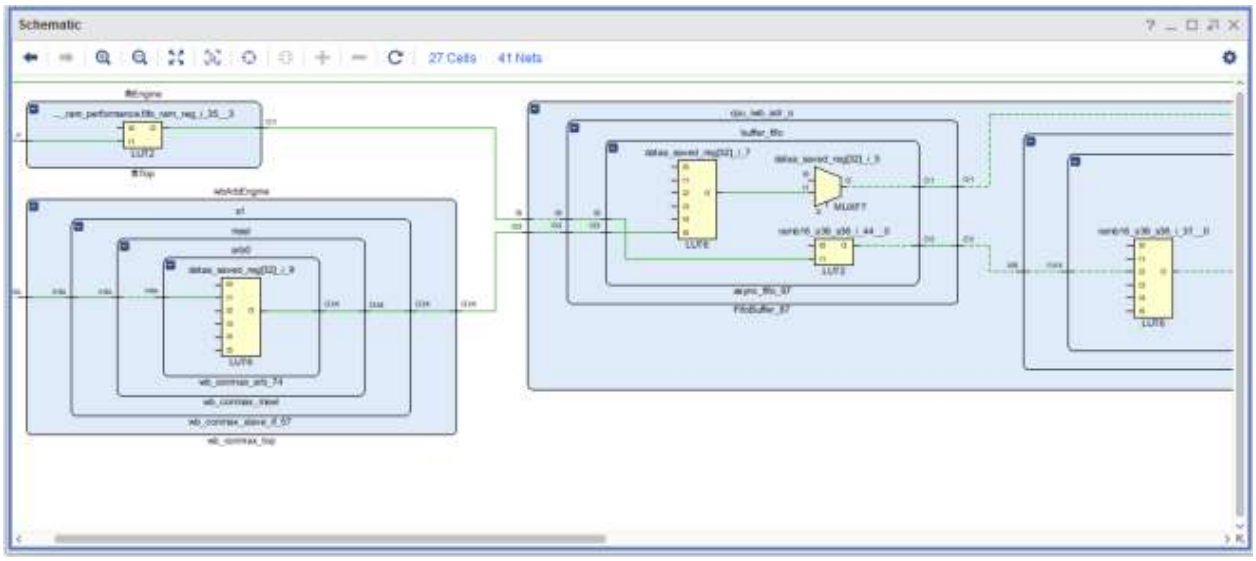
Next **double click on Run Behavioral Simulation** to check the errors. If no errors are found then double click on simulate behavioral model to get the output waveforms.

After clicking on **simulate behavioral model**, the simulation widow will appear pass the input values by making force constant and if it is clock by making force clock. Mention the simulation period and run for certain time and results will appear as shown in following window. Verify the results to the given input values.

Using the Schematic Window:

You can generate a Schematic window for any level of the logical or physical hierarchy. You can select a logic element in an open window, such as a primitive or net in the Netlist window, and use the Schematic command in the popup menu to create a Schematic window for the selected object.

An elaborated design always opens with a Schematic window of the top-level of the design, as shown in below figure.



Schematic window

Using the Project Summary

The Vivado IDE includes an interactive Project Summary (Figure 3-11) that updates dynamically as design commands are run and the design progresses through the design flow. It provides project and design information, such as the project part, board, and state of synthesis and implementation. It also provides links to detailed information, such as links to the Messages and Reports windows as well as the Settings dialog box.

As synthesis and implementation complete, DRC violations, timing values, utilization percentages, and power estimates are also populated. To open the Project Summary, do either of the following:

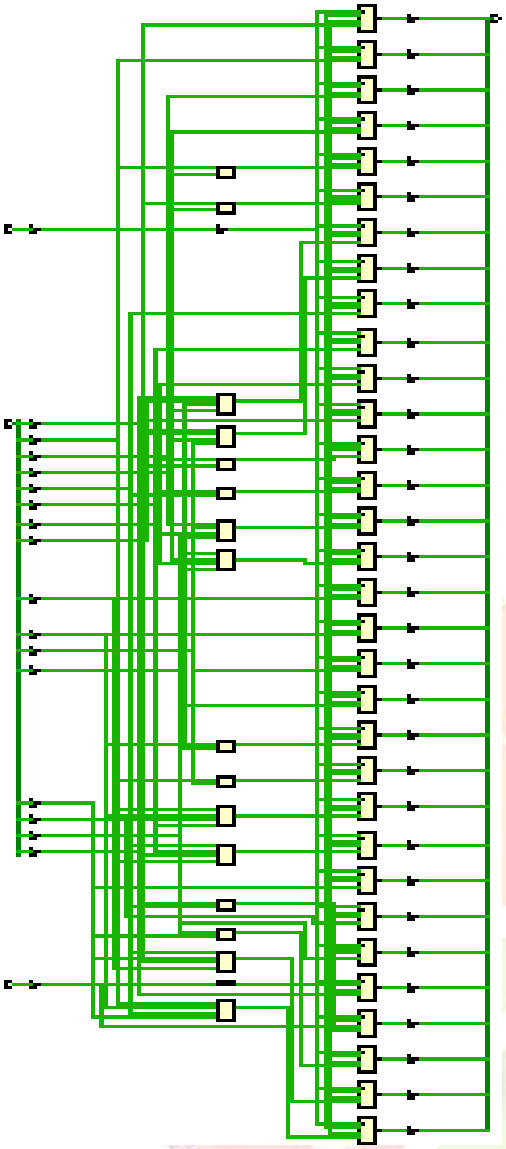
- Select Window > Project Summary.
- Click the Project Summary toolbar button.

- **Project Summary**

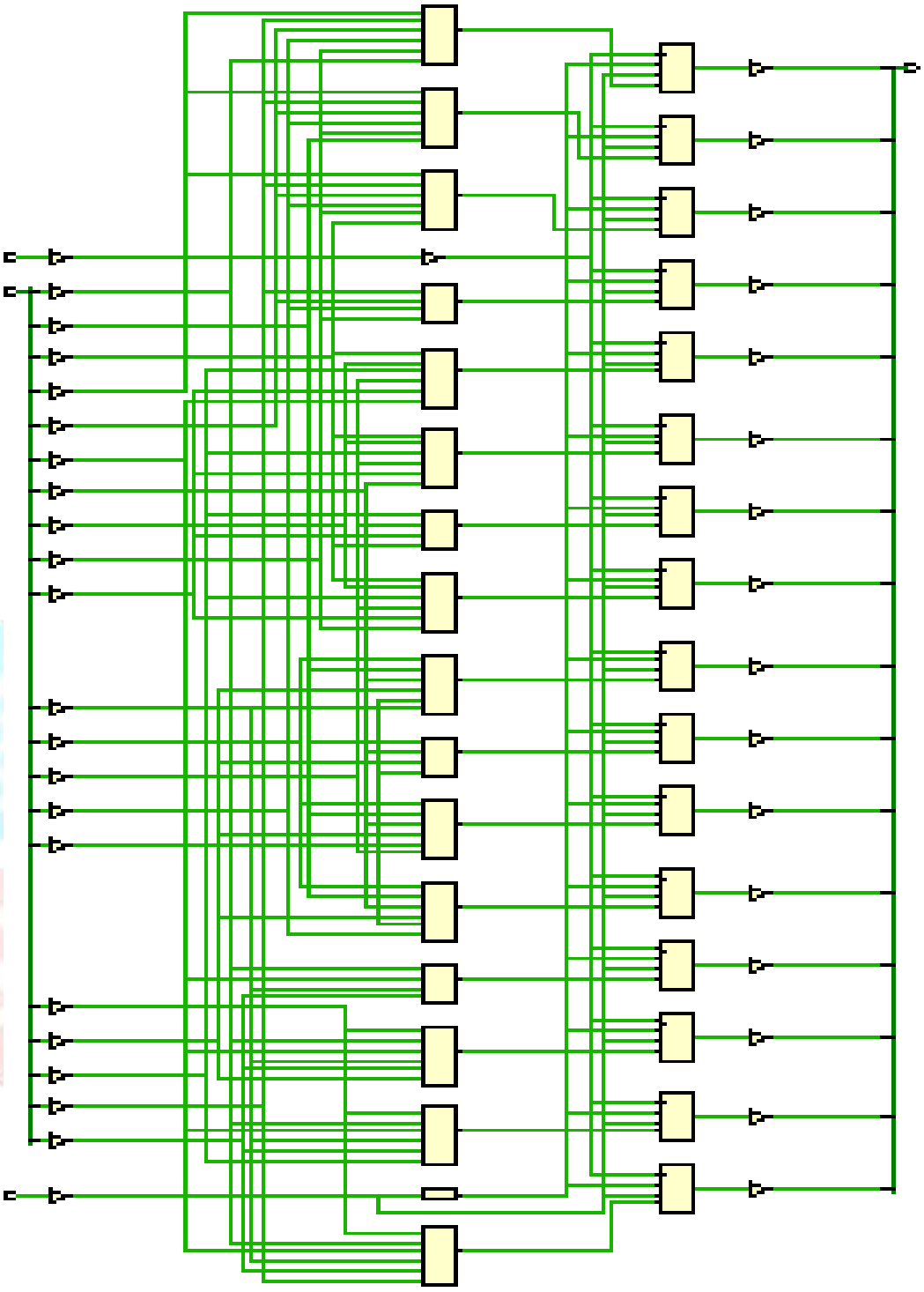
CHAPTER VI

RESULTS

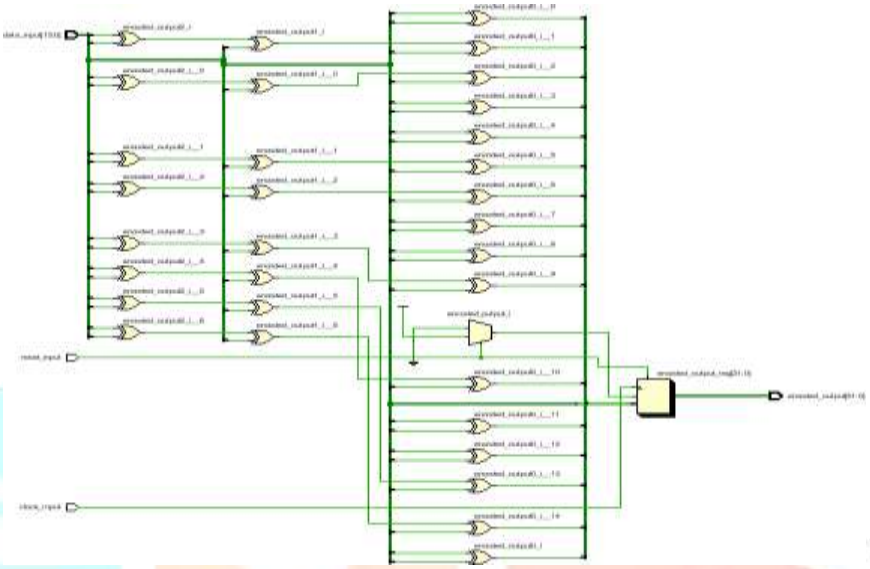
RTL Schematic of Encoder:



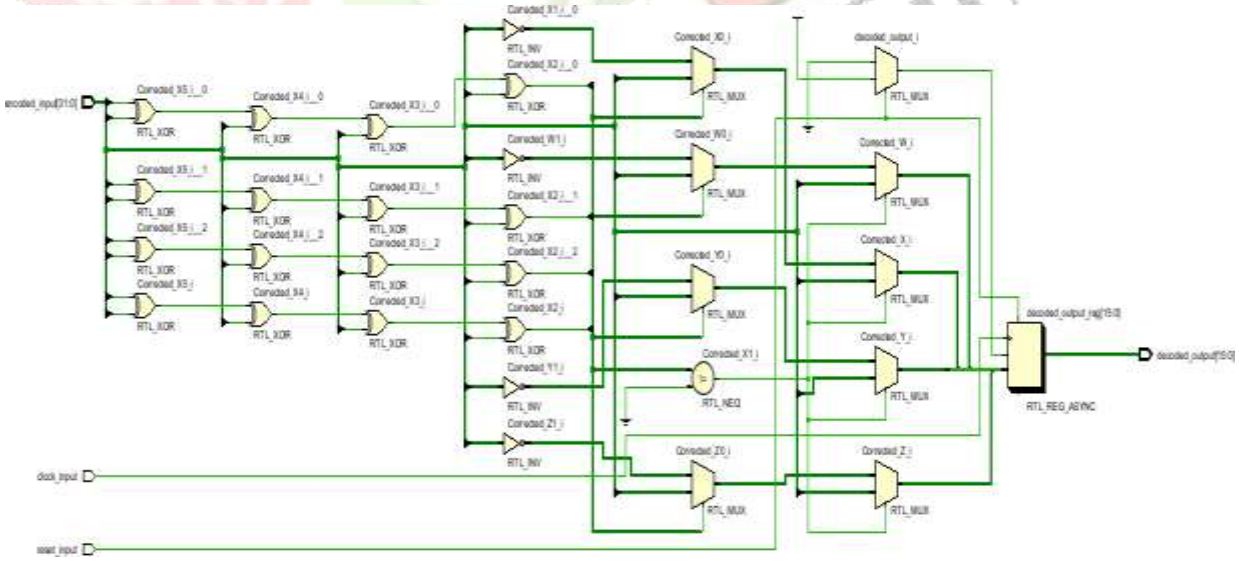
Schematic RTL Decoder:



Elaboration Design of Encoder:



Elaboration Design Decoder:



Encoder Delay:

Statistics

Type	Total Endpoints
Max Delay	128
Min Delay	128

Decoder Delay:

Statistics

Type	Total Endpoints
Max Delay	64
Min Delay	64

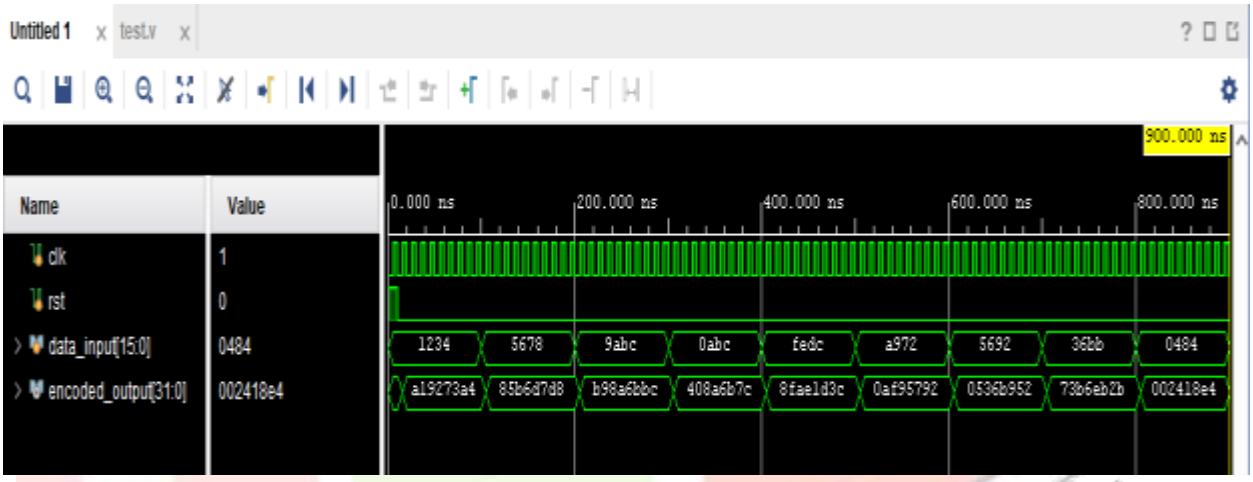
Encoder Resource utilization:

Resource	Utilization	Available	Utilization %
LUT	9	53200	0.02
FF	32	106400	0.03
IO	50	200	25.00

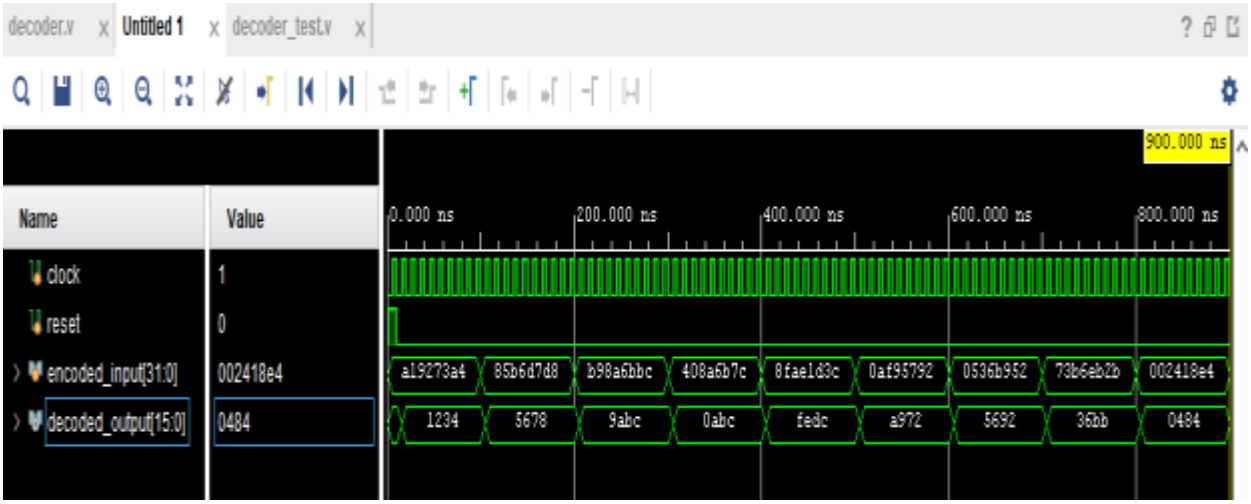
Decoder Resource utilization:

Resource	Utilization	Available	Utilization %
LUT	17	53200	0.03
FF	16	106400	0.02
IO	38	200	19.00

Encoder Simulation:



Decoder Simulation:



CHAPTER VII CONCLUSION

The paper "VLSI Implementation of Error Detection and Correction Codes for Space Engineering" presents a novel approach to enhancing memory reliability in space applications. The authors propose a new error correction 2-dimensional code that effectively detects and corrects errors, thus mitigating the risk of data corruption in volatile memories. This novel scheme was successfully simulated and synthesized using Xilinx Vivado implemented in Verilog HDL.

The power consumed during the encoding and decoding process was found to be significantly low, and the area and delay occupied were minimal compared to existing methods. This makes the proposed algorithm a viable solution for space engineering, where energy efficiency is paramount. The algorithm is expected to be extended to further reduce the area, delay, and power consumption, making it an ideal choice for harsh environments where resources are scarce.

The authors also demonstrated that the proposed algorithm can detect and correct multi-bit errors, a critical feature for maintaining data integrity in space applications. The algorithm's effectiveness was validated through simulation results, showing that after decoding, the original data is accurately retrieved, proving its effectiveness in error correction and detection.

In conclusion, the paper presents a promising solution for error correction and detection in multiple cell upset (MCUs) in space engineering. The proposed algorithm provides a way to enhance memory reliability, a critical aspect in space applications. Its successful simulation and synthesis, combined with its low power consumption and minimal area and delay, make it an attractive choice for future implementations. However, the authors acknowledge that there is room for improvement, particularly in terms of reducing the decoder area. Future work could focus on exploring more advanced region selection criteria to further optimize the algorithm.

REFERENCES:

- R. C. Baumann (2005), "Soft errors in advanced computer systems," IEEE Des. Test. Comput., vol. 22, no. 3, pp. 258-266.
- C. L. Chen and M. Y. Hsiao (1984), "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," IBM J. Res. Develop., vol.28, no. 2, pp. 124-134.
- E. Ibe, S. Chung, S. Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto and T.Akioka (2006), "Spreading diversity in multi-cell neutron-induced upsets with device scaling," in Proc. IEEE Custom Integrated Circuit Conf., pp. 437-444.
- P. Reviriego, J.A. Maestro and C. Cervantes (2007), "Reliability analysis of memories suffering multiple bit upsets," IEEE Trans. Device Mater. Rel., vol. 7, no. 4, pp. 592-601.
- G. C. Yang (1995), "Reliability of semiconductor RAMs with soft-error scrubbing techniques," IEEE Proc. Computers and Digital techniques, vol. 142, no. 5, pp. 337-344.
- P. Reviriego, J. Maestro, S. Wen and R. Wong (2010), "Protection of memories suffering MCUs through the selection of the optimal interleaving distance," IEEE Trans. On nuclear science., vol. 57, no. 4, pp. 2124-2128.
- Z. Ming, X. Li Yi and L. Hong Wei (2011), "New SEC-DED-DAEC codes for multiple bit upsets mitigation in memory," IEEE/IFIP 19th international conference on VLSI and system-on-chip., pp. 254-259.
- P. Reviriego, S. Liu, L. Xiao and J. Maestro (2015), "An efficient single and double- adjacent error correcting parallel decoder for the (24, 12) extended golay code," IEEE Trans. On very large scale integration (VLSI) systems, pp. 1-4.
- J. Guo, L. Xiao, Z. Mao and Q. Zhao (2013), "Enhanced memory reliability against multiple cell upsets using decimal matrix code," IEEE Trans. On very large scale integration (VLSI) systems., pp. 1-4.