A

Major Project Report on

# DESIGN AND VERIFICATION OF VENDING MACHINE USING VERILOG HDL

Submitted in partial fulfilment of the requirement for the award of degree of

**BACHELOR OF TECHNOLOGY IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

SUBMITTED BY

| | |
|---|---|
| **OBINENI NITHIN** | **218R1A04N9** |
| **PAGIDIMARRI PRAVEEN** | **218R1A04O0** |
| **PANJALA KISHORE** | **218R1A04O1** |
| **PAPPU PURNACHANDU** | **218R1A04O2** |

Under the Esteemed Guidance of

**Ms. L. LAVANYA**
**Assistant Professor**

**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING**

**CMR ENGINEERING COLLEGE**

**UGC AUTONOMOUS**

**(Approved by AICTE, UGC AUTONOMOUS, Accredited by NBA, NAAC)**
**Kandlakoya(V), Medchal(M), Telangana.**

**(2024-2025)**

# CMR ENGINEERING COLLEGE

**UGC AUTONOMOUS**

**(Approved by AICTE, UGC AUTONOMOUS, Accredited by NBA, NAAC)**
**Kandlakoya (V), Medchal , Telangana.**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

## CERTIFICATE

This is to certify that Major-project work entitled **"DESIGN AND VERIFICATION OF VENDING MACHINE USING VERILOG HDL"** is being submitted **O. NITHIN** bearing Roll No **218R1A04N9, P. PRAVEEN** bearing Roll No **218R1A04O0, P. KISHORE** bearing Roll No **218R1A04O1**, **P. PURNA CHANDU** bearing Roll No **218R1A04O2** in B.Tech IV-II semester, Electronics and Communication Engineering is a record Bonafide work carried out by then during the academic year 2024-25. The results embodied in this report have not been submitted to any other University for the award of any degree.

| | |
|---|---|
| INTERNAL GUIDE | HEAD OF THE DEPARTMENT |
| **Ms. L. LAVANYA** | **Dr. SUMAN MISHRA** |

**EXTERNAL EXAMINER**

# ACKNOWLEDGEMENTS

We sincerely thank the management of our college **CMR Engineering College** for providing required facilities during our project work. We derive great pleasure in expressing our sincere gratitude to our Principal **Dr. A. S. Reddy** for his timely suggestions, which helped us to complete the project work successfully. It is the very auspicious moment we would like to express our gratitude to **Dr. SUMAN MISHRA,** Head of the Department, ECE for his consistent encouragement during the progress of this project.

We take it as a privilege to thank our project coordinator **Dr. T. SATYANARAYANA**, Associate Professor, Department of ECE for the ideas that led to complete the project work and we also thank him for his continuous guidance, support and unfailing patience, throughout the course of this work. We sincerely thank our project internal guide **Ms. L. LAVANYA,** Assistant Professor of ECE for guidance and encouragement in carrying out this project work.

## DECLARATION

We hereby declare that the major project entitled **"DESIGN AND VERIFICATION OF VENDING MACHINE USING VERILOG HDL"** is the work done by us in campus at **CMR ENGINEERING COLLEGE,** Kandlakoya during the academic year 2024-2025 and is submitted as Major project in partial fulfilment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING FROM JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD.**

| | |
|---|---|
| **O. NITHIN** | **(218R1A04N9)** |
| **P. PRAVEEN** | **(218R1A04O0)** |
| **P. KISHORE** | **(218R1A04O1)** |
| **P. PURNA CHANDU** | **(218R1A04O2)** |

# ABSTRACT

The design, implementation, and verification of a vending machine using the Finite State Machine (FSM) methodology in Verilog HDL. The FSM is used to manage the multiple states of the vending machine, including “idle,” “accepting coins,” “dispensing item,” and “returning change.” The implementation of the vending machine is done in Verilog HDL, and the FSM is implemented as a state diagram. The design is then synthesized using the Genus synthesis tool and implemented using the Encounter implementation tool. The Genus tool uses advanced optimization techniques, such as timing-driven placement and clock tree synthesis, to improve the design’s performance and area. The Encounter tool performs physical design, including placement and routing, to meet the design’s timing, power, and area constraints. To validate the design’s correctness and functionality, a test bench is created to simulate the behavior of the vending machine. The simulation results are then used to verify that the design meets the required specifications and that the FSM behaves as expected. The proposed design is then can be implemented on a Field Programmable Gate Array (FPGA) to demonstrate its effectiveness in a real-world scenario. The results of the implementation are presented and analyzed to validate the design’s performance, power consumption, and area. Overall, the vending machine using FSM in Verilog HDL, implemented in Genus and Encounter, provides a reliable and efficient solution for users to purchase items from the machine. The proposed design and implementation demonstrate the feasibility and effectiveness of this approach, and the results show that the design meets the required specifications and performs well in a real-world scenario.

# CONTENTS

# LIST OF FIGURES

# CHAPTER 1
# INTRODUCTION

Vending machines have become an integral part of modern automated retail systems, providing quick and efficient delivery of products such as snacks, beverages, and other small items. These machines play a significant role in various locations, including offices, schools, hospitals, shopping malls, and transportation hubs, where they offer convenience and round-the-clock availability of goods. The primary advantage of vending machines is their ability to operate without human intervention, reducing operational costs and enhancing efficiency.

## 1.1 INTRODUCTION

The functionality of a vending machine is controlled by an embedded system that processes user inputs, validates transactions, and dispenses selected items. These systems typically consist of multiple hardware and software components, including sensors, actuators, microcontrollers, and payment processing mechanisms. Advanced vending machines integrate cashless payment options, such as credit/debit cards, mobile wallets, and QR code scanning, improving accessibility and user experience.

To ensure reliability and accuracy, vending machines employ digital logic circuits to handle various operations, such as coin detection, product selection, and change dispensing. The implementation of these digital circuits requires precise design methodologies that guarantee smooth transaction processing and error-free operation. This has led to the adoption of Hardware Description Languages (HDLs) such as Verilog for designing and implementing vending machine controllers.

With the advancement of digital design techniques, Verilog HDL has gained popularity for modeling, simulation, and verification of digital circuits. It allows designers to develop efficient vending machine controllers that meet industry standards and operational requirements. Verilog HDL facilitates the implementation of finite state machines (FSMs) to manage different states of a vending machine, including idle, coin acceptance, product selection, transaction validation, and item dispensing. where they offer convenience and round-the-clock availability of goods. The primary advantage of vending machines is their ability to operate without human intervention, reducing operational costs and enhancing efficiency.

This project focuses on the design and verification of a vending machine using Verilog HDL. The system will be modeled as a finite state machine (FSM) to handle coin insertion, product selection, and item dispensing while ensuring correct transaction processing. The design will be thoroughly tested and verified using simulation techniques to validate its correctness, efficiency, and reliability. Through this project, a fully functional vending machine controller will be developed, demonstrating the application of digital design principles in real-world automated systems.

## 1.2. OBJECTIVE

The objective of this project is to design, implement, and verify a digital vending machine using Finite State Machine (FSM) methodology in Verilog HDL. The vending machine is designed to handle multiple functions, including coin acceptance, item selection, dispensing, and returning change, ensuring smooth and efficient transactions.

**1. Design Implementation**

The vending machine is implemented in Verilog HDL, a hardware description language suitable for designing digital circuits. The system is divided into different modules, including coin processing, product selection, dispensing mechanism, and change return logic. The design follows a structured approach, ensuring modularity and scalability.

**2. Finite State Machine (FSM) Development**

FSM methodology is employed to manage the vending machine's operations efficiently. The system transitions through various states, including:

- **Idle State:** The machine waits for user input.
- **Coin Acceptance State:** The inserted coins are validated and stored.
- **Selection State:** The user selects a product, and the system verifies sufficient balance.
- **Dispensing State:** The selected item is dispensed.
- **Change Return State:** If required, the remaining balance is returned to the user. This structured FSM approach ensures smooth operation and error handling.

**3. Synthesis and Implementation**

The Verilog design is synthesized using Cadence Genus, optimizing for performance, power, and area (PPA). The implementation is further refined using Cadence Encounter, ensuring minimal area usage, efficient power consumption, and faster processing.

**4. Verification**

To ensure correctness, a test bench is created for simulation. The test bench verifies different scenarios, including valid/invalid inputs, product availability, and balance handling, ensuring the FSM behaves as expected.

**5. Real-World Application**

The final design is deployed on an FPGA board to demonstrate its feasibility in real-world applications. It interacts with physical components like buttons, displays, and actuators, validating real-time performance and making it suitable for commercial use.

## 1.3 OVERVIEW OF THE PROJECT

The project "Design and Verification of Vending Machine Using Verilog HDL" focuses on the development of a digital vending machine using Finite State Machine (FSM) methodology. The vending machine transitions through multiple states, including idle, coin acceptance, validation, product dispensing, and change return. FSM-based design ensures structured state transitions, improving reliability and operational efficiency

**Finite State Machine (FSM) Approach**

FSM is a sequential circuit design method that ensures the vending machine correctly handles user inputs and system operations. The major states include:

- **Idle State:** The system remains in standby mode until a coin is inserted.
- **Coin Acceptance State:** The machine detects, verifies, and accumulates the inserted coins.
- **Validation State:** It checks if the inserted amount is sufficient for the selected product.
- **Product Selection and Dispensing State:** If the balance is adequate, the selected product is dispensed.
- **Change Return State:** Any remaining balance is returned to the user.
- **Error Handling State:** Manages invalid selections, insufficient funds, or incorrect inputs.

FSM ensures efficient state management and minimizes errors in user interactions.

Verilog HDL Implementation

The vending machine is designed using Verilog HDL, a hardware description language suitable for modeling digital circuits. The design is divided into modular components, including:

- **Coin Processing Module:** Registers and validates coin inputs.
- **Product Selection Module:** Detects and processes user selections.
- **Control Logic Module:** Implements FSM to control system operations.
- **Dispensing Mechanism Module:** Controls the release of selected products and change return.

This modular approach enhances design scalability, flexibility, and debugging efficiency. Verification through Testbench Simulation to ensure the design functions as expected, a testbench is created to simulate different operational scenarios:

- Valid and invalid coin insertions.
- Correct and incorrect product selections.
- Proper handling of insufficient balance.
- Ensuring correct change return and reset conditions.

Tools like Model Sim or Xilinx Vivado are used for functional verification.

Synthesis and FPGA Deployment. The Verilog code is synthesized using Cadence Genus, optimizing performance, power, and area (PPA). It is implemented using Cadence Encounter and tested on an FPGA board with real-time buttons, displays, and actuators for item dispensing.

# CHAPTER 2
# LITERATURE SURVEY

## 2.1. EXISTING SYSTEM

In modern vending machines, many designs employ microcontrollers or embedded systems for high-level control, often complemented by hardware description languages (HDLs) like Verilog to handle time-critical tasks. A widely adopted method for implementing vending machine logic involves using a Finite State Machine (FSM) architecture, typically implemented in Verilog, to manage the various operational states of the machine.

**Description of the Existing Method:**

The core of the vending machine operation is managed through a Finite State Machine (FSM)**,** where each state represents a specific operation (e.g., idle, accepting coins, dispensing items, returning change). The FSM architecture is implemented using Verilog HDL to ensure efficient, fast, and reliable state transitions based on the user inputs, coin detection, and machine status.

In this existing method, the overall vending machine control is distributed between:

1. **A Microcontroller or Embedded Processor**: Handles high-level tasks such as managing user interactions, interpreting inputs, and controlling the overall sequence of operations. The microcontroller might interface with a keypad or touchscreen for item selection and payment options.
2. **Verilog-based FSM**: Responsible for controlling hardware components like coin acceptance, item dispensing, and change return. The FSM ensures that the machine behaves predictably under different conditions, transitioning between various states based on user actions (inserting coins, selecting items, etc.).
3. **Peripheral Devices and Control Logic**: Verilog is also used to implement the control logic for peripherals like motors, solenoids, coin acceptors, and sensors. For example, the FSM in Verilog can trigger a motor to dispense an item or control the solenoid to return change.

**Disadvantages of the Existing Method:**

1. **High Complexity in Hardware Design:**

- Requires specialized knowledge of Verilog and digital logic design.
- Steep learning curve for engineers unfamiliar with hardware description languages.

2. **Long Design and Testing Cycles:**
- Any changes to the FSM or hardware logic require full re-synthesis and re-testing.
- Time-consuming iteration and debugging process.
3. **Power Consumption:**
- FPGA-based designs generally consume more power than microcontroller-based solutions.
- Higher power usage may increase operational costs, especially in large deployments.
4. **Difficulty in Handling Software-like Tasks:**
- Verilog is not suitable for complex, high-level tasks like encryption, dynamic pricing, or mobile payment integration.
- May require additional microcontrollers or systems to handle non-time-critical operations.
5. **Hardware Dependency and Difficulty in Upgrades:**
- Changes to the hardware or peripherals require significant changes to the Verilog code.
- Upgrades are not as flexible as software-based systems and may require reprogramming the FPGA.
6. **Difficulty in Scaling for More Complex Systems:**
- As the system grows (e.g., more items, advanced payment options), the FSM becomes harder to manage.
- Scaling the design can lead to increased complexity and potential resource exhaustion.

**[1] Chopde, Abhay, et al. "Vending Machine Using Verilog (FPGA)."** ***2024 4th Asian Conference on Innovation in Technology (ASIANCON).*** **IEEE, 2024.**

The design and implementation of a vending machine system using Verilog HDL on an FPGA board. The vending machine is equipped with multiple states including product selection, amount selection, dispensing, out-of-stock detection, refund processing, change calculation, and system reset. The project aims to demonstrate the integration of hardware description language (Verilog) with FPGA technology to create a functional vending machine prototype.The design process involved the decomposition of the vending machine's functionality into smaller modules, each responsible for a specific task. These modules were then integrated to form the complete system.

**[2] Fuad, Mahamudul Hassan, et al. "Design of a Vending Machine Using Verilog HDL and Implementation in Genus & Encounter."** ***European Journal of Electrical Engineering and Computer Science*** **7.6 (2023): 88-95**.

This paper proposes the design, implementation, and verification of a vending machine using the Finite State Machine (FSM) methodology in Verilog HDL. The FSM is used to manage the multiple states of the vending machine, including “idle,” “accepting coins,” “dispensing item,” and “returning change.” The implementation of the vending machine is done in Verilog HDL, and the FSM is implemented as a state diagram. The design is then synthesized using the Genus synthesis tool and implemented using the Encounter implementation tool. The Genus tool uses advanced optimization techniques, such as timing-driven placement and clock tree synthesis, to improve the design’s performance and area. The Encounter tool performs physical design, including placement and routing, to meet the design’s timing, power, and area constraints. To validate the design’s correctness and functionality, a test bench is created to simulate the behaviour of the vending machine. The simulation results are then used to verify that the design meets the required specifications and that the FSM behaves as expected. The proposed design is then can be implemented on a Field Programmable Gate Array (FPGA) to demonstrate its effectiveness in a real-world scenario. The results of the implementation are presented and analyzed to validate the design’s performance, power consumption, and area. Overall, the vending machine using FSM in Verilog HDL, implemented in Genus and Encounter, provides a reliable and efficient solution for users to purchase items from the machine. The proposed design and implementation demonstrate the feasibility and effectiveness of this approach, and the results show that the design meets the required specifications and performs well in a real-world scenario.

**[3] Ravikumar, P., MV Ganeswara Rao, and Vamaraju Nikitha. "Verilog Based Automated Retail System."** ***2024 OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0*****. IEEE, 2023.**

The abstract describes an approach for implementing a vending machine using Verilog HDL. Verilog HDL is a hardware description language that can be used to model digital circuits. The proposed approach involves designing the vending machine using Verilog HDL and simulating the design to ensure it meets the required specifications. The design is then synthesized and implemented on a field-programmable gate array (FPGA). The vending machine is capable of accepting coins and dispensing products based on the user's selection.

## 2.2 PROPOSED SYSTEM

An abstract machine that can only exist in one of the finite states at any one moment is called a finite state machine. It is a synchronous sequential machine described by an abstract model. A sequential machine's most basic model includes inputs, outputs, and internal states. A machine might have an endless number of alternative histories, therefore it would require an infinite capacity for storing them, since the output of a sequential circuit depends on both the current input and the previous inputs, or past histories. We only take into consideration finite state machines because it is not possible to create machines with limitless storage capacity.

Finite state machines are sequential circuits whose past histories can affect their future behaviour in only a finite number of ways, i.e., they are machines with a fixed number of states. These machines can distinguish among a finite number of classes of input histories. These classes of input histories are referred to as internal states of the machine. Every finite state machine therefore contains a finite number of memory devices. Problem Formulation: The design of a vending machine that dispenses three different priced products and takes coins with denominations of one, two, five and ten is the difficulty. The machine should also have the ability to return money upon request cancellation and change back when a coin of a higher denomination is input. The consumer is prompted by the machine to choose the product before inserting coins according to the product's price.

Therefore, the machine receives the select and coin signals as inputs. There is one more input, the cancel option. When the inserted quantity and the selected product's pricing match, the machine dispenses the product. The machine delivers the customer the product and the change if they insert a coin with a larger denomination. In the event that the machine does not have change, it returns the total. Thus, the machine's outputs are product and change. If a request is cancled, the money is repaid. Return money is thus an additional output. Two registers are required: one to record the number of coins in the current transaction and the other to record the overall number of coins in the machine. It is assumed that each register has ten bits of width. Change can only be provided when a higher denomination coin is inserted and the overall coin count exceeds the coin count of the current transaction. The machine determines if the change is available in this manner. These machines can distinguish among a finite number of classes of input histories.

These classes of input histories are referred to as internal states of the machine. Every finite state machine therefore contains a finite number of memory devices. Problem Formulation: The design of a vending machine that dispenses three different priced products and takes coins with denominations of one, two, five and ten is the difficulty. The machine should also have the ability to return money upon request cancellation and change back when a coin of a higher denomination is input. A machine might have an endless number of alternative histories, therefore it would require an infinite capacity for storing them, since the output of a sequential circuit depends on both the current input and the previous inputs, or past histories.

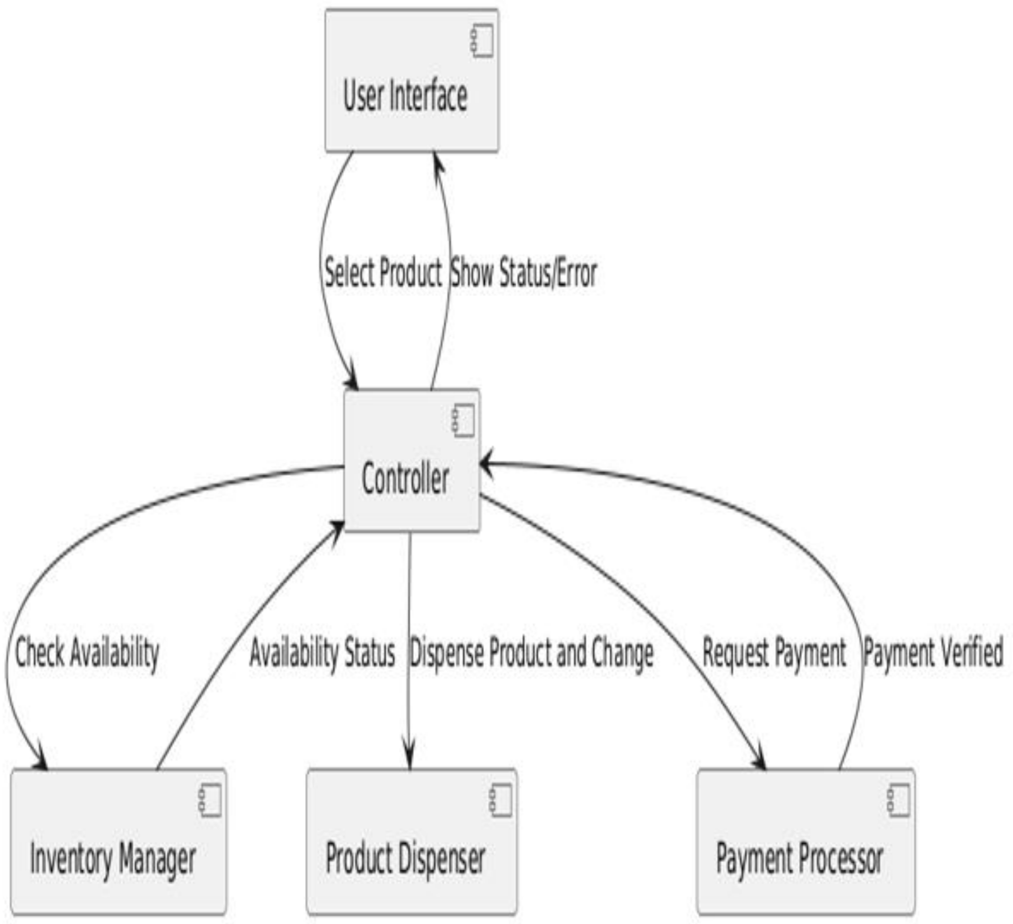


**FIG 2.1: SIMPLIFIED VENDING MACHINE ARCHITECTURE**

The functionality of a vending machine is controlled by an embedded system that processes user inputs, validates transactions, and dispenses selected items. These systems typically consist of multiple hardware and software components, including sensors, actuators, microcontrollers, and payment processing mechanisms.

Advanced vending machines integrate cashless payment options, such as credit/debit cards, mobile wallets, and QR code scanning, improving accessibility and user experience.
To ensure reliability and accuracy, vending machines employ digital logic circuits to handle various operations, such as coin detection, product selection, and change dispensing. The implementation of these digital circuits requires precise design methodologies that guarantee smooth transaction processing and error-free operation. This has led to the adoption of Hardware Description Languages (HDLs) such as Verilog for designing and implementing vending machine controllers. A machine might have an endless number of alternative histories, therefore it would require an infinite capacity.

**Flow Chart:**

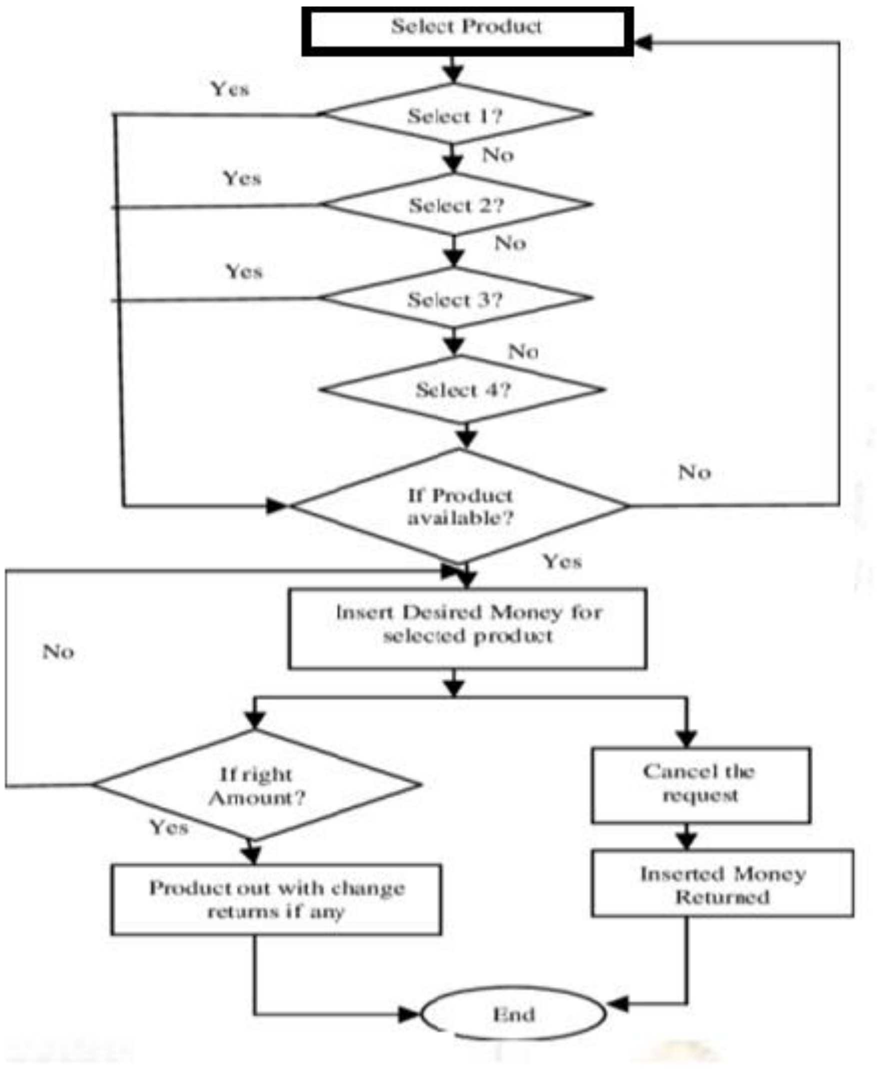


**FIG 2.2: FLOW CHART**

The product must be chosen first, and then the coins must be inserted. As coins are added, the count is increased. The machine decrements count and restores inserted money in the event that a cancel signal is delivered. If not, the device compares the amount of money entered with the cost of the chosen item. The machine dispenses the right product if both are equal. The machine will provide the right goods and change if the amount entered exceeds the pricing. The machine waits for the user to insert more coins if the amount of money inserted is less than the cost of the item they have chosen.

## 2.3. INTRODUCTION TO VLSI

### 2.3.1. VLSI TECHNOLOGY

VLSI Design presents state-of-the-art papers in VLSI design, computer-aided design, design analysis, design implementation, simulation and testing. Its scope also includes papers that address technical trends, pressing issues, and educational aspects in VLSI Design. The Journal provides a dynamic high-quality international forum for original papers and tutorials by academic, industrial, and other scholarly contributors in VLSI Design.

The development of microelectronics spans a time which is even lesser than the average life expectancy of a human, and yet it has seen as many as four generations. Early 60's saw the low density fabrication processes classified under Small Scale Integration (SSI) in which transistor count was limited to about 10. This rapidly gave way to Medium Scale Integration in the late 60's when around 100 transistors could be placed on a single chip.
It was the time when the cost of research began to decline and private firms started entering the competition in contrast to the earlier years where the main burden was borne by the military. Transistor-Transistor logic (TTL) offering higher integration densities outlasted other IC families like ECL and became the basis of the first integrated circuit revolution. It was the production of this family that gave impetus to semiconductor giants like Texas Instruments, Fairchild and National Semiconductors. Early seventies marked the growth of transistor count to about 1000 per chip called the Large Scale Integration.

By mid-eighties, the transistor count on a single chip had already exceeded 1000 and hence came the age of Very Large Scale Integration or VLSI. Though many improvements have been made and the transistor count is still rising, further names of generations like ULSI are generally avoided. It was during this time when TTL lost the battle to MOS family owing

to the same problems that had pushed vacuum tubes into negligence, power dissipation and the limit it imposed on the number of gates that could be placed on a single die.
The second age of Integrated Circuits revolution started with the introduction of the first microprocessor, the 4004 by Intel in 1972 and the 8080 in 1974. Today many companies like Texas Instruments, Infineon, Alliance Semiconductors, Cadence, Synopsys, Celox Networks, Cisco, Micron Tech, National Semiconductors, ST Microelectronics, Qualcomm, Lucent, Mentor Graphics, Analog Devices, Intel, Philips, Motorola and many other firms have been established and are dedicated to the various fields in "VLSI" like Programmable Logic Devices, Hardware Descriptive Languages, Design tools, Embedded Systems etc.
In 1980s, hold-over from outdated taxonomy for integration levels. Obviously, influenced from frequency bands, i.e., HF, VHF, and UHF. Sources disagree on what is measured (gates or transistors)

SSI – Small-Scale Integration (0-102)

MSI – Medium-Scale Integration (102 -103)

LSI – Large-Scale Integration (103 -105)

VLSI – Very Large-Scale Integration (105 - 107)

ULSI – Ultra Large-Scale Integration (>= 107)

The company was based in Silicon Valley, with headquarters at 1109 McKay Drive in San Jose, California. Along with LSI Logic, VLSI Technology defined the leading edge of the application-specific integrated circuit (ASIC) business, which accelerated the push of powerful embedded systems into affordable products. The company was founded in 1979 by a trio from Fairchild Semiconductor by way of Synertek - Jack Balletto, Dan Floyd, and Gunnar Wetlesen - and by Doug Fairbairn of Xerox PARC and Lambda (later VLSI Design) magazine. Alfred J. Stein became the CEO of the company in 1982. Subsequently VLSI built its first fab in San Jose; eventually a second fab was built in San Antonio, Texas. VLSI had its initial public offering in 1983, and was listed on the stock market as (NASDAQ: VLSI). The company was later acquired by Philips and survives to this day as part of NXP Semiconductors.

The first semiconductor chips held two transistors each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held Only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device.

Now, known retrospectively as small-scale integration (SSI), improvements in technique led to devices with hundreds of logic gates, known as medium-scale integration (MSI). Further improvements led to large-scale integration (LSI), i.e. systems with at least a thousand logic gates.

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like ultra-large-scale integration (ULSI) were used. But the huge number of gates and transistors available on common. Current designs, as opposed to the earliest devices, use extensive design automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM (Static Random Access Memory) cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability) [citation needed]. VLSI technology is moving towards radical level miniaturization with introduction of NEMS technology. A lot of problems need to be sorted out before the transition is actually made.

## 2.3.2. WHY VLSI?

Integration improves the design, lowers the parasitics, which means higher speed and lower power consumption and physically smaller. The Integration reduces manufacturing cost - (almost) no manual assembly.

The course will cover basic theory and techniques of digital VLSI design in CMOS technology. Topics include: CMOS devices and circuits, fabrication processes, static and dynamic logic structures, chip layout, simulation and testing, low power techniques, design tools and methodologies.

There is an emphasis on modern design issues in interconnect and clocking. We will also use several case-studies to explore recent real-world VLSI designs (e.g. Pentium, Alpha, PowerPC Strong ARM, etc.) and papers from the recent research literature. On-campus students will design small test circuits using various CAD tools. Circuits will be verified and analyzed for performance with various simulators. Some final project designs will be fabricated and returned to students the following semester for testing.

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. The term is no longer as common as it once was, as chips have increased in complexity into the hundreds of millions of transistors. The first semiconductor chips held one transistor each.

Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Further improvements led to large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and billions of individual transistors.

Now known retrospectively as "small-scale integration" (SSI), improvements in technique led to devices with hundreds of logic gates, known as large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and hundreds of millions of individual transistors.

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like Ultra-large-scale Integration (ULSI) were used. But the huge number of gates and transistors available on common devices has rendered such fine distinctions moot. Terms suggesting greater than VLSI levels of integration are no longer in widespread use. Even VLSI is now somewhat quaint, given the common assumption that all microprocessors are VLSI or better.

This microprocessor is unique in the fact that its 1.4 Billion transistor count, capable of a teraflop of performance, is almost entirely dedicated to logic (Itanium's transistor count is largely due to the 24MB L3 cache). Current designs, as opposed to the earliest devices, use extensive design automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability).

The original business plan was to be a contract wafer fabrication company, but the venture investors wanted the company to develop IC (Integrated Circuit) design tools to help fill the foundry. Thanks to its Caltech and UC Berkeley students, VLSI was an important pioneer in the electronic design automation industry. It offered a sophisticated package of tools, originally based on the 'lambda-based' design style advocated by Carver Mead and Lynn Conway.

VLSI became an early vendor of standard cell (cell-based technology) to the merchant market in the early 80s where the other ASIC-focused company, LSI Logic, was a leader in gate arrays. Prior to VLSI's cell-based offering, the technology had been primarily available only within large vertically integrated companies with semiconductor units such as AT&T and IBM. VLSI's design tools eventually included not only design entry and simulation but eventually cell-based routing (chip compiler), a data path compiler, SRAM and ROM compilers and a state machine compiler. The tools were an integrated design solution for IC design and not just point tools, or more general purpose system tools.

A designer could edit transistor-level polygons and/or logic schematics, then run DRC and LVS, extract parasites from the layout and run Spice simulation, then back-annotate the timing or gate size changes into the logic schematic database. Characterization tools were integrated to generate Frame Maker Data Sheets for Libraries. VLSI eventually spun off the CAD and Library operation into Compass Design Automation but it never reached IPO before it was purchased by Avanti Corp.

VLSI's physical design tools were critical not only to its ASIC business, but also in setting the bar for the commercial EDA industry. When VLSI and its main ASIC competitor, LSI Logic, were establishing the ASIC industry, commercially-available tools could not deliver the productivity necessary to support the physical design of hundreds of ASIC designs each year without the deployment of a substantial number of layout engineers. The companies' development of automated layout tools was a rational "make because there's nothing to buy" decision. The EDA industry finally caught up in the late 1980s when Tangent Systems released its Tan Cell and Tan Gate products. In 1989, Tangent was acquired by Cadence Design Systems (founded in 1988).

Unfortunately, for all VLSI's initial competence in design tools, they were not leaders in semiconductor manufacturing technology. VLSI had not been timely in developing a 1.0 µm manufacturing process as the rest of the industry moved to that geometry in the late 80s. VLSI entered a long-term technology partnership with Hitachi and finally released a 1.0 µm process and cell library (actually more of a 1.2 µm library with a 1.0 µm gate).As VLSI struggled to gain parity with the rest of the industry in semiconductor technology, the design flow was moving rapidly to a Verilog HDL and synthesis flow. Cadence acquired Gateway, the leader in Verilog hardware design language (HDL) and Synopsys was dominating the exploding field of design synthesis.

Meanwhile, VLSI entered the merchant high speed static RAM (SRAM) market as they needed a product to drive the semiconductor process technology development. All the large semiconductor companies built high speed SRAMs with cost structures VLSI could never match. ARM Ltd was formed in 1990 as a semiconductor intellectual property licensor, backed by Acorn, Apple and VLSI. VLSI became a licensee of the powerful ARM processor and ARM finally funded processor tools. Initial adoption of the ARM processor was slow. Few applications could justify the overhead of an embedded 32 bit processor. Only in PC chipsets, did VLSI dominate in the early 90s. This product was developed by five engineers using the 'Mega cells" in the VLSI library that led to a business unit at VLSI that almost equaled its ASIC business in revenue. VLSI eventually ceded the market to Intel because Intel was able to package-sell its processors, chipsets, and even board level products together.

VLSI also had an early partnership with PMC, a design group that had been nurtured of British Columbia Bell. When PMC wanted to divest its semiconductor intellectual property venture, VLSI's bid was beaten by a creative deal by Sierra Semiconductor. The telecom business unit management at VLSI opted to go it alone. PMC Sierra became one of the most important telecom ASSP vendors. Scientists and innovations from the 'design technology' part of VLSI found their way to Cadence Design Systems (by way of Redwood Design Automation). Compass Design Automation (VLSI's CAD and Library spin-off) was sold to Avant! Corporation, which itself was acquired by Synopsys.

### 2.3.3. Structured design

Structured VLSI design is a modular methodology originated by Carver Mead and Lynn Conway for saving microchip area by minimizing the interconnect fabrics area.

This is obtained by repetitive arrangement of rectangular macro blocks which can be interconnected using wiring by abutment. An example is partitioning the layout of an adder into a row of equal bit slices cells. In complex designs this structuring may be achieved by hierarchical nesting.

Structured VLSI design had been popular in the early 1980s, but lost its popularity later because of the advent of placement and routing tools wasting a lot of area by routing, which is tolerated because of the progress of Moore's Law. When introducing the hardware description language KARL in the mid' 1970s, Reiner Hartenstein coined the term "structured VLSI design" echoing Edsger Dijkstra's structured programming approach by procedure nesting to avoid chaotic spaghetti-structured program.

Only in PC chipsets, did VLSI dominate in the early 90s. This product was developed by five engineers using the 'Mega cells" in the VLSI library that led to a business unit at VLSI that almost equaled its ASIC business in revenue. semiconductor technology, the design flow was moving rapidly to a Verilog HDL and synthesis flow. Cadence acquired Gateway, the leader in Verilog hardware design language (HDL) and Synopsys was dominating the exploding field of design synthesis.

## 2.3.4. APPLICATIONS OF VLSI

- Electronic system in cars.
- Digital electronics control VCRs
- Transaction processing system, ATM
- Personal computers and Workstations

Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to service. In other cases electronic systems have created totally new applications. Electronic systems perform a variety of tasks; some of them are visible while some are hidden.

Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy. Electronic systems in cars operate stereo systems and displays; they also control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking systems.

Digital electronics compress and decompress video, even at high-definition data rates, on-the-fly in consumer electronics. Low-cost terminals for Web browsing still require sophisticated electronics, despite their dedicated function. Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units and special-purpose hardware for disk access, faster screen display, etc.

Medical electronic systems measure bodily functions and perform complex processing algorithms to warn about unusual conditions. The availability of these complex systems, far from overwhelming consumers. The growing sophistication of applications continually pushes the design and manufacturing of integrated circuits and electronic systems to new levels of complexity. And perhaps the most amazing characteristic of this collection of systems is its variety-as systems become more complex, we build not a few general-purpose computers but an eve  wider range of special-purpose systems.

Our ability to do so is a testament to our growing mastery of both integrated circuit manufacturing and design, but the increasing demands of customers continue to test the limits of design and manufacturing. Electronic systems in cars operate stereo systems and displays; they also control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking systems.

### 2.3.5. ASIC

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a cell phone is an ASIC. Intermediate between ASICs and industry standard integrated circuits, like the 7400 or the 4000 series, are application specific standard products (ASSPs).

As feature sizes have shrunk and design tools improved over the years, the maximum complexity (and hence functionality) possible in an ASIC has grown from 5,000 gates to over 100 million. Modern ASICs often include entire 32-bit processors, memory blocks including ROM, RAM, EEPROM, Flash and other large building blocks. Such an ASIC is often termed a SoC (system-on-a-chip). Designers of digital ASICs use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs. Field-programmable gate arrays (FPGA).

The modern-day technology for building a breadboard or prototype from standard parts; programmable logic blocks and programmable interconnects allow the same FPGA to be used in many different applications. For smaller designs and/or lower production volumes, FPGAs may be more cost effective than an ASIC design even in production.

## 2.3.6. ASIC DESIGN FLOW

As with any other technical activity, development of an ASIC starts with an idea and takes tangible shape through the stages of development. The first step in this process is to expand the idea in terms of behaviour of the target circuit. The growing sophistication of applications continually pushes the design and manufacturing of integrated circuits and electronic systems to new levels of complexity.

Design description is an activity independent of the target technology or manufacturer. It results in a description of the digital circuit. To translate it into a tangible circuit, one goes through the physical design process. The same constitutes a set of activities closely linked to the manufacturer and the target technology.

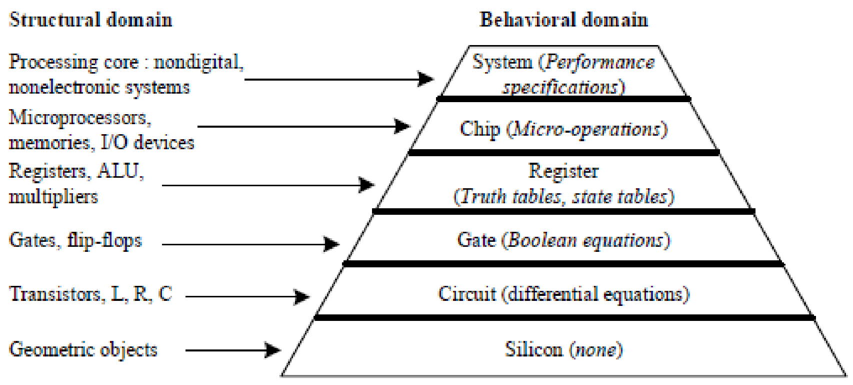

Structural domain
Behavioral domain
Processing core : nondigital, nonelectronic systems
System (Performance specifications)
Microprocessors, memories, I/O devices
Chip (Micro-operations)
Registers, ALU, multipliers
Register (Truth tables, state tables)
Gates, flip-flops
Gate (Boolean equations)
Transistors, L, R, C
Circuit (differential equations)
Geometric objects
Silicon (none)

**FIG 2.3: ASIC DESIGN**

The design is tested through a simulation process; it is to check, verify, and ensure that what is wanted is what is described. Simulation is carried out through dedicated tools. With every simulation run, the simulation results are studied to identify errors in the design description. The errors are corrected and another simulation run carried out.

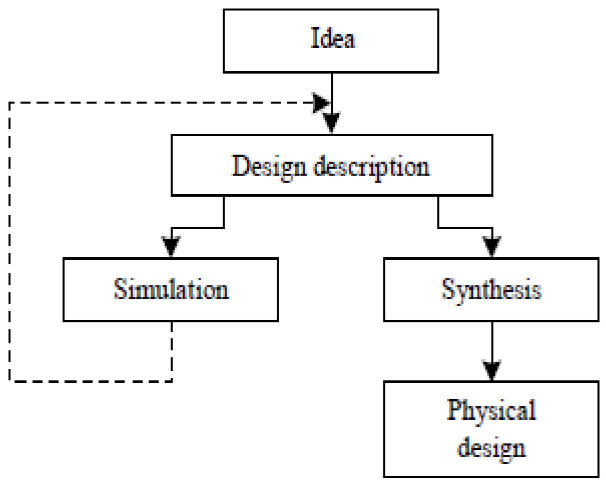


**FIG 2.4: ASIC DESIGN FLOW**

# CHAPTER 3
# SOFTWARE REQUIREMENTS

## 3.1 XILINX ISE

Xilinx, Inc. is the world's largest provider of programmable common sense devices, the inventor of the field programmable gate array (FPGA) and the primary semiconductor organization with a fabless manufacturing version. Xilinx designs, develops and markets programmable logic merchandise including incorporated circuits (ICs), software program design equipment and predefined gadget functions added as intellectual property (IP) cores, design offerings, patron education, area engineering and technical assist. Xilinx sells each FPGAs and CPLDs programmable common sense devices for electronic equipment producers in cease markets along with communications, commercial, customer, automobile and statistics processing.

Xilinx's FPGAs have even been used for the ALICE (A huge Ion Collider test) on the CERN ecu laboratory at the French-Swiss border to map and disentangle the trajectories of heaps of subatomic debris. The Vertex-II seasoned, Virtex-6, Virtex-five, and Virtex-6 FPGA families are mainly focused on gadget-on-chip (SOC) designers due to the fact they consist of up to two embedded IBM PowerPC cores. The ISE layout Suite is the critical digital design automation (EDA) product own family sold by using Xilinx.

The ISE design Suite features include design access and synthesis assisting Verilog or VHDL, place-and-route (PAR), completed verification and debug using Chip Scope pro equipment, and advent of the bit documents which can be used to configure the chip. XST-Xilinx Synthesis era performs device particular synthesis for Cool Runner XPLA3/-II and XC9600/XL/XV households and generates an NGC report ready for the CPLD more fit.

## 3.2 XILINX ISE 13.2i:

Xilinx is the maximum important tool and in this device we are able to carry out both simulation and synthesis.

### 3.2.1 Simulation:

In this process, we are going to verify our required output to get the simulation technique first of all we need to enforce a top module (combination of all modules) after which in the simulation conduct we can simulate the result.

### 3.2.2 Synthesis:

Synthesis process defines converting Verilog code into gate level which creates a net list.

### 3.2.3 Procedure:

- Click project navigator
- Create new project
- Selection of FPGA

Create new source

- Select source type (Verilog module)
- Coding
- Declaration of inputs and output
- Sources for implementation

Synthesize – XST

- Check syntax
- View design summary
- View RTL schematic
- View technology schematic
- Sources for behavioural simulation

Create new source

- Select source type (Verilog text fixture)
- Write test bench code
- Xilinx ISE simulator
- Behavioural check syntax
- Simulate behavioural model

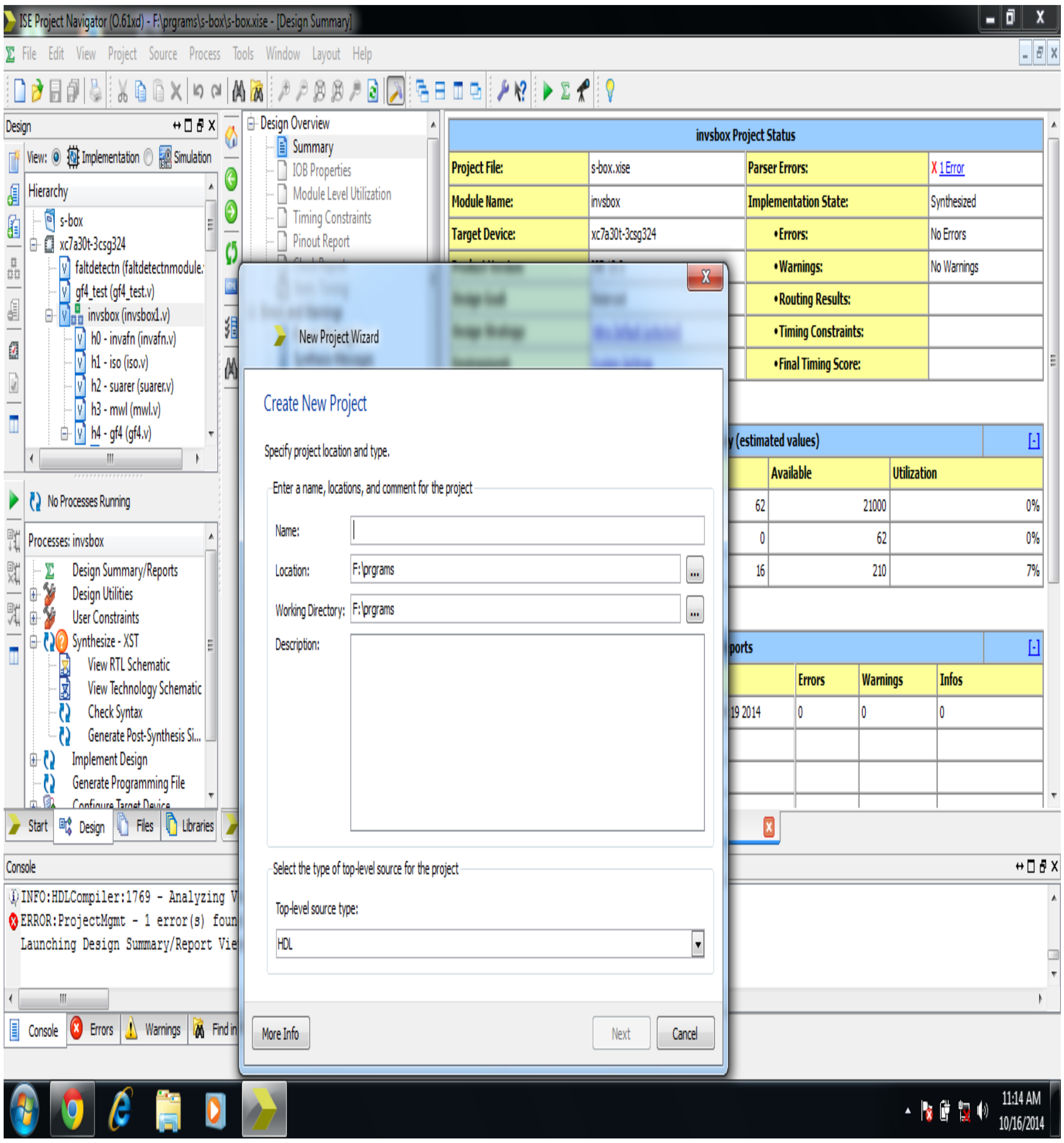
## 3.3 PROCEDURE FOR SYNTHESIS:

Xilinx designs, develops and markets programmable logic merchandise including incorporated circuits (ICs), software program design equipment and predefined gadget functions added as intellectual property (IP) cores, design offerings, patron education, area engineering and technical assist. Xilinx sells each FPGAs and CPLDs programmable common sense devices for electronic equipment producers in cease markets along with communications, commercial, customer, automobile and statistics processing.

It is a comprehensive software suite that offers a modern platform for the design, synthesis, and verification of FPGA-based systems.

## PROCEDURE FOR XILINX

1.To create new project in xilinx we should open the filemenu,click on new project then it will open the dialogbox as below in that typethe filename click on next



Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name:

Location: F:\prgrams

Working Directory: F:\prgrams

Description:

Select the type of top-level source for the project

Top-level source type: HDL

| invsbox Project Status | | | |
|---|---|---|---|
| Project File: | s-box.xise | Parser Errors: | X 1 Error |
| Module Name: | invsbox | Implementation State: | Synthesized |
| Target Device: | xc7a30t-3csg324 | •Errors: | No Errors |
| | | •Warnings: | No Warnings |
| | | •Routing Results: | |
| | | •Timing Constraints: | |
| | | •Final Timing Score: | |

y (estimated values)

| | Available | Utilization |
|---|---|---|
| 62 | 21000 | 0% |
| 0 | 62 | 0% |
| 16 | 210 | 7% |

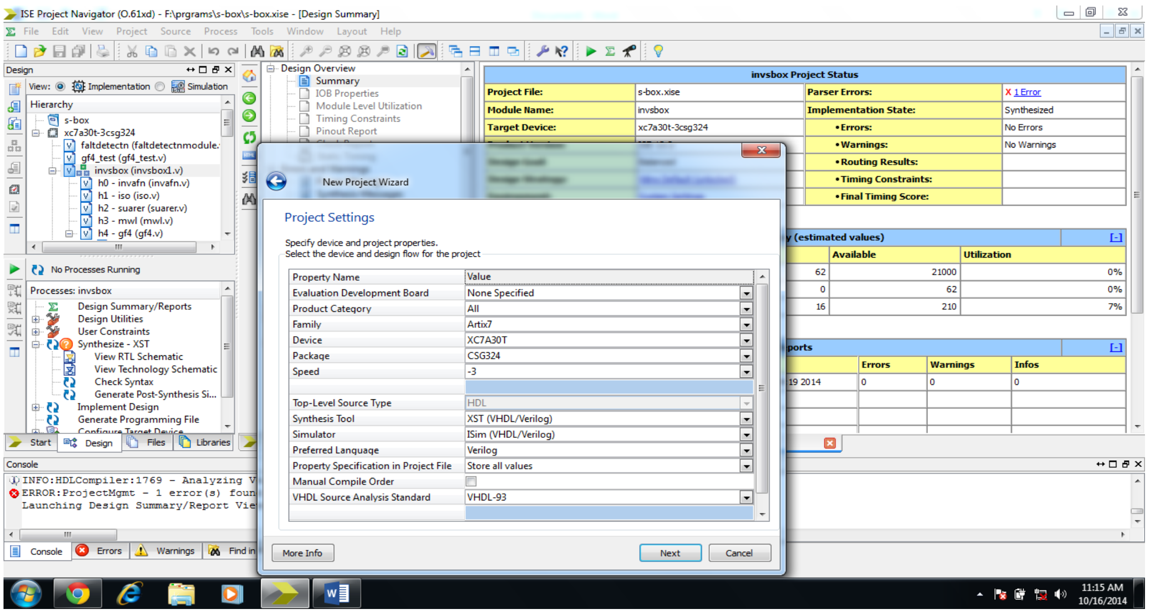
ports

| | Errors | Warnings | Infos |
|---|---|---|---|
| 19 2014 | 0 | 0 | 0 |

```
INFO:HDLCompiler:1769 - Analyzing V
ERROR:ProjectMgmt - 1 error(s) foun
Launching Design Summary/Report Vie
```

FIG 3.1: CREATING NEW FILE IN XILINX

2.Then it isplays one more dialogbox which will give us the specifications of the project,click on next



ISE Project Navigator (O.61xd) - F:\prgrams\s-box\s-box.xise - [Design Summary]

New Project Wizard

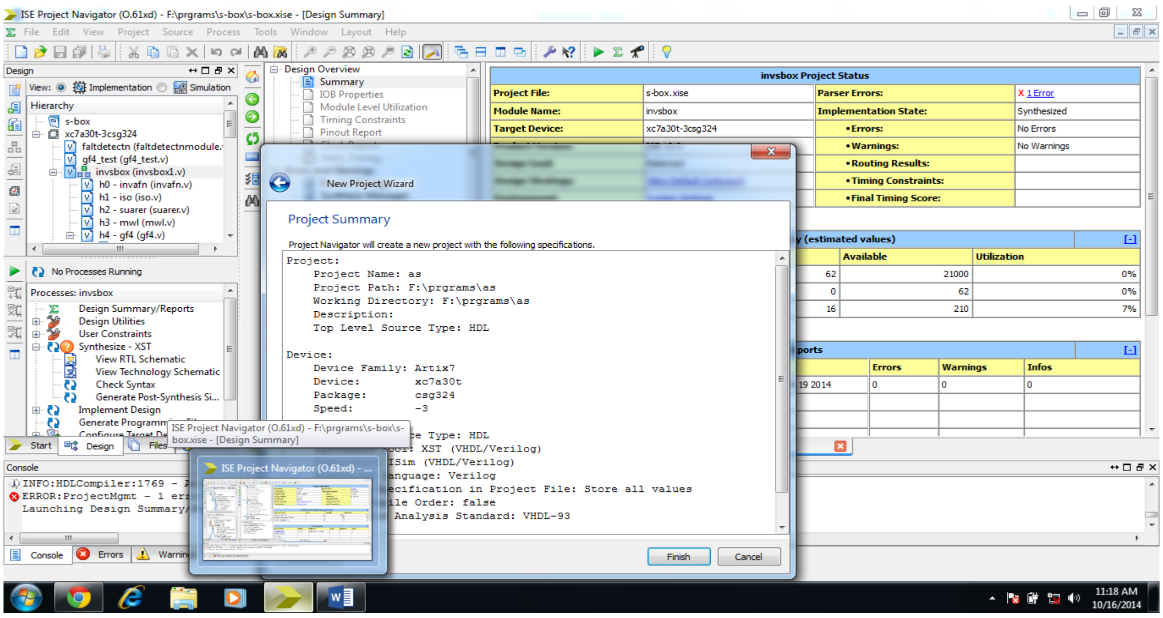
Project Settings

Specify device and project properties.
Select the device and design flow for the project

| Property Name | Value |
|---|---|
| Evaluation Development Board | None Specified |
| Product Category | All |
| Family | Artix7 |
| Device | XC7A30T |
| Package | CSG324 |
| Speed | -3 |
| | |
| Top-Level Source Type | HDL |
| Synthesis Tool | XST (VHDL/Verilog) |
| Simulator | ISim (VHDL/Verilog) |
| Preferred Language | Verilog |
| Property Specification in Project File | Store all values |
| Manual Compile Order | ☐ |
| VHDL Source Analysis Standard | VHDL-93 |

FIG 3.2: DISPLAYS ONE MORE DAILOG BOX FOR SPECIFICATIONS IN XILINX

3.Then it again displays a dialogue box as shown below with the created project description and click finish to compelte the process of creating new project



ISE Project Navigator (O.61xd) - F:\prgrams\s-box\s-box.xise - [Design Summary]

New Project Wizard

Project Summary

Project Navigator will create a new project with the following specifications.

```
Project:
    Project Name: as
    Project Path: F:\prgrams\as
    Working Directory: F:\prgrams\as
    Description:
    Top Level Source Type: HDL

Device:
    Device Family: Artix7
    Device:        xc7a30t
    Package:       csg324
    Speed:         -3

             ce Type: HDL
          ool: XST (VHDL/Verilog)
          ISim (VHDL/Verilog)
          anguage: Verilog
          ecification in Project File: Store all values
          ile Order: false
           Analysis Standard: VHDL-93
```

FIG 3.3: DISPLAYS AGAIN A DAILOG BOX AS SHOWN BELOW WITH THE CREATED PROJECT DESCRIPTION IN XILINX

4.Now project with specifyed name is created then create the verilog files in the project. To create filesr, right click on the project that will show options like as shown below

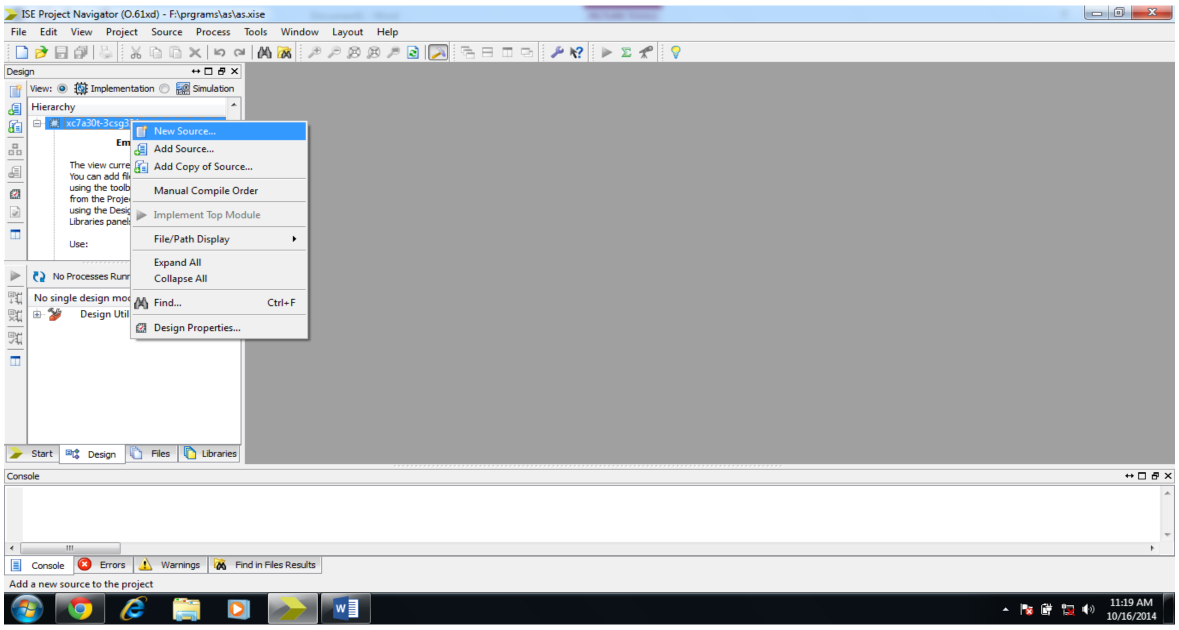
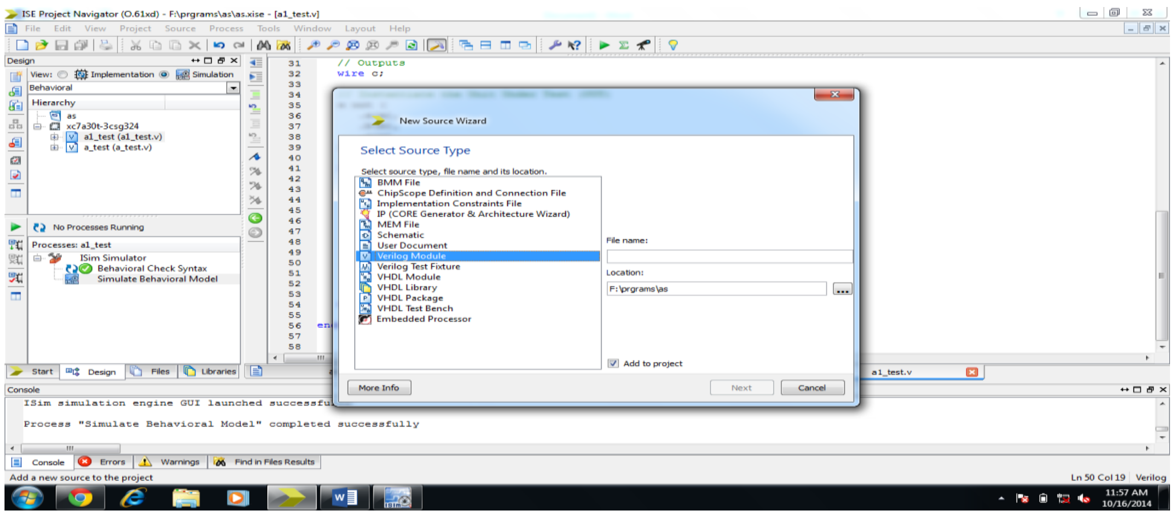


FIG 3.4: SPECIFIED NAME IS CREATED THEN CREATE THE VERILOG FILES IN THE PROJECT

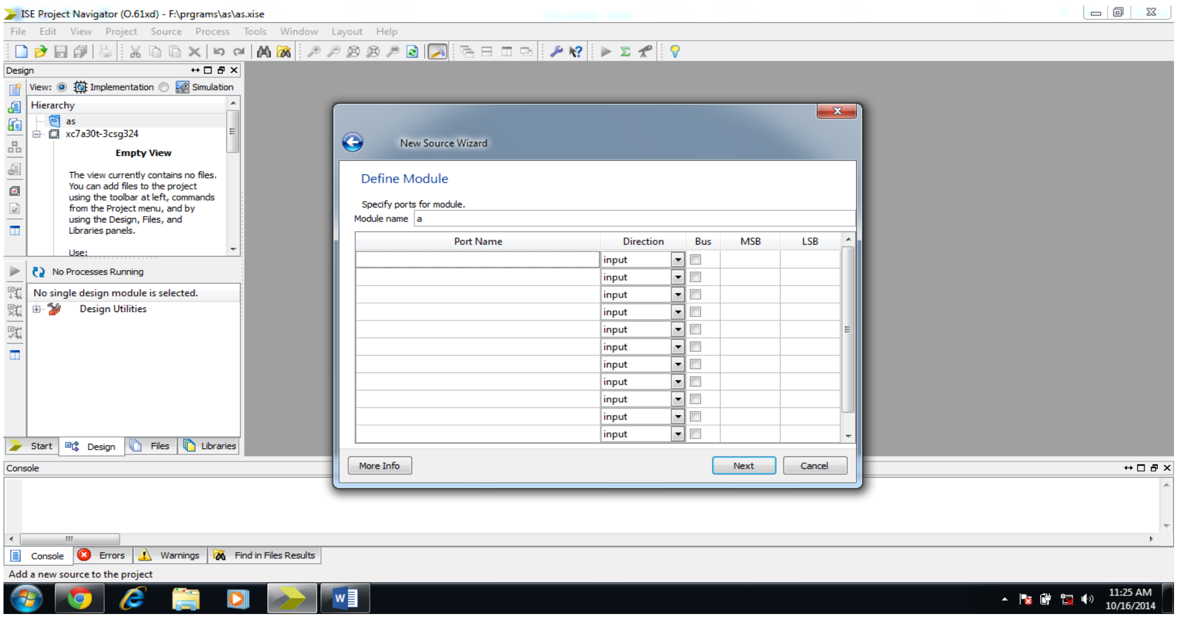
5.From the given options select new source then it diaplays dialogbox which is containing of list of fileformat now we want to create verlogfile so select veilog module,and give the name to the file. Then click on next


```
// Outputs
wire c;
```

```
ISim simulation engine GUI launched successfu
Process "Simulate Behavioral Model" completed successfully
```

FIG 3.5: SELECT NEW SOURCE THE IT DISPLAYS DAILOGBOX WHICH IS CONTAINING OF LIST OF FILE FORMAT IN XILINX

6.Then it will ask us to select inputs,outputs and inouts. We can specify our inputs and outputs here else we may also specify as part of programme depend upon the user requirement, click on next


ISE Project Navigator (O.61xd) - F:\prgrams\as\as.xise

New Source Wizard

Define Module

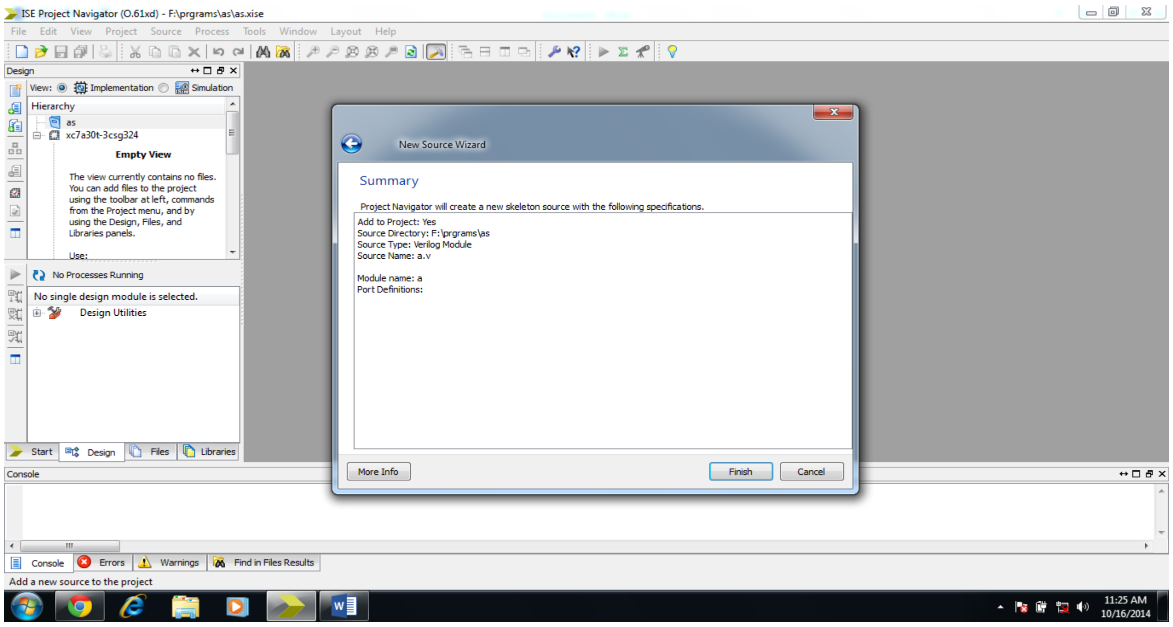
Specify ports for module.

Module name: a

| Port Name | Direction | Bus | MSB | LSB |
|---|---|---|---|---|
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |
| | input | ☐ | | |

FIG 3.6: SELECT INPUTS & OUTPUTS AND INOUTS IN XILINX

7.It will again displays a dilagbox by fiving details of filename etc, click on next


ISE Project Navigator (O.61xd) - F:\prgrams\as\as.xise

New Source Wizard

Summary

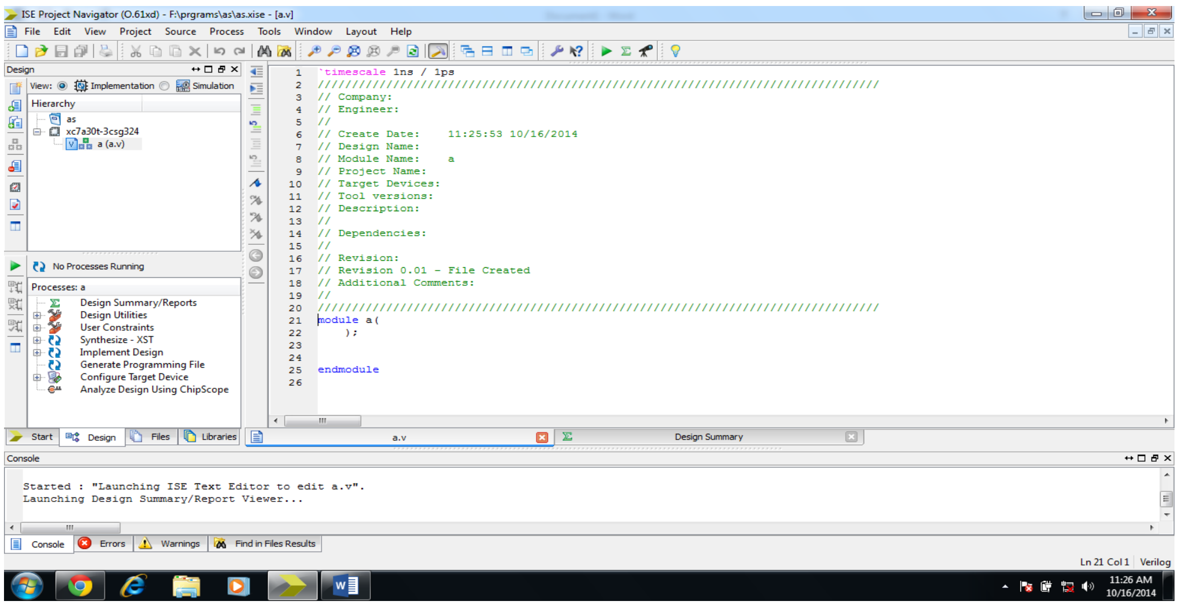
Project Navigator will create a new skeleton source with the following specifications.

Add to Project: Yes
Source Directory: F:\prgrams\as
Source Type: Verilog Module
Source Name: a.v

Module name: a
Port Definitions:

FIG 3.7: DISPLAYS AGAIN A DAILOGBOX BY FIVING DETAILS OF FILENAME IN XILINX

8.It will open a white space in the project window containing filename the double click on the file name so that it will displays respective file windows, where we should write the code

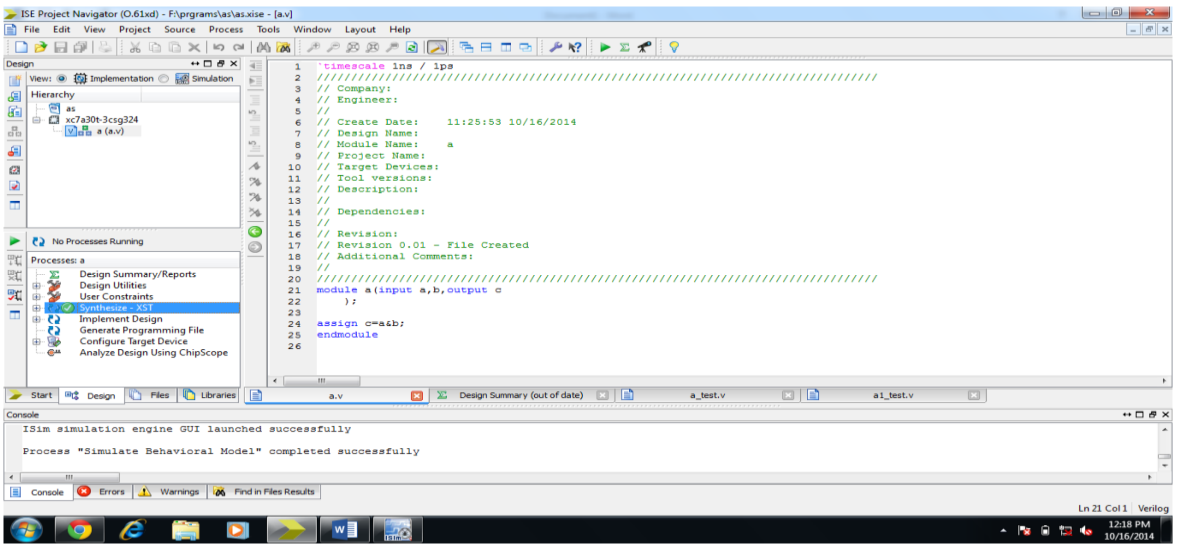


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    11:25:53 10/16/2014
// Design Name:
// Module Name:    a
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module a(
    );


endmodule
```

```
Started : "Launching ISE Text Editor to edit a.v".
Launching Design Summary/Report Viewer...
```

**FIG 3.8: OPEN A WHITE SPACE IN THE PROJECT WINDOW CONTAINING FILENAME IN XILINX**

9.After completion writing code select the file name and click on synthesis which will check for errors, if there are any errors in syntax or design errors are checked and shown in the below of file window

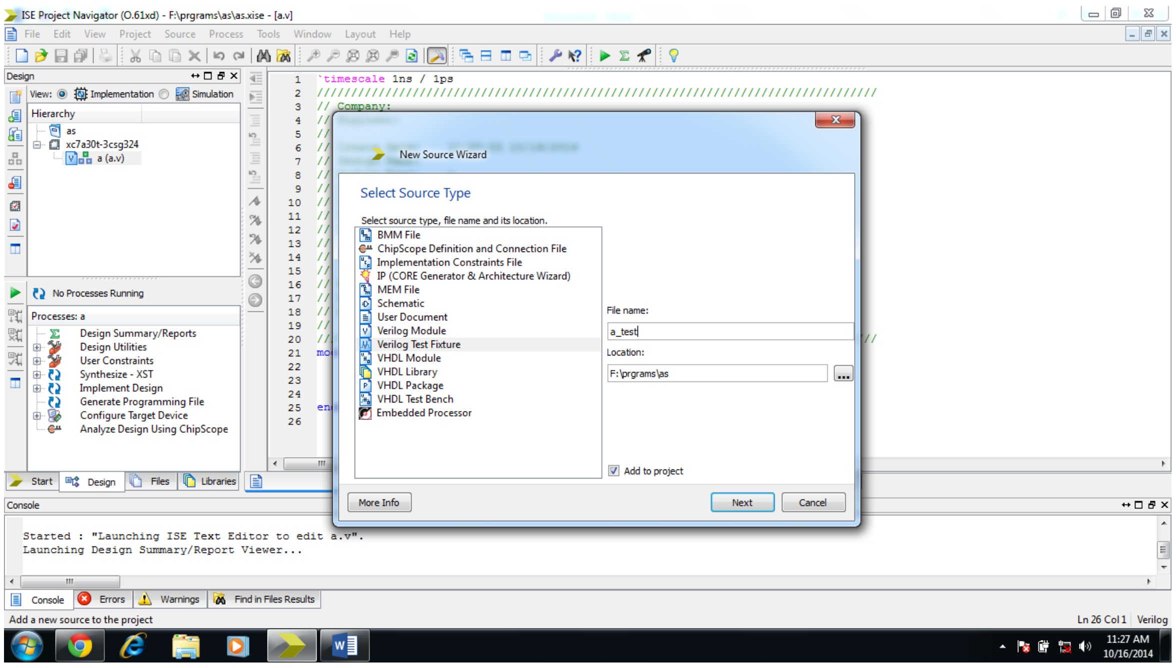


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    11:25:53 10/16/2014
// Design Name:
// Module Name:    a
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module a(input a,b,output c
    );

assign c=a&b;
endmodule
```

```
ISim simulation engine GUI launched successfully

Process "Simulate Behavioral Model" completed successfully
```

**FIG 3.9: WRITING CODE SELECT THE FILENAME IN XILINX**

10.After sucessful synthesis we should have to create tesh bench file with extension as test,for that again riht click on the file name as shown below,give filename



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
```

New Source Wizard

Select Source Type

Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: a_test

Location: F:\prgrams\as

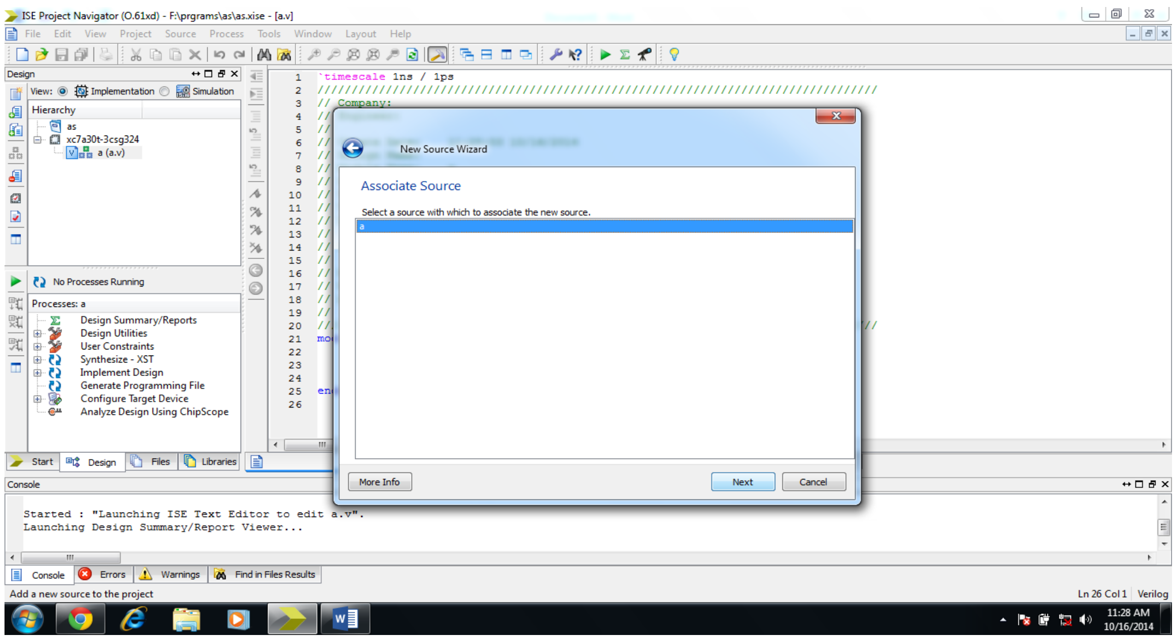
Add to project

```
Started : "Launching ISE Text Editor to edit a.v".
Launching Design Summary/Report Viewer...
```

Add a new source to the project

**FIG 3.10: AFTER SUCCESSFUL SYNTHESIS WE SHOULD HAVE TO CREATE TEST BENCH FILE IN XILINX**

11.If there are list files then select file for which we are creating the test bench. Click on next



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
```

New Source Wizard

Associate Source

Select a source with which to associate the new source.

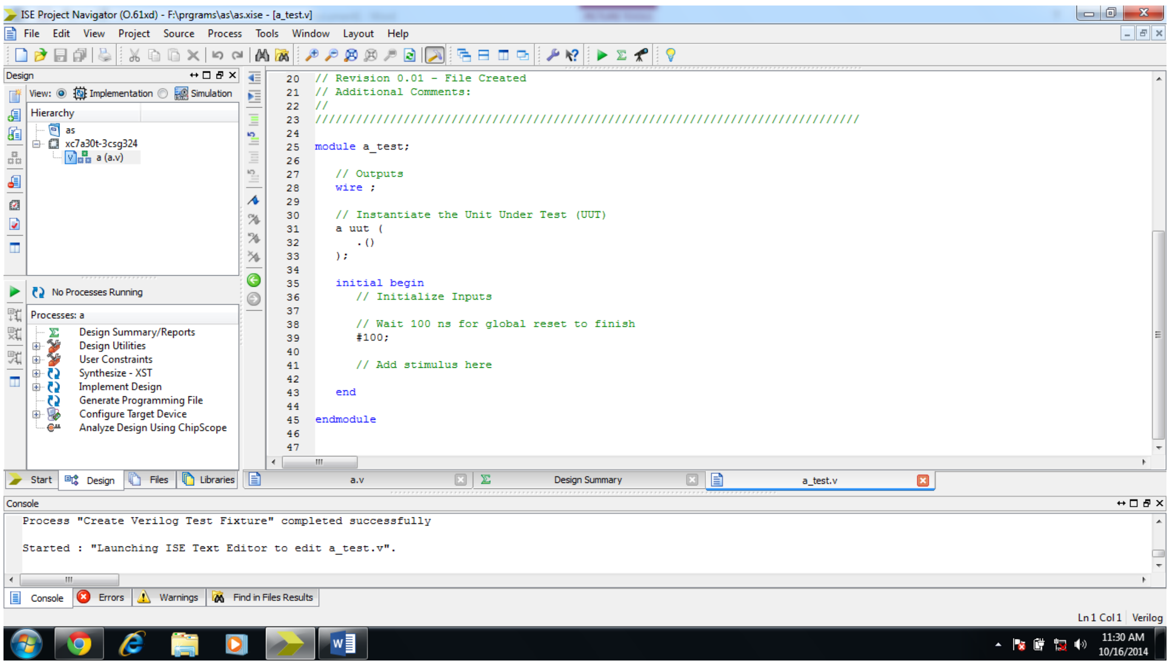
a

```
Started : "Launching ISE Text Editor to edit a.v".
Launching Design Summary/Report Viewer...
```

Add a new source to the project

**FIG 3.11: LIST FILES THEN SELECT FILE FOR WHICH WE ARE CREATING THE TEST BENCH IN XILINX**

12.It again gives a testbench file in the project window, then give reqired inputs

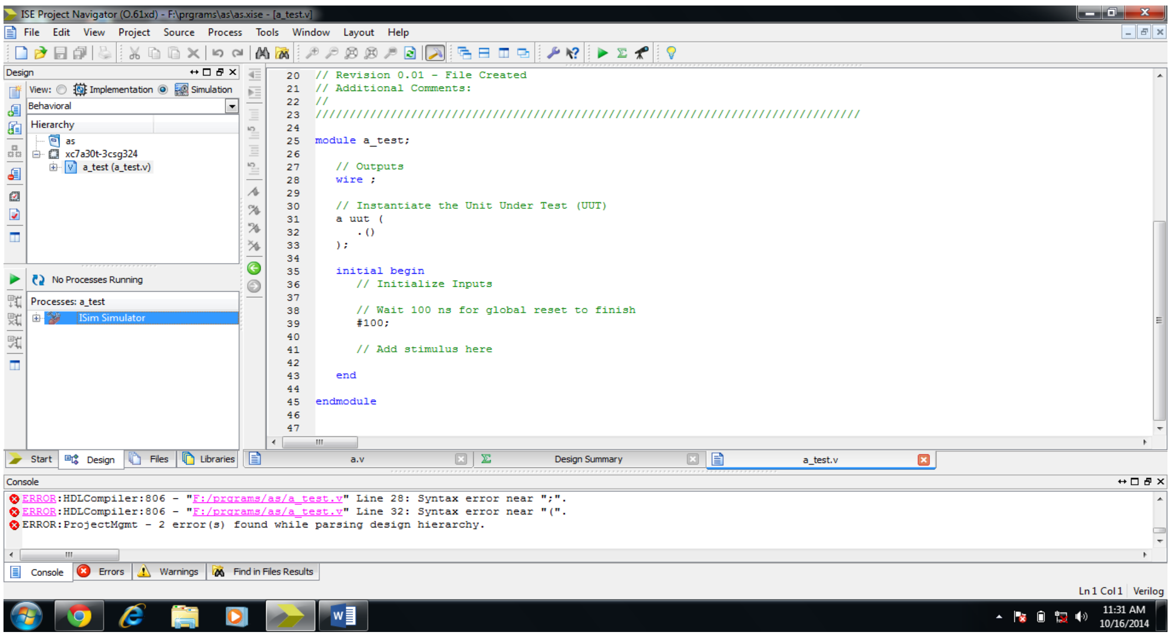

ISE Project Navigator (O.61xd) - F:\prgrams\as\as.xise - [a_test.v]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
as
xc7a30t-3csg324
a (a.v)
No Processes Running
Processes: a
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start Design Files Libraries
a.v Design Summary a_test.v
Console Errors Warnings Find in Files Results
Ln 1 Col 1 Verilog
11:30 AM 10/16/2014

```
20  // Revision 0.01 - File Created
21  // Additional Comments:
22  //
23  ////////////////////////////////////////////////////////////////////////////////
24
25  module a_test;
26
27     // Outputs
28     wire ;
29
30     // Instantiate the Unit Under Test (UUT)
31     a uut (
32        .()
33     );
34
35     initial begin
36        // Initialize Inputs
37
38        // Wait 100 ns for global reset to finish
39        #100;
40
41        // Add stimulus here
42
43     end
44
45  endmodule
46
47
```

```
Process "Create Verilog Test Fixture" completed successfully

Started : "Launching ISE Text Editor to edit a_test.v".
```

FIG 3.12: GIVE A TEST BENCH FILE IN THE PROJECT WINDOW IN XILINX

13.select simulation  from the view bar in the project window above the hiearchy window as follows.

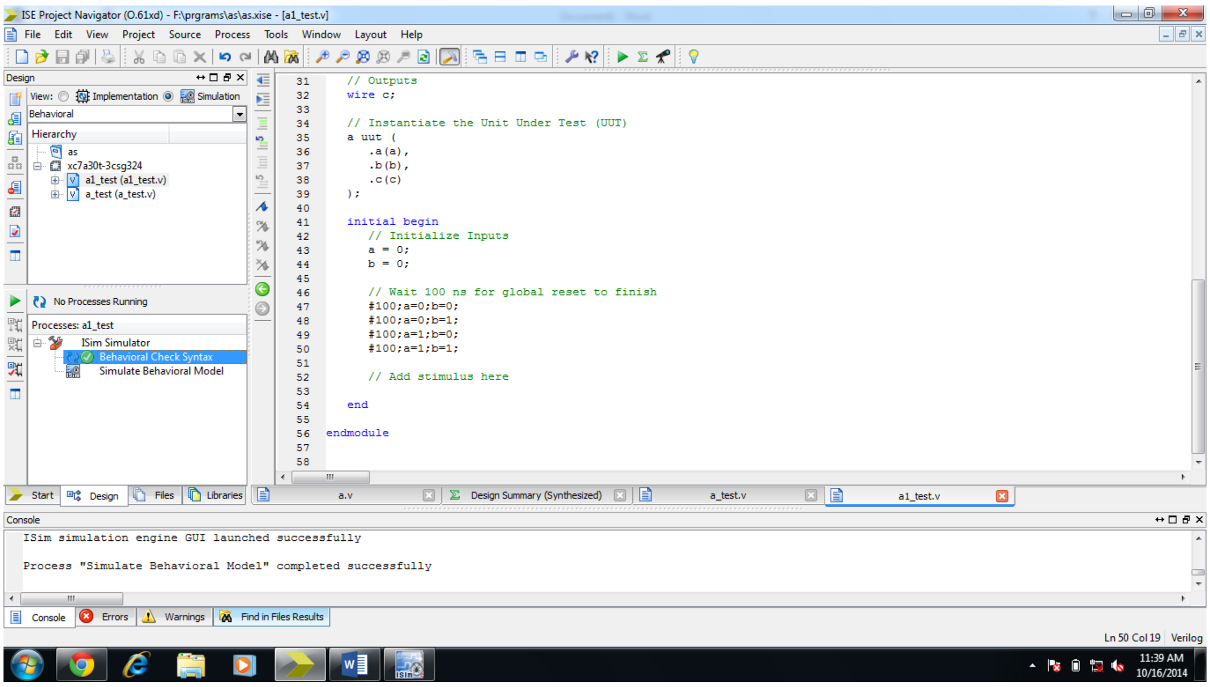

ISE Project Navigator (O.61xd) - F:\prgrams\as\as.xise - [a_test.v]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
as
xc7a30t-3csg324
a_test (a_test.v)
No Processes Running
Processes: a_test
ISim Simulator
Start Design Files Libraries
a.v Design Summary a_test.v
Console Errors Warnings Find in Files Results
Ln 1 Col 1 Verilog
11:31 AM 10/16/2014

```
20  // Revision 0.01 - File Created
21  // Additional Comments:
22  //
23  ////////////////////////////////////////////////////////////////////////////////
24
25  module a_test;
26
27     // Outputs
28     wire ;
29
30     // Instantiate the Unit Under Test (UUT)
31     a uut (
32        .()
33     );
34
35     initial begin
36        // Initialize Inputs
37
38        // Wait 100 ns for global reset to finish
39        #100;
40
41        // Add stimulus here
42
43     end
44
45  endmodule
46
47
```

```
ERROR:HDLCompiler:806 - "F:/prgrams/as/a_test.v" Line 28: Syntax error near ";".
ERROR:HDLCompiler:806 - "F:/prgrams/as/a_test.v" Line 32: Syntax error near "(".
ERROR:ProjectMgmt - 2 error(s) found while parsing design hierarchy.
```

FIG 3.13: SELECT SIMULATION FROM THE VIEW BAR IN THE PROJECT WINDOW IN XILINX

14. Double click on Isim Simulator it will expand as follows click on behavioural check syntax and it will check for syntax errors in test bench file

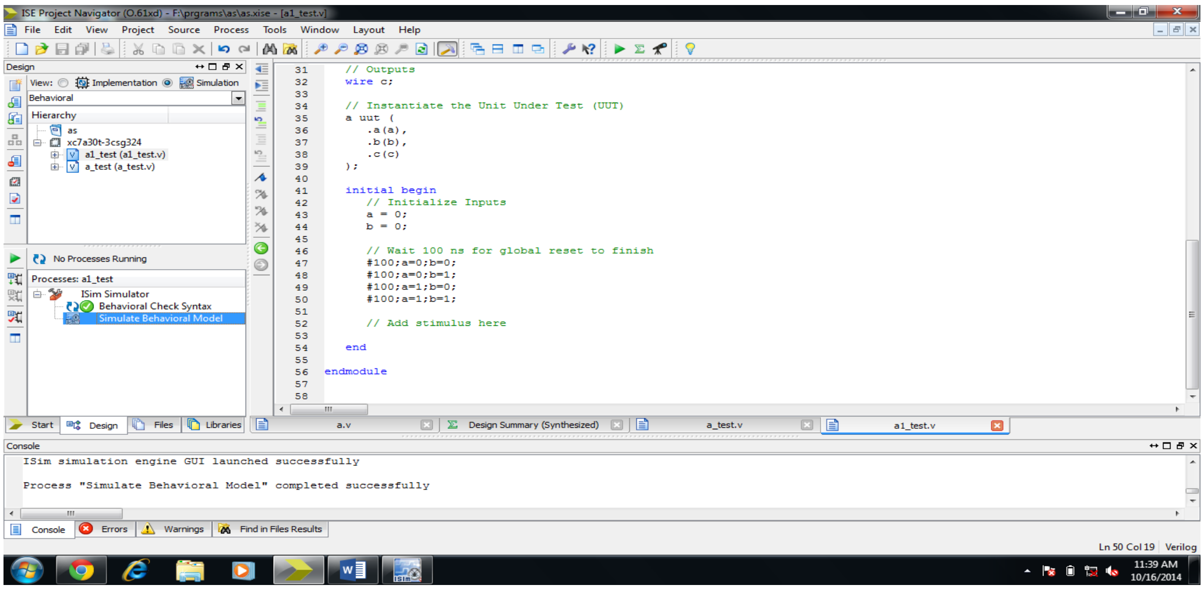

ISE Project Navigator (O.61xd) - F:\prgrams\as\as.xise - [a1_test.v]

```
    // Outputs
    wire c;

    // Instantiate the Unit Under Test (UUT)
    a uut (
        .a(a),
        .b(b),
        .c(c)
    );

    initial begin
        // Initialize Inputs
        a = 0;
        b = 0;

        // Wait 100 ns for global reset to finish
        #100;a=0;b=0;
        #100;a=0;b=1;
        #100;a=1;b=0;
        #100;a=1;b=1;

        // Add stimulus here

    end

endmodule
```

```
ISim simulation engine GUI launched successfully

Process "Simulate Behavioral Model" completed successfully
```

FIG 3.14: DOUBLE CLICK ON ISE SIMULATOR IN XILINX

15. click on simulate behavioural model, it will displays wave form for in response to the inputs given in the test bench file

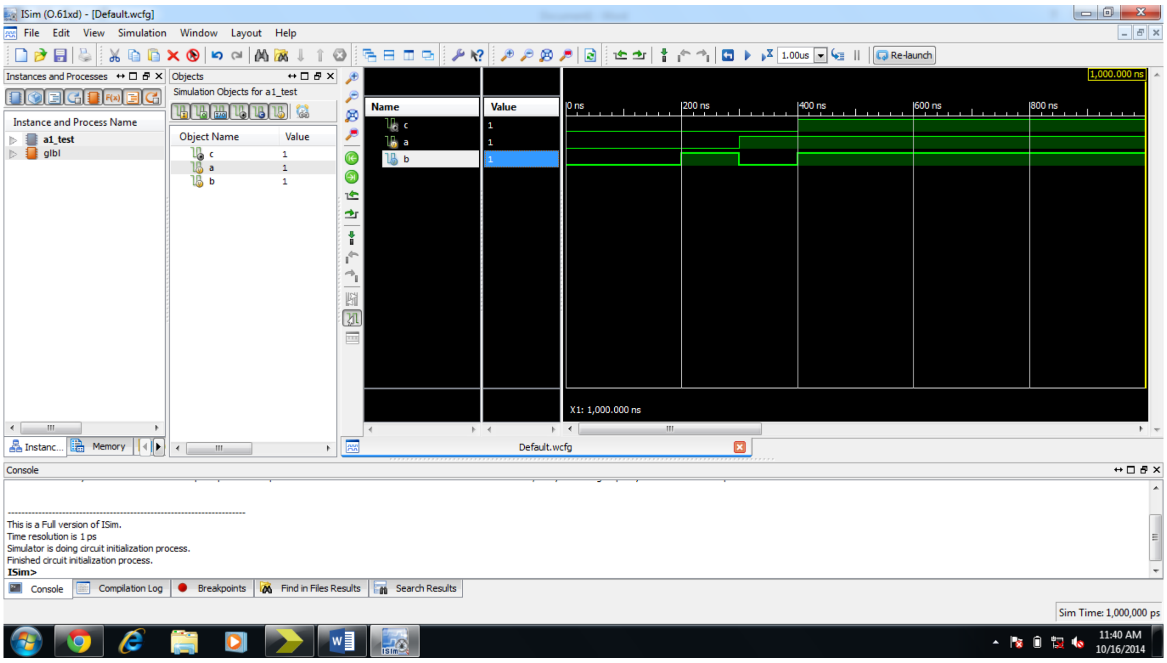

ISE Project Navigator (O.61xd) - F:\prgrams\as\as.xise - [a1_test.v]

```
    // Outputs
    wire c;

    // Instantiate the Unit Under Test (UUT)
    a uut (
        .a(a),
        .b(b),
        .c(c)
    );

    initial begin
        // Initialize Inputs
        a = 0;
        b = 0;

        // Wait 100 ns for global reset to finish
        #100;a=0;b=0;
        #100;a=0;b=1;
        #100;a=1;b=0;
        #100;a=1;b=1;

        // Add stimulus here

    end

endmodule
```

```
ISim simulation engine GUI launched successfully

Process "Simulate Behavioral Model" completed successfully
```

FIG 3.15: CLICK ON SIMULATE BEHAVIOURAL MODEL IN XILINX

16.That wave form window having option to zoom out, zoom in to analyze the wave form clearly in order to understand behaviour of design



```
-------------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

FIG 3.16: WAVEFORM WINDOW HAVING OPTION TO ZOOMOUT & ZOOM IN, IN XILINX

## VIVADO SOFTWARE

Here we have used XILINX updated version that is VIVADO SOFTWARE TOOL

Vivado is Xilinx's flagship design suite for FPGA and System-on-Chip (SoC) development. It is a comprehensive software suite that offers a modern platform for the design, synthesis, and verification of FPGA-based systems. Vivado replaces the older ISE Design Suite and provides a more advanced and integrated environment for high-performance FPGA development.

### Key Features of Vivado:

1. **High-Level Synthesis (HLS):** Vivado allows users to design systems using C, C++, and System C code, which is then automatically converted into hardware description language (HDL) code for FPGA implementation. This feature speeds up the design process and makes it easier to implement complex algorithms.
2. **IP Integrator:** Vivado includes a powerful IP catalog that provides pre-designed modules (IP cores) that can be integrated into a design. These IP cores cover a wide range of functionality, including communication protocols, memory controllers, and more.

3. **Synthesis and Implementation:** Vivado provides powerful tools for synthesizing and optimizing designs. It uses advanced place and route algorithms to ensure efficient use of FPGA resources, maximizing performance, minimizing power consumption, and reducing area.
4. **Simulation and Debugging:** Vivado includes built-in simulation tools like Vivado Simulator and support for Model  Sim, making it easier to test and debug your design. Debugging tools like Integrated Logic Analyzer (ILA) and Chip Scope help you capture internal signals and verify functionality in real-time on the FPGA hardware.
5. **Constraint Management:** Vivado allows you to define timing, placement, and resource constraints, ensuring your design meets performance goals and works within the limitations of the target FPGA.
6. **Programming and Deployment:** Vivado allows direct FPGA programming through the software, helping
7. load the final bitstream file onto the FPGA board.
8. VIVADO is made with a superior synergistic blend of natural extracts and vitamins to provide complete immunity and cardio coverage.

**How Vivado is Used in FPGA Development:**

1. **Design Creation:** You can create your digital design in Vivado using either HDL (Verilog or VHDL) or high-level synthesis tools (C, C++).
2. **Synthesis:** After writing your HDL code, Vivado synthesizes the design, turning your HDL code into a netlist that describes how the hardware will be implemented on the FPGA.
3. **Implementation:** The implementation step involves mapping the synthesized netlist onto the FPGA hardware, considering factors like placement (where elements are physically located on the FPGA) and routing (how signals are connected between elements).
4. **Simulation:** Vivado supports both pre-synthesis and post-synthesis simulation to ensure that your design functions correctly before it's deployed onto the actual FPGA.
5. **Optimization:** Vivado helps optimize the design for performance (speed), power, and area by adjusting the synthesized netlist and applying specific place and route optimizations.

6. **Bitstream Generation:** Once the design is verified, Vivado generates a bitstream file that configures the FPGA with your design. This file is then downloaded onto the FPGA for real-time operation.
7. **Debugging and Verification:** Vivado supports real-time debugging by connecting the FPGA to the Vivado Logic Analyzer or Chip Scope, enabling you to capture and analyze internal signals during execution, ensuring your design works as expected.

**How Vivado Works:**

Vivado simplifies the process of designing, synthesizing, verifying, and deploying FPGA systems. After creating the design (in Verilog, VHDL, or C/C++), Vivado compiles the code into a configuration file (bitstream), which is uploaded to the FPGA hardware. The FPGA then implements the logic and functionality specified in the design, interacting with external peripherals, sensors, or other devices. Vivado Design Suite, developed by AMD (formerly Xilinx), is a comprehensive software environment for the synthesis and analysis of hardware description language (HDL) designs. It's crucial for designing and implementing complex digital circuits on Field-Programmable Gate Arrays (FPGAs) and Adaptive SoCs.

**STEPS FOR VIVADO SOFTWARE**

**STEP1:** Interface of the vivado software

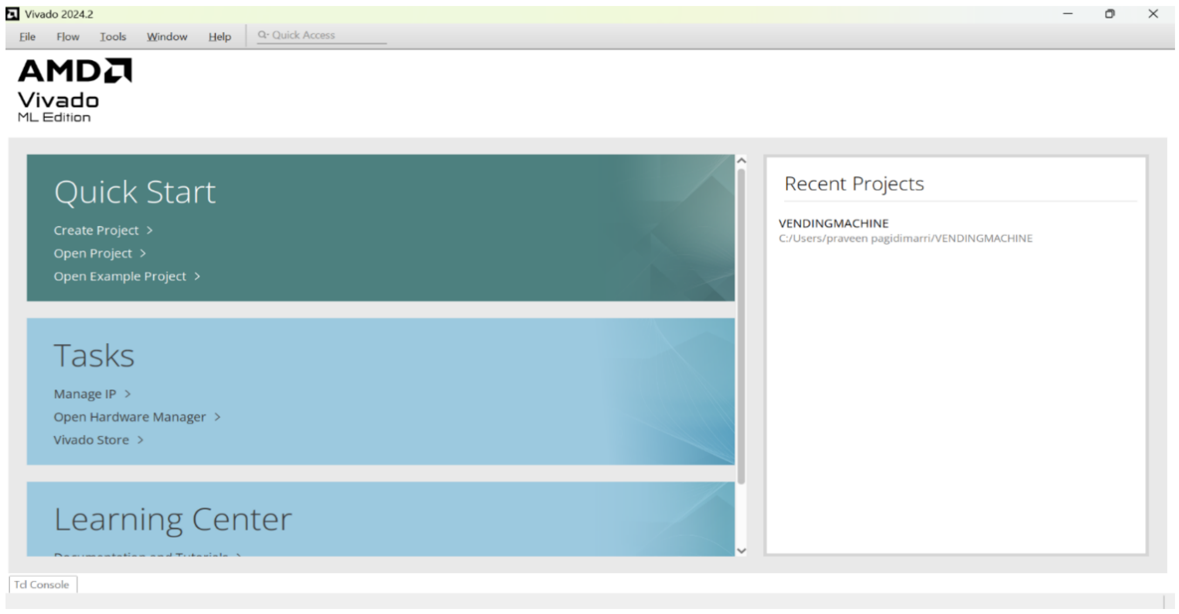


**FIG 3.17: INTERFACE OF THE VIVADO SOFTWARE**

**STEP2:** To create a project go to the file and click on project then click on new

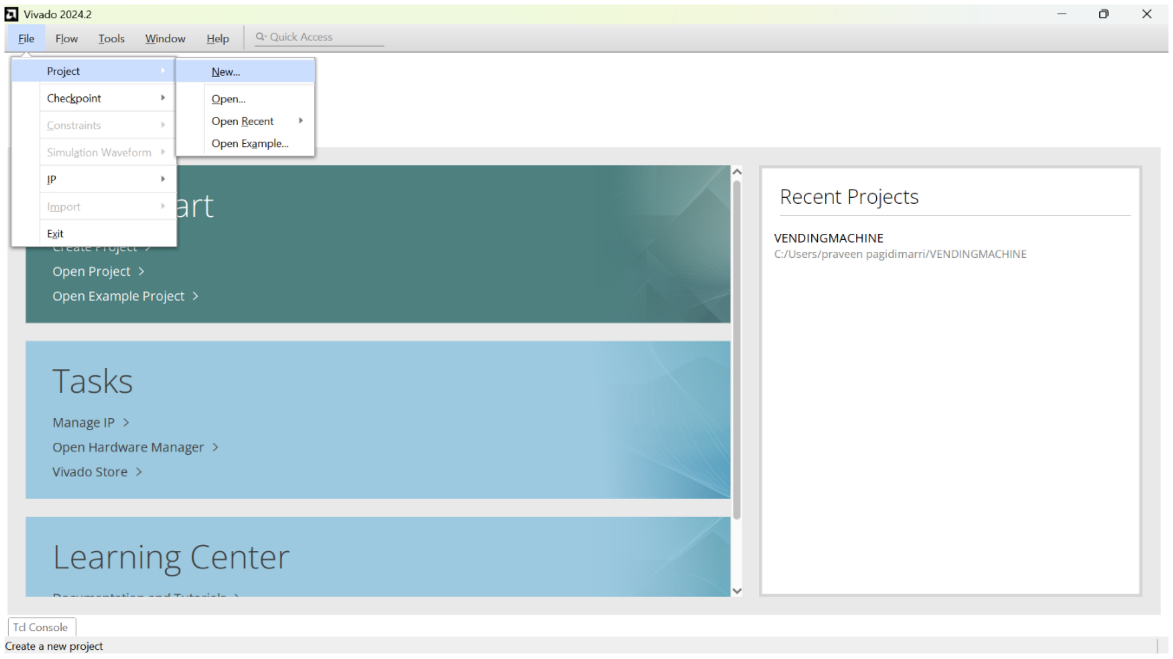
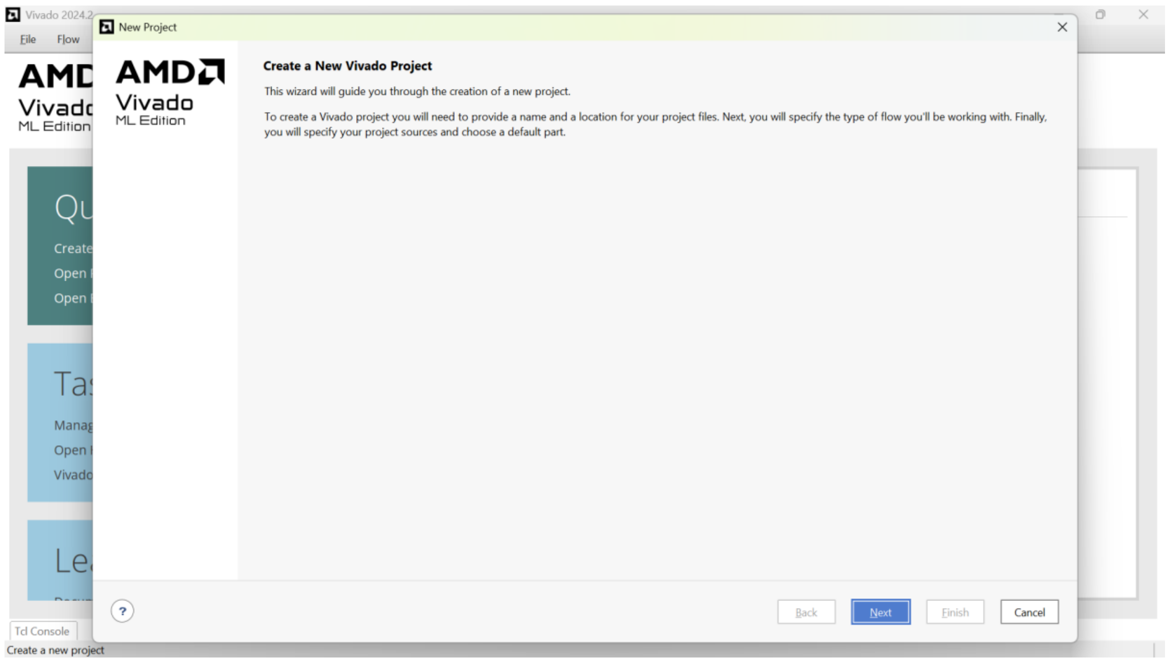


**FIG 3.18: TO CREATE A PROJECT GO TO THE FILE AND CLICK ON PROJECT**

**STEP3:** Click on next



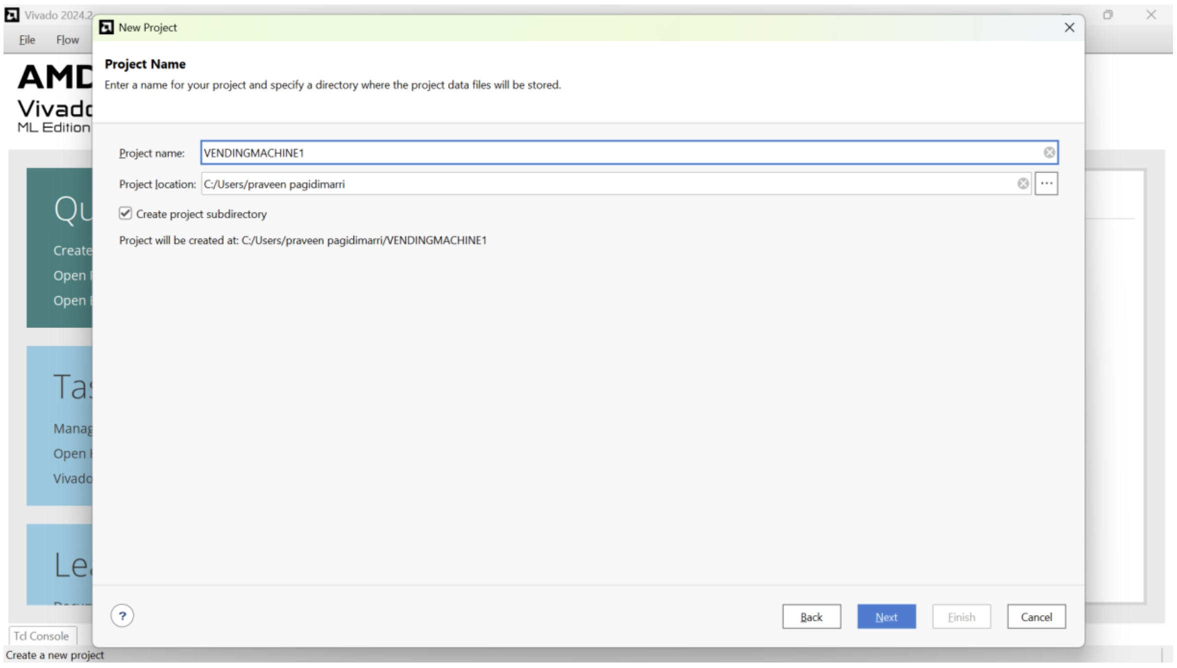
Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

**FIG 3.18: CREATE A NEW VIVADO PROJECT**

**STEP4:** Give project name without space then click next



New Project

**Project Name**

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: VENDINGMACHINE1

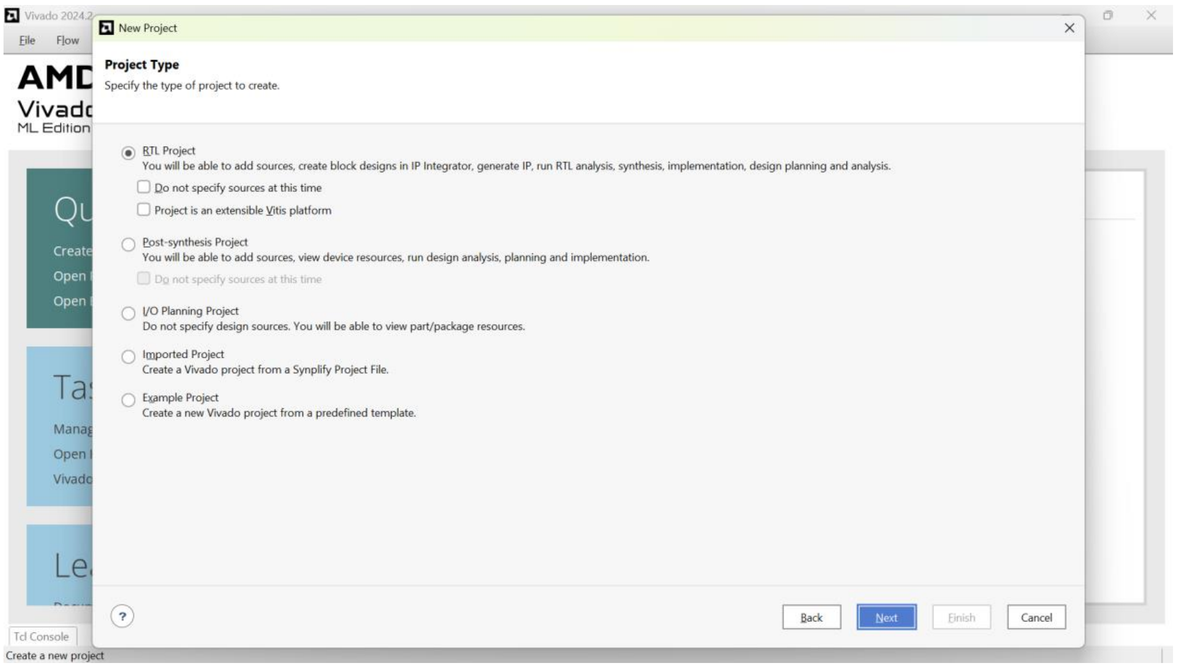
Project location: C:/Users/praveen pagidimarri

☑ Create project subdirectory

Project will be created at: C:/Users/praveen pagidimarri/VENDINGMACHINE1

FIG 3.19: CREATE PROJECT NAME

**STEP5:** Select RTL Project then click next



New Project

**Project Type**

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time
☐ Project is an extensible Vitis platform

○ Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

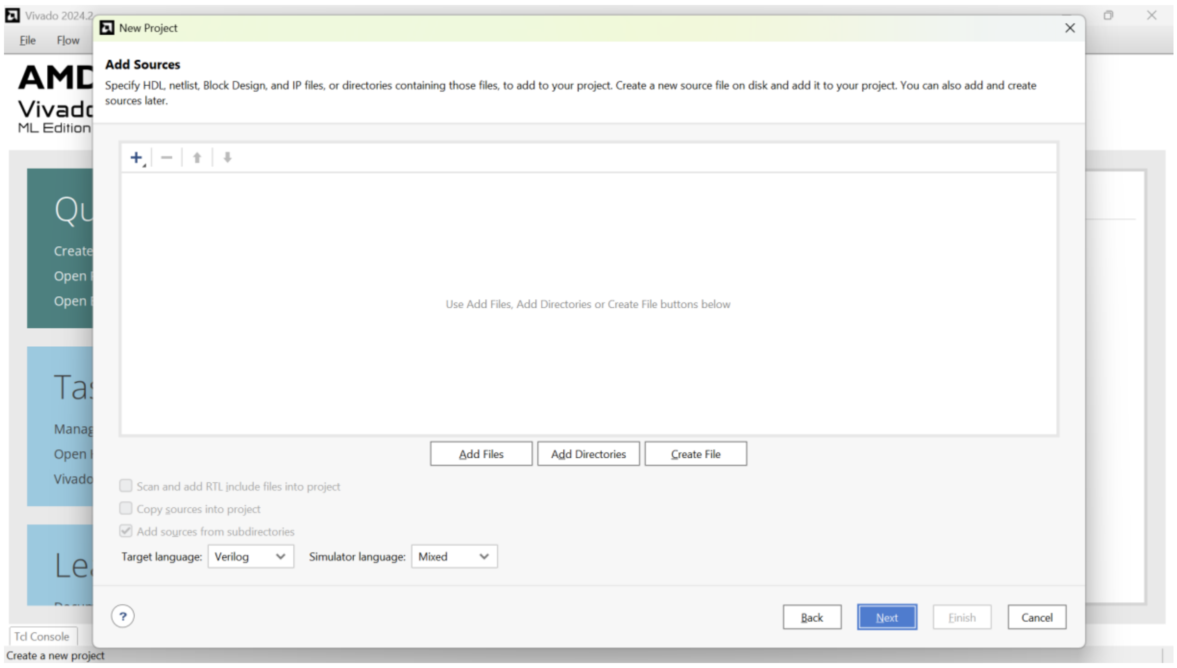
○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify Project File.

○ Example Project
Create a new Vivado project from a predefined template.

FIG 3.20: SELECTING PROJECT TYPE

**STEP6:** Select add files



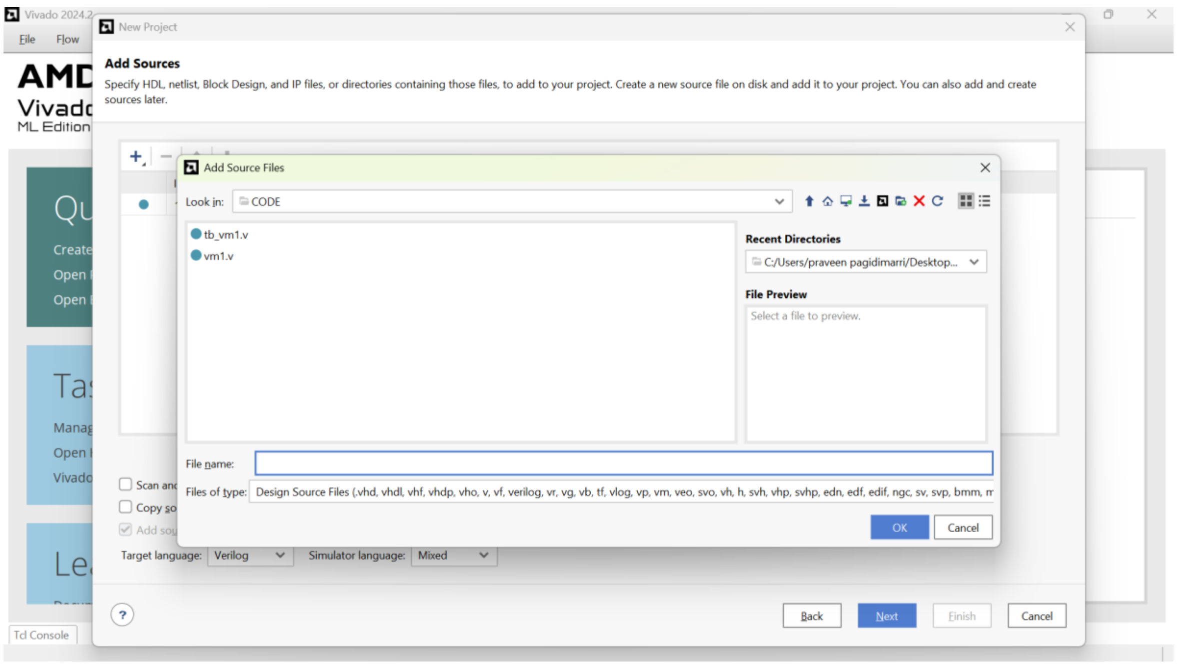
Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

**FIG 3.21: ADDING FILES VIVADO**

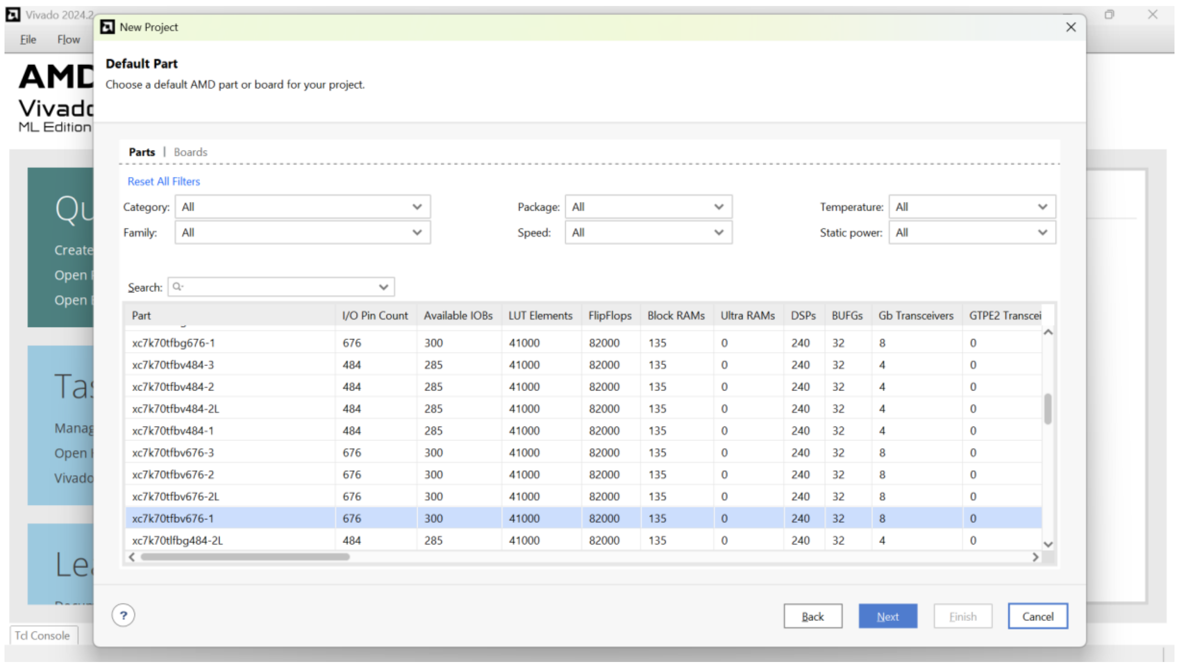
**STEP7**: Then it shows some file select the code click on ok



Vivado 2024.2
New Project
Add Sources
Add Source Files
Look in: CODE
tb_vm1.v
vm1.v
Recent Directories
C:/Users/praveen pagidimarri/Desktop...
File Preview
Select a file to preview.
File name:
Files of type: Design Source Files (.vhd, vhdl, vhf, vhdp, vho, v, vf, verilog, vr, vg, vb, tf, vlog, vp, vm, veo, svo, vh, h, svh, vhp, svhp, edn, edf, edif, ngc, sv, svp, bmm, m
OK
Cancel
Target language: Verilog
Simulator language: Mixed
Back
Next
Finish
Cancel
Tcl Console

**FIG 3.22: SELECTING CODE IN VAVADO**

**STEP8:** Click on next


Default Part

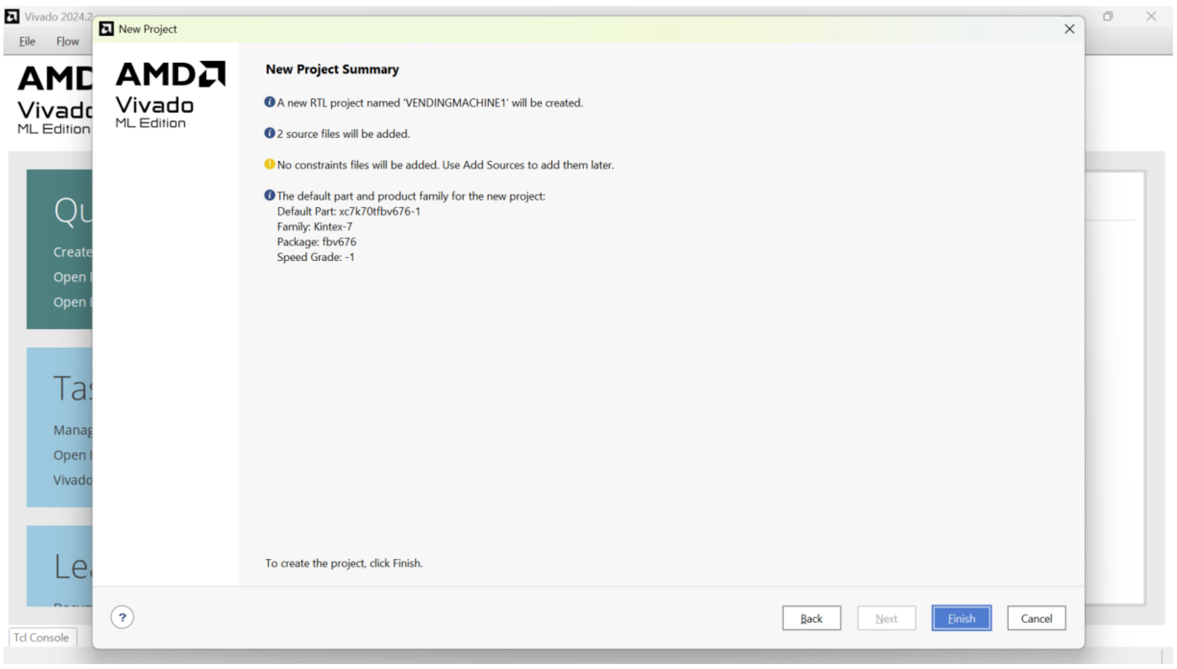
Choose a default AMD part or board for your project.

Category: All
Family: All
Package: All
Speed: All
Temperature: All
Static power: All

| Part | I/O Pin Count | Available IOBs | LUT Elements | FlipFlops | Block RAMs | Ultra RAMs | DSPs | BUFGs | Gb Transceivers | GTPE2 Transcei |
|---|---|---|---|---|---|---|---|---|---|---|
| xc7k70tfbg676-1 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 32 | 8 | 0 |
| xc7k70tfbv484-3 | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 32 | 4 | 0 |
| xc7k70tfbv484-2 | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 32 | 4 | 0 |
| xc7k70tfbv484-2L | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 32 | 4 | 0 |
| xc7k70tfbv484-1 | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 32 | 4 | 0 |
| xc7k70tfbv676-3 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 32 | 8 | 0 |
| xc7k70tfbv676-2 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 32 | 8 | 0 |
| xc7k70tfbv676-2L | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 32 | 8 | 0 |
| xc7k70tfbv676-1 | 676 | 300 | 41000 | 82000 | 135 | 0 | 240 | 32 | 8 | 0 |
| xc7k70tlfbg484-2L | 484 | 285 | 41000 | 82000 | 135 | 0 | 240 | 32 | 4 | 0 |

FIG 3.23: SELECTING THE DEFAULT PART

**STEP9:** Click on finish


New Project Summary

A new RTL project named 'VENDINGMACHINE1' will be created.

2 source files will be added.

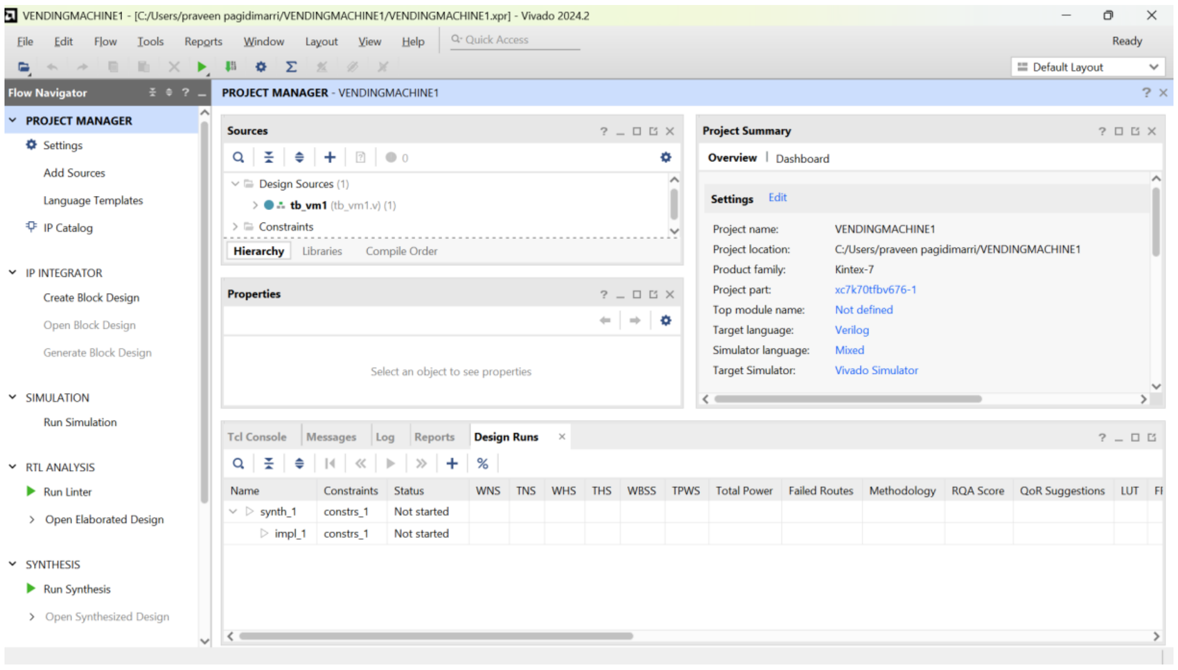
No constraints files will be added. Use Add Sources to add them later.

The default part and product family for the new project:
Default Part: xc7k70tfbv676-1
Family: Kintex-7
Package: fbv676
Speed Grade: -1

To create the project, click Finish.

FIG 3.24: SELECTING THE FINAL STEP OF PROJECT CREATING

**STEP10:** It is the project manager

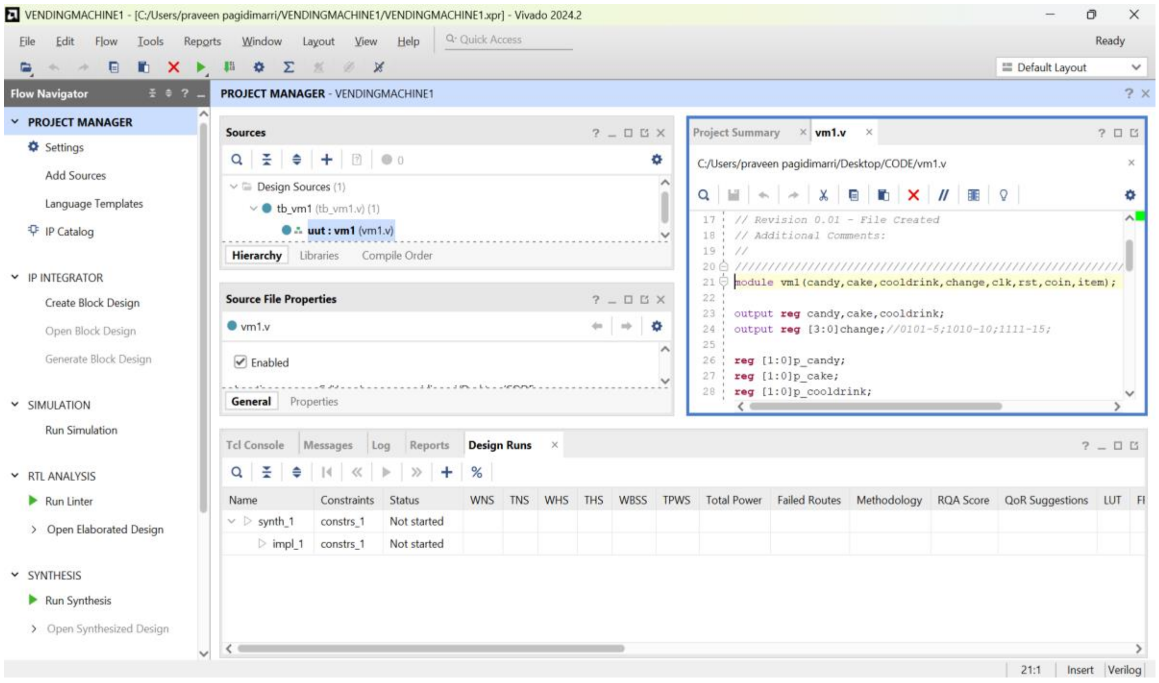


| | |
|---|---|
| Project name: | VENDINGMACHINE1 |
| Project location: | C:/Users/praveen pagidimarri/VENDINGMACHINE1 |
| Product family: | Kintex-7 |
| Project part: | xc7k70tfbv676-1 |
| Top module name: | Not defined |
| Target language: | Verilog |
| Simulator language: | Mixed |
| Target Simulator: | Vivado Simulator |

| Name | Constraints | Status | WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | Methodology | RQA Score | QoR Suggestions | LUT | FI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| synth_1 | constrs_1 | Not started | | | | | | | | | | | | | |
| impl_1 | constrs_1 | Not started | | | | | | | | | | | | | |

FIG 3.25: PROJECT MANAGER OF VIVADO SOFTWERE

**STEP11:** On left side is project summary

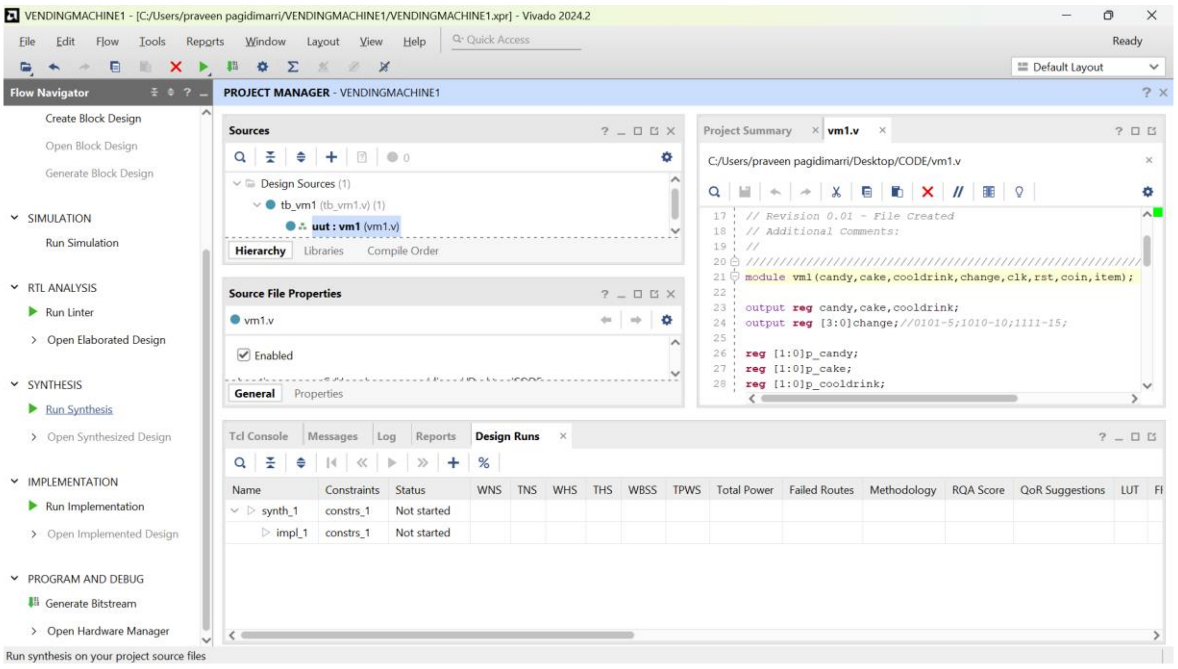


```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
module vm1(candy,cake,cooldrink,change,clk,rst,coin,item);

output reg candy,cake,cooldrink;
output reg [3:0]change;//0101-5;1010-10;1111-15;

reg [1:0]p_candy;
reg [1:0]p_cake;
reg [1:0]p_cooldrink;
```

| Name | Constraints | Status | WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | Methodology | RQA Score | QoR Suggestions | LUT | FI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| synth_1 | constrs_1 | Not started | | | | | | | | | | | | | |
| impl_1 | constrs_1 | Not started | | | | | | | | | | | | | |

FIG 3.26: PROJECT SUMMARY OF VIVADO SOFTWARE

**STEP12:** In design source click on uut: vm1 then click Run Synthesis

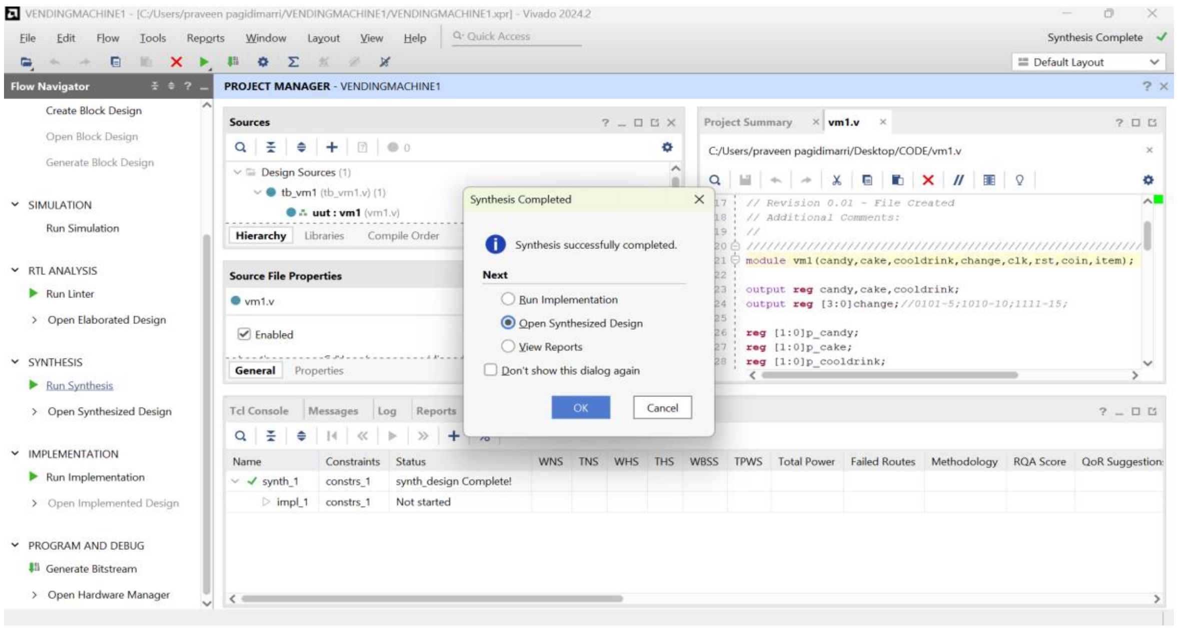

VENDINGMACHINE1 - [C:/Users/praveen pagidimarri/VENDINGMACHINE1/VENDINGMACHINE1.xpr] - Vivado 2024.2
Ready
Flow Navigator
PROJECT MANAGER - VENDINGMACHINE1
Design Sources (1)
tb_vm1 (tb_vm1.v) (1)
uut : vm1 (vm1.v)
Run Synthesis
Run synthesis on your project source files

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
module vm1(candy,cake,cooldrink,change,clk,rst,coin,item);

output reg candy,cake,cooldrink;
output reg [3:0]change;//0101-5;1010-10;1111-15;

reg [1:0]p_candy;
reg [1:0]p_cake;
reg [1:0]p_cooldrink;
```

| Name | Constraints | Status |
|---|---|---|
| synth_1 | constrs_1 | Not started |
| impl_1 | constrs_1 | Not started |

**FIG 3.27: IN DESIGNING SOURCE SELECTING RUN SYNTHESIS**

**STEP13:** Select open synthesized design click ok



Synthesis successfully completed.

Next

( ) Run Implementation
(●) Open Synthesized Design
( ) View Reports

☐ Don't show this dialog again

| Name | Constraints | Status |
|---|---|---|
| synth_1 | constrs_1 | synth_design Complete! |
| impl_1 | constrs_1 | Not started |

**FIG 3.28: OPEN SYNTHESIZED DESIGN CLICK OK**

**STEP14:** Synthesized design

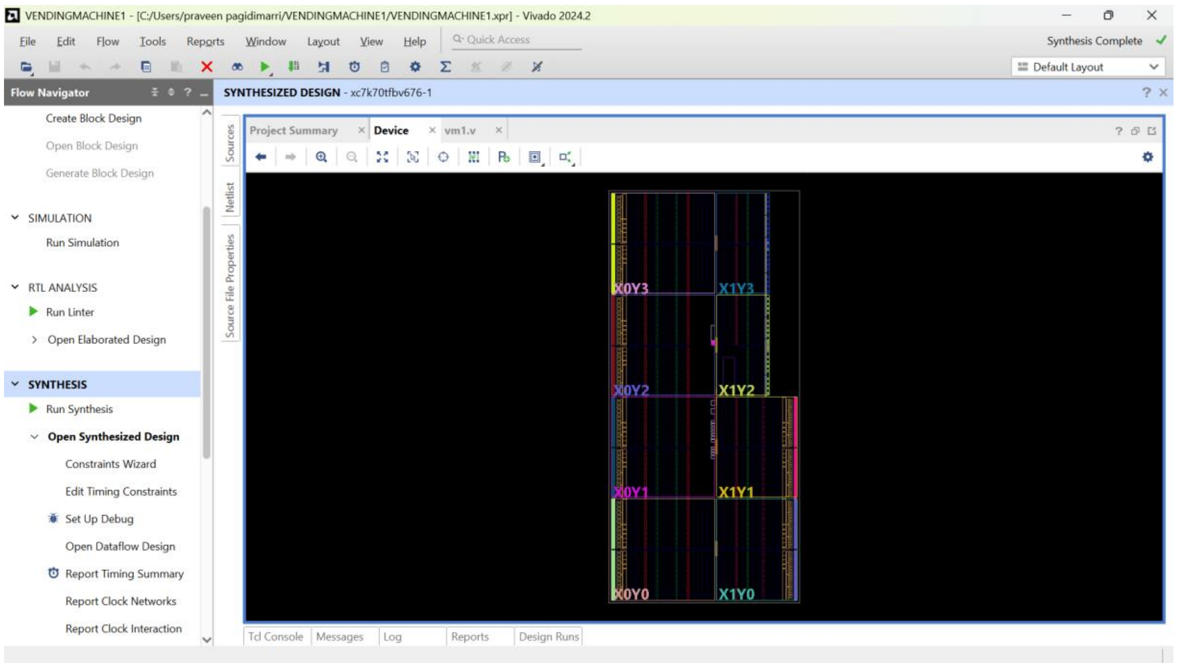


**FIG 3.29: SYNTHESIZED DESIGN**

**STEP15**: Inside synthesized design

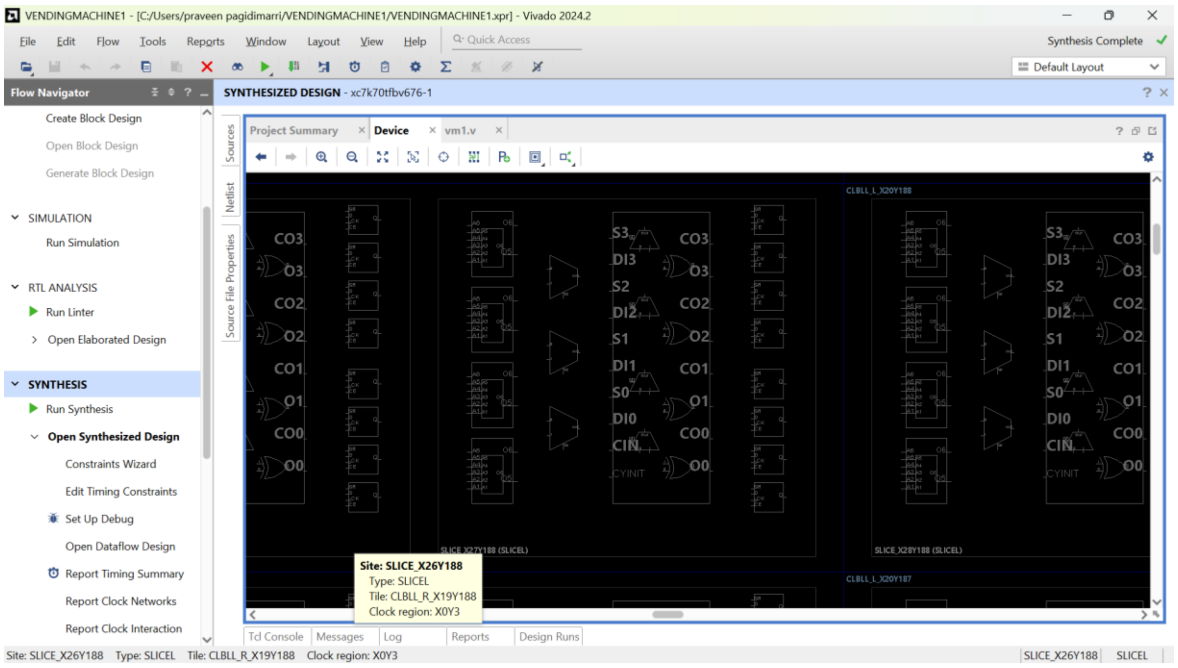
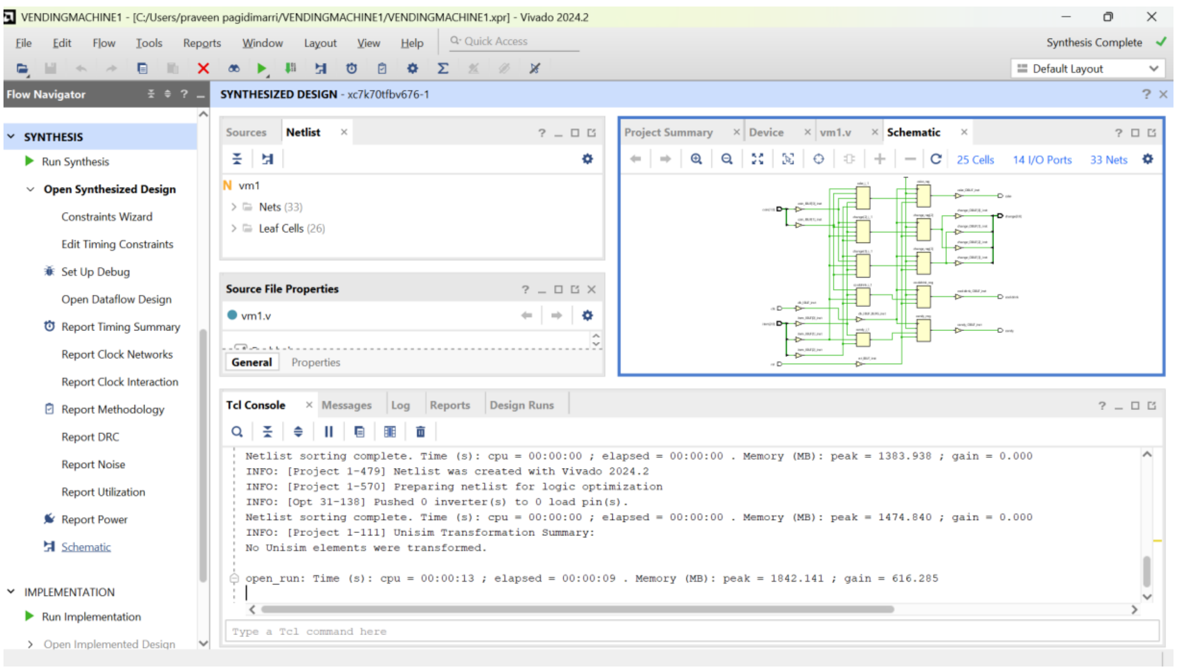


**FIG 3.30: INSIDE SYNTHESIZED DESIGN**

**STEP16:** Clicking on schematic in project summary schematic diagram

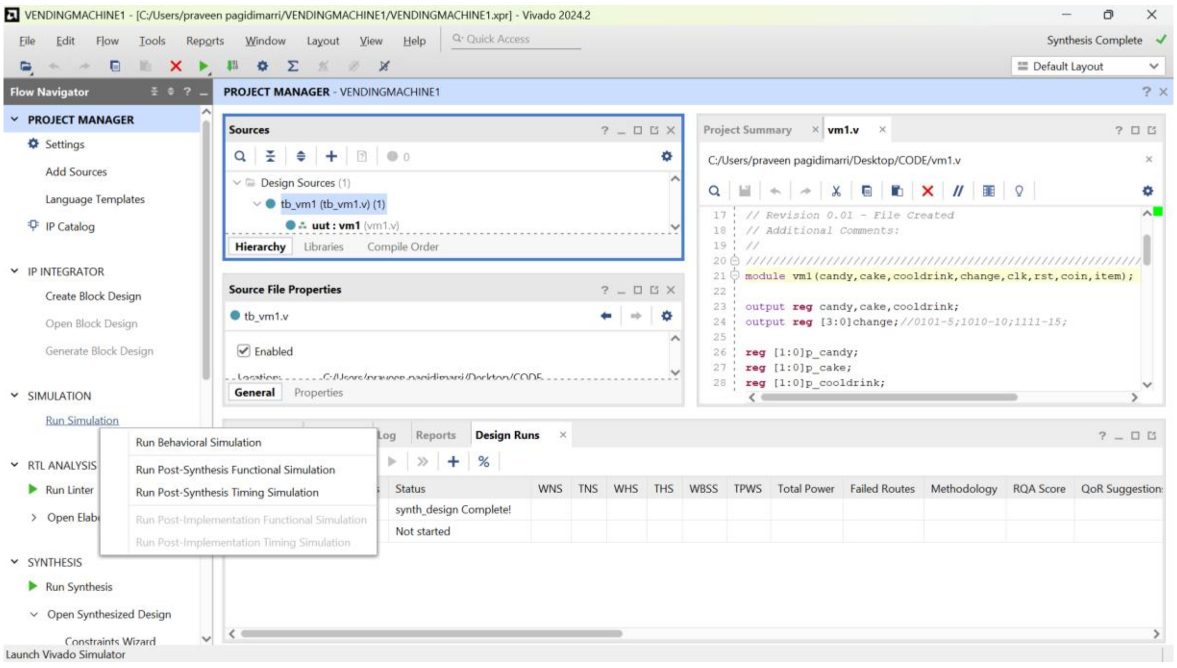


```
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 1383.938 ; gain = 0.000
INFO: [Project 1-479] Netlist was created with Vivado 2024.2
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 1474.840 ; gain = 0.000
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

open_run: Time (s): cpu = 00:00:13 ; elapsed = 00:00:09 . Memory (MB): peak = 1842.141 ; gain = 616.285
```

Type a Tcl command here

**FIG 3.31: SELECTING ON SCHEMATIC IN PROJECT SUMMARY**

**STEP 17:** In design source click on tb_vm1 then Run Simulation select Run Behavioural Simulation



```
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////
module vm1(candy,cake,cooldrink,change,clk,rst,coin,item);

output reg candy,cake,cooldrink;
output reg [3:0]change;//0101-5;1010-10;1111-15;

reg [1:0]p_candy;
reg [1:0]p_cake;
reg [1:0]p_cooldrink;
```

| Status | WNS | TNS | WHS | THS | WBSS | TPWS | Total Power | Failed Routes | Methodology | RQA Score | QoR Suggestion |
|---|---|---|---|---|---|---|---|---|---|---|---|
| synth_design Complete! | | | | | | | | | | | |
| Not started | | | | | | | | | | | |

**FIG 3.32: RUN SIMULATION AND SELECT BEHAVIOURAL SIMULATION**

**STEP18:** It is simulation

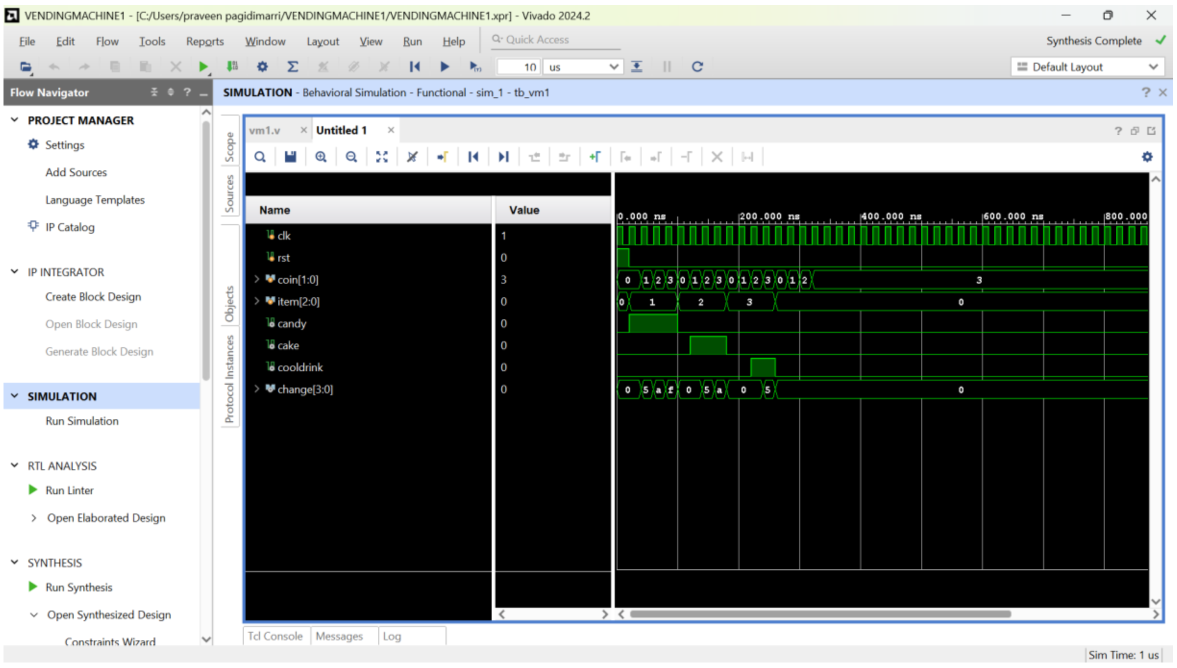


FIG 3.33: FINAL PART IN VIVADO SIMULATION PART

# CHAPTER 4

# WORKING OF THE PROJECT

## 4.1 WORKING OF THE PROJECT

### BLOCK DIAGRAM

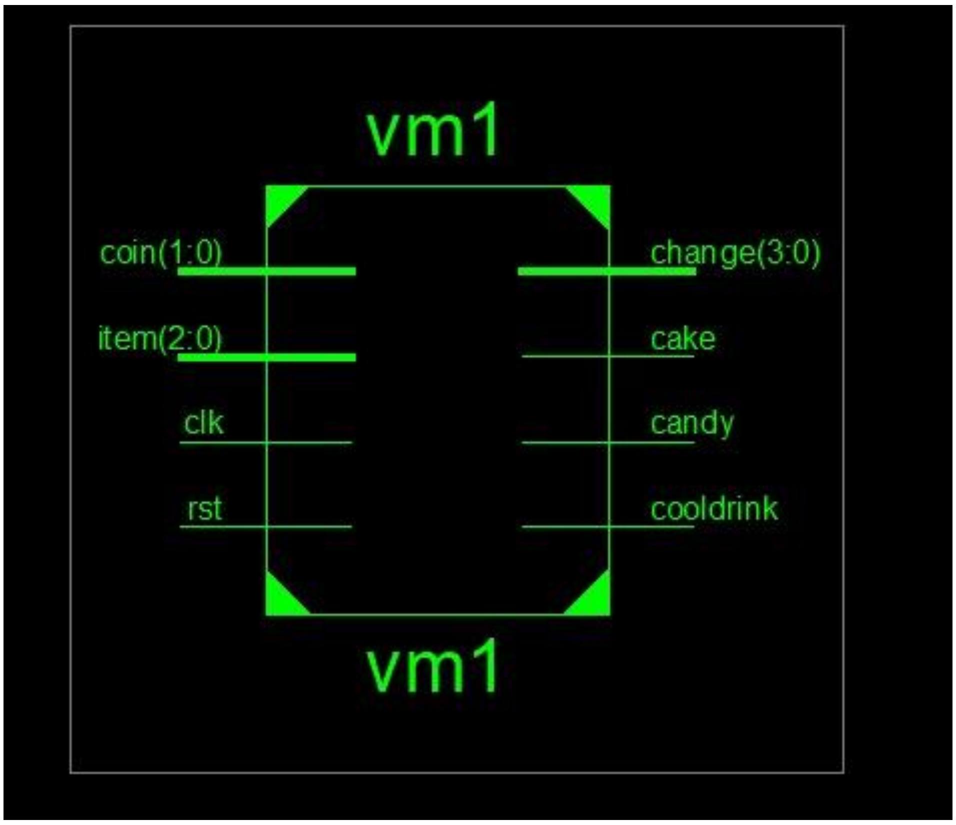


**FIG 4.1: WORKING BLOCK DIAGRAM**

Automation has revolutionized the world. Today instead of shopkeepers selling items manually, we have automatic machines accepting money and dispensing products. Vending machines are devices where the customers insert coins or credit cards to purchase newspapers, snacks, beverages, tickets, etc. It is the most practical method of purchasing.

Vending machines operate through a combination of electromechanical processes. Initially, the user inputs payment, which is then validated by the machine's internal system, checking for authenticity and value. Once payment is confirmed, the user selects a product via a keypad or touchscreen. This selection triggers an internal mechanism, often involving motors and coils, to release the chosen item.

Sensors within the machine ensure the product has been dispensed correctly, and if not, the system may retry or initiate a refund. Modern machines also incorporate digital technologies, allowing for cashless payments and remote inventory monitoring, thereby providing a very convenient and efficient way for consumers to obtain products.

In the project, we have designed a sequential circuit that dispenses various products when coins are inserted into the machine. The article describes the modeling of the Finite State-based Vending Machine using the mealy model. The code for the vending machine is written in Verilog HDL and simulated in the Model Sim. A finite state machine is an abstract machine that can be in any number of states at any given time. It is a model that describes the synchronous sequential machine. Every sequential circuit has inputs, outputs, and an internal state.

Based on the present state and the present input, the finite state machines can be of two types:

1. **MooreFiniteStateMachine**

If the next state of the system depends only on the present state, then it is called a Moore Finite State Machine.

2. **MealyFiniteStateMachine**

If the next state of the system depends on both the present state and the present input, then it is called a Mealy Finite State Machine.

The Mealy Finite State Machine is implemented in the project. In this project, we have designed a vending machine that can dispense three products of different prices with additional features of '**return balance**' when the money of higher denomination is inserted and 'return money' when the request is cancelled. Any Sequential digital circuit can be converted into a state machine using a state diagram. In a state machine, the circuit's output is a different set of states i.e. each output is a state. There is a State Register to hold the state of the machine. There is a next state logic to decode the next state.

Furthermore, many vending machines now feature temperature control, ensuring perishable items like beverages and snacks are stored at optimal conditions. Advanced models can also display nutritional information and allergen warnings, catering to consumer health needs. Network connectivity allows for real-time sales data collection, enabling operators to optimize product offerings and restock efficiently. Some machines even offer interactive displays, providing advertising and entertainment options. The integration of mobile payment systems and loyalty programs enhances the user experience.

There is also an output register that defines the output of the machine. The next state (ns) logic is the sequential part of the machine. The output and present state (ps ) is the Register part.

The products available in the program are as follows.

Newspaper- priced at Rs.5

Cadbury bar- priced at Rs. 10

Tropicana juice- priced at Rs. 15

The buyer can insert money of the following denomination.

Rs. 5

Rs. 10

Rs. 20

The states defined in the design of the vending machine are:

Void State

Five State

Ten State

Fifteen State

As per the money inserted by the buyer, the machine will output the products and change if extra money is inserted.

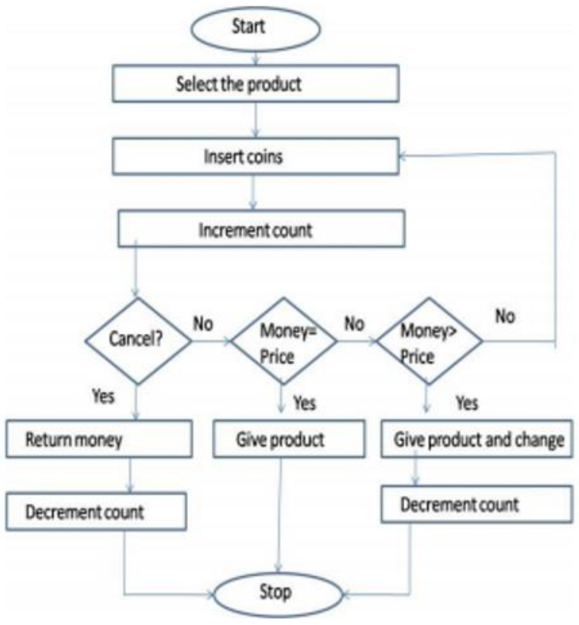


**FIG 4.2: FLOW CHART OF VENDING MACHINE**

The flowchart explains the entire process of how the vending machine functions in the program. Initially, the customer must select the product followed by inserting the money. After inserting the money, the machine checks if the cancel button is pressed by the customer. If the customer presses the cancel button, the money is returned to the customer. Else the machine checks the inserted money with the price of the selected product. If both are equal, then the machine dispenses the product. If the inserted money is greater than the price, then the vending machine gives the appropriate product along with the change. If the inserted money is less than the price of the selected product, then the vending machine waits for the customer to insert more money.

The flowchart illustrates the operational logic of a simple vending machine, beginning with the user selecting a product. Subsequently, the user inserts coins, and the machine increments the count of the money received. A decision point is then reached: if the user cancels, the machine returns the money and stops. If not, the machine checks if the inserted money equals the product's price. If it does, the product is dispensed, and the process ends. If the money exceeds the price, the product is dispensed along with the change, and the process concludes. However, if the inserted money is less than the price, the process loops back, prompting the user to insert more coins. This cycle continues until either the correct amount is inserted, a cancellation occurs, or change is dispensed, ultimately leading to the "Stop" state. The flowchart effectively portrays the sequential and conditional steps involved in a basic vending machine transaction.

This flowchart represents a foundational model of a vending machine's transaction process, highlighting the core interactions between the user and the machine. It simplifies the complex electronic and mechanical operations into a series of logical steps, focusing on the user's perspective. The "Select the product" stage initiates the transaction, setting the stage for the payment process. The "Insert coins" and "Increment count" steps form the iterative payment phase, where the machine accumulates the user's input. The inclusion of the "Cancel?" decision point provides the user with an option to terminate the transaction, ensuring flexibility and user control. The "Money = Price?" and "Money > Price?" decision points are critical for determining the outcome of the transaction, distinguishing between exact payment, overpayment, and underpayment scenarios. The loop back to "Insert coins" emphasizes the machine's ability to handle insufficient funds, prompting the user for additional payment. The "Give product" and "Give product and change" actions signify successful transactions, marking the culmination of the process.

# CHAPTER 5
# RESULTS

## 5.1 RESULT OF THE PROJECT

The Verilog testbench (tb_vm1.v) is designed to verify the functionality of a vending machine module (vm1). It initializes various input signals, applies test cases, and observes the system's behaviour. The testbench first sets the clock (clk) to 1, enables the reset (rst), and initializes the coin and item inputs to zero. The reset is then deactivated to allow normal operation. The vending machine accepts a 2-bit coin input representing different denominations and a 3-bit item input to select different products, including candy, cake, and cooldrink.

The testbench systematically tests each product selection by setting the item input to a specific value and varying the coin input across four different values. The first test case selects candy (item = 3'b001) and applies coin values sequentially to observe if the correct product is dispensed. The same approach is used to test cake (item = 3'b010) and cooldrink (item = 3'b011). Additionally, the testbench verifies the scenario where no item is selected (item = 3'b000), checking if the vending machine correctly handles cases with invalid selections.

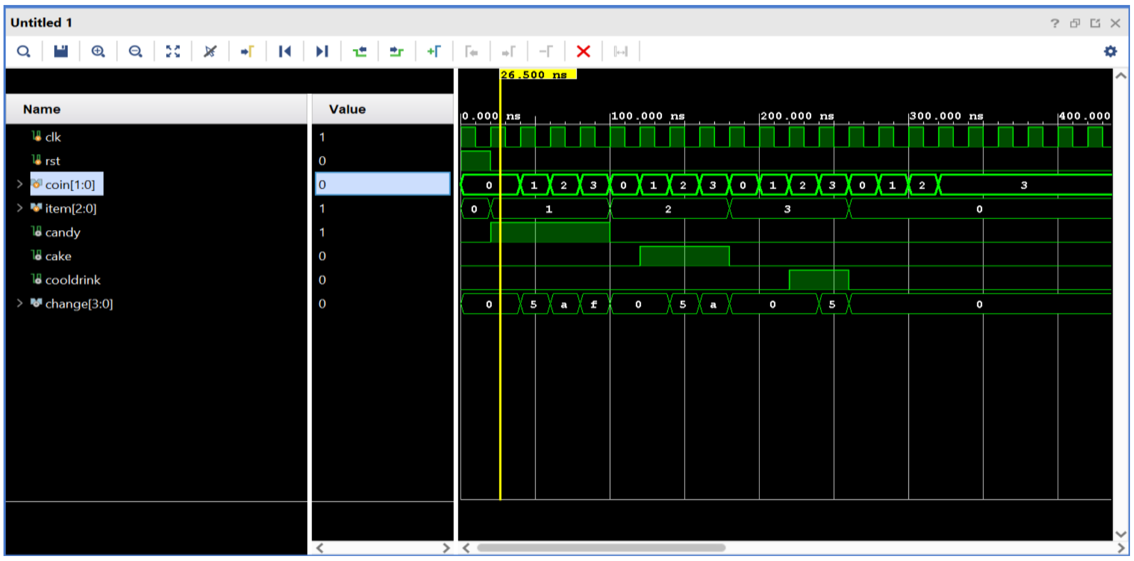
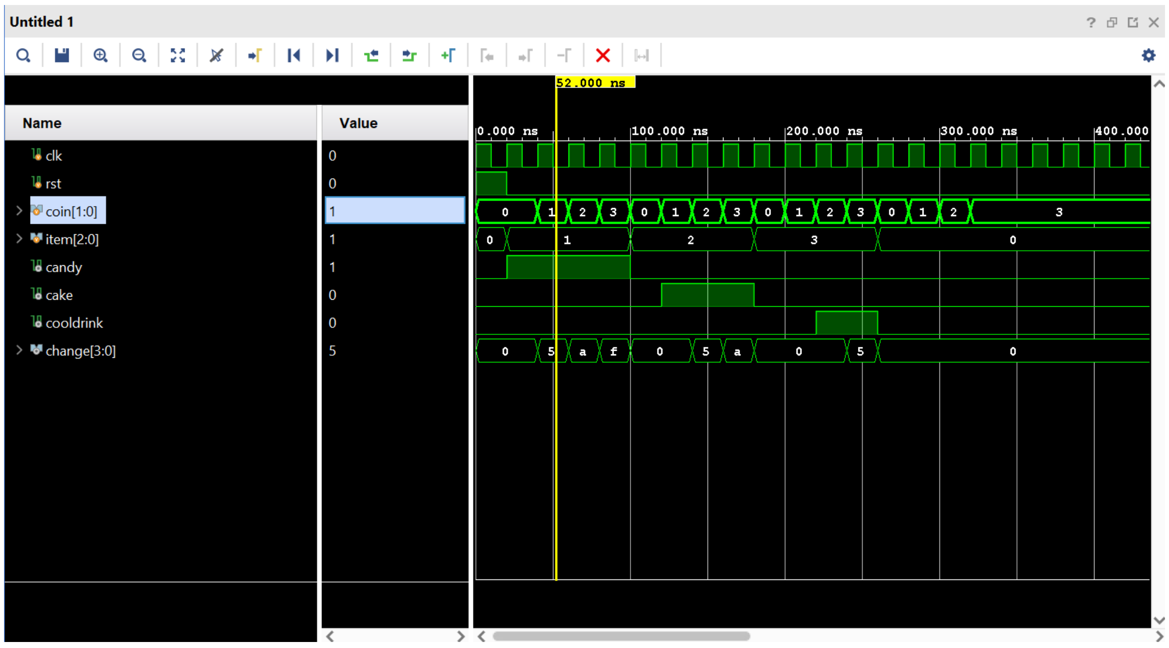


FIG 5.1: CASE 1

A clock signal is generated using an always block that toggles the clk signal every 10 time units, ensuring the vending machine operates synchronously.

The testbench provides a comprehensive validation of the vending machine's behavior by simulating various input combinations and monitoring the outputs, including the dispensed product signals (candy, cake, cooldrink) and the returned change (change[3:0]). The simulation results will help determine whether the vending machine logic is functioning correctly.

The provided figure presents a simulation waveform, a crucial tool in digital circuit design, used to visualize and verify the behavior of a circuit over time. The right side of the image displays the waveforms, where the horizontal axis represents time, measured in

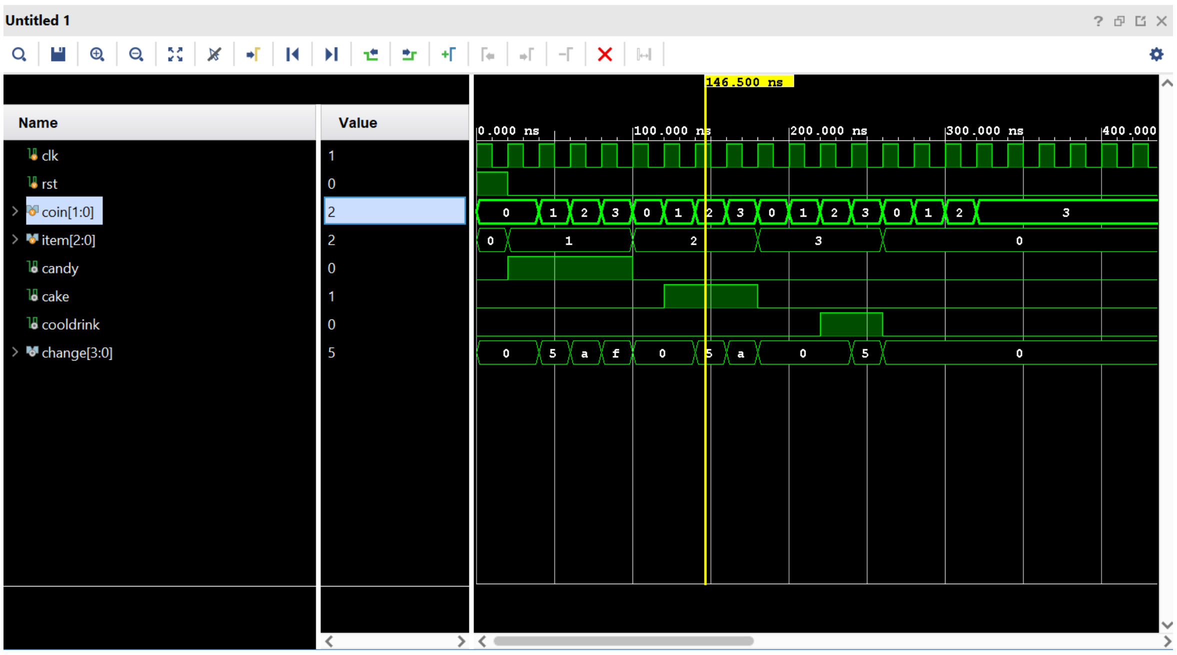


| Name | Value |
|---|---|
| clk | 0 |
| rst | 0 |
| coin[1:0] | 1 |
| item[2:0] | 1 |
| candy | 1 |
| cake | 0 |
| cooldrink | 0 |
| change[3:0] | 5 |

**FIG 5.2: CASE 2**

nanoseconds, and the vertical axis indicates the value of each signal. The left side enumerates the signals being simulated, including a clock signal (clk), a reset signal (rst), user inputs for item selection (item [2:0]) and coin insertion (coin [1:0]), output signals for item dispensing (candy, cake, cooldrink), and the calculated change (change [3:0]). The waveforms illustrate the dynamic changes in these signal values, allowing designers to observe the circuit's response to different inputs and conditions. For instance, the item and coin signals show user selections, while the dispensing signals and change signal reflect the machine's output. The yellow vertical line at 52.000 ns pinpoints a specific time for detailed analysis. Overall, this simulation waveform serves as a visual representation of the circuit's operation, enabling designers to confirm its functionality and identify any potential issues.

Notably, the waveform includes a clock signal (clk), characterized by its periodic square wave, which serves as the timing reference for the circuit. A reset signal (rst) is also depicted, transitioning from low to high momentarily, indicating an initialization phase. The user's input is represented by the coin [1:0] signal, which cycles through values 0 to 3, likely representing different coin denominations, and the item [2:0] signal, which



| Name | Value |
|---|---|
| clk | 1 |
| rst | 0 |
| coin[1:0] | 2 |
| item[2:0] | 2 |
| candy | 0 |
| cake | 1 |
| cooldrink | 0 |
| change[3:0] | 5 |

**FIG 5.3: CASE 3**

indicates the selected product. The output of the circuit is shown through the candy, cake, and cooldrink signals, which go high when the corresponding item is dispensed, and the change  [3:0] signal, displaying the calculated change in hexadecimal format. The yellow vertical line at 146.500 ns marks a specific point in time for detailed analysis. This waveform effectively visualizes the circuit's operation, enabling designers to verify its functionality and timing characteristics.

The figure presents a digital circuit simulation waveform, a visual representation of signal behaviour over time, crucial for verifying the functionality of digital designs. The horizontal axis represents time in nanoseconds, while the vertical axis displays the values of various signals. The simulation includes a clock signal (clk), a periodic square wave essential for synchronizing the circuit's operations. A reset signal (rst) is also present, transitioning briefly to a high state, indicating an initialization phase. The coin [1:0] signal represents user input, cycling through values 0 to 3, likely corresponding to different coin denominations.

The item [2:0] signal indicates the selected item, with values 1, 2, 3, and 0 representing different choices.

The output of the circuit is shown through the candy, cake, and cooldrink signals, which go high when the respective item is dispensed, and the change [3:0] signal, displaying the calculated change in hexadecimal format. The yellow vertical line at 168.000

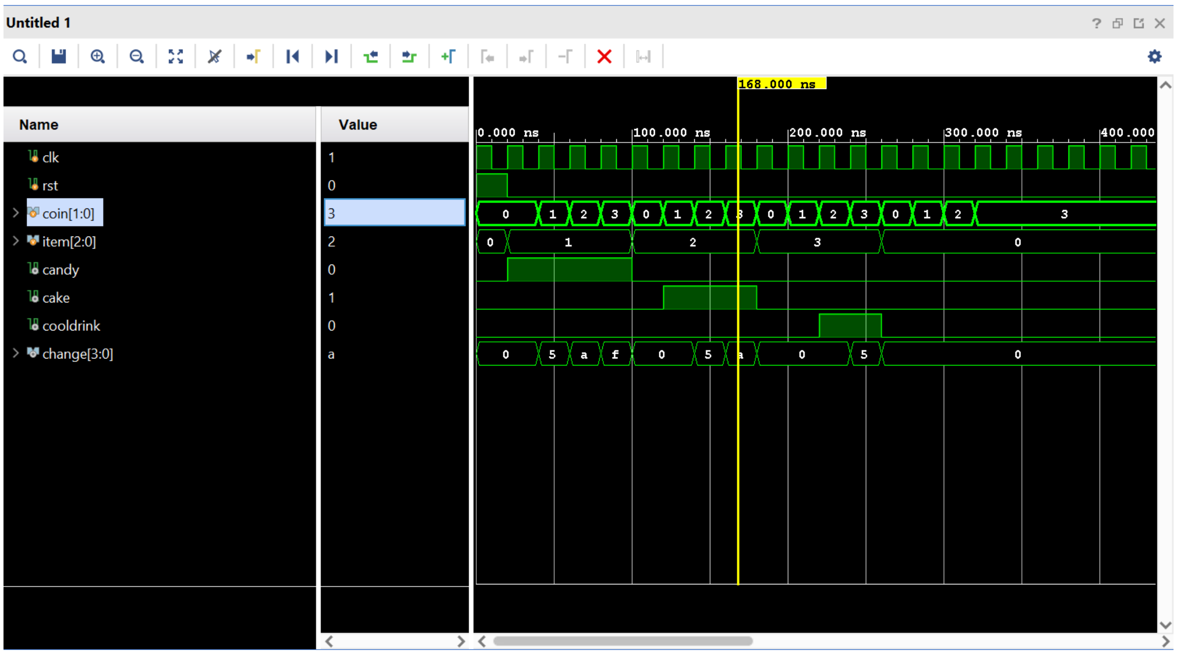


**FIG 5.4: CASE 4**

marks a specific time for detailed analysis of the signal values. This waveform allows designers to observe the timing and behaviour of the circuit, ensuring it functions as intended.

The image displays a simulation waveform, a critical tool for verifying the functionality of digital circuits. The horizontal axis represents time, measured in nanoseconds, while the vertical axis shows the values of various signals. The simulation includes a clock signal (clk), a periodic square wave essential for synchronizing the circuit's operations, and a reset signal (rst), which is initially low and briefly goes high for initialization.

The output of the circuit is shown through the candy, cake, and cooldrink signals, which go high when the respective item is dispensed, and the change [3:0] signal, displaying the calculated change in hexadecimal format.

The simulation includes a clock signal (clk), a periodic square wave essential for synchronizing the circuit's operations.

A reset signal (rst) is also present, transitioning briefly to a high state, indicating an initialization phase. The coin [1:0] signal represents user input, cycling through values 0 to 3, likely corresponding to different coin denominations.

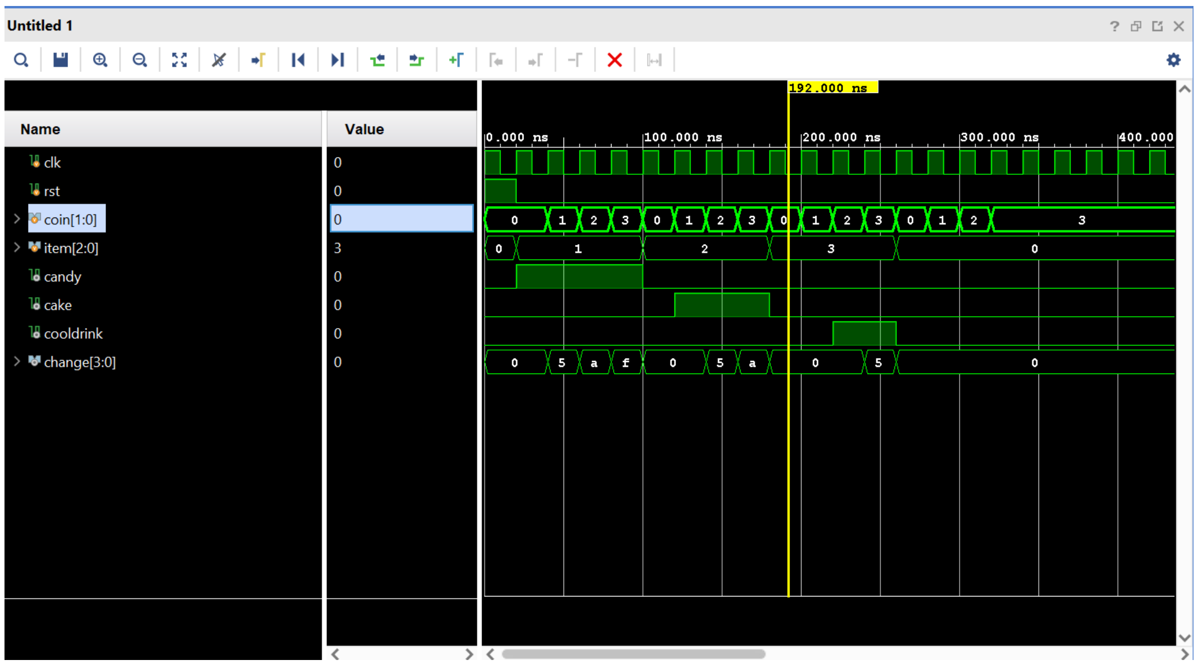


**FIG 5.5: CASE 5**

The coin[1:0] signal represents user input, cycling through values 0 to 3, likely indicating different coin denominations. The item[2:0] signal indicates the selected item, with values 1, 2, 3, and 0 representing different choices. The output of the circuit is shown through the candy, cake, and cooldrink signals, which go high when the respective item is dispensed, and the change[3:0] signal, displaying the calculated change in hexadecimal format. The yellow vertical line at 192.000 ns marks a specific time for detailed analysis. This waveform allows designers to observe the timing and behaviour of the circuit, ensuring it functions as intended.

The image depicts a simulation waveform, a fundamental tool used in digital circuit design for verifying the behaviour of a circuit over time. The horizontal axis represents time, measured in nanoseconds, while the vertical axis displays the values of various signals. The simulation includes a clock signal (clk), a periodic square wave essential for synchronizing the circuit's operations, and a reset signal (rst), which is initially low and briefly goes high for initialization. The coin [1:0] signal represents user input, cycling through values 0 to 3, likely indicating different coin denominations. The item [2:0] signal indicates the selected item, with values 1, 2, 3, and 0 representing different choices.

## 5.2 APPLICATIONS

**1. Automated Retail Systems:** Vending machines designed with Verilog can be used in automated retail applications where items such as snacks, beverages, or electronics are dispensed upon payment.

**2.Coin-Based Systems:** Verilog can be used to model the logic of handling and verifying coin inputs, ensuring that only valid payments are processed and correct change is given.

**3.Automated Ticket Machines:** A vending machine design in Verilog can be adapted for ticket vending systems for transportation (e.g., metro tickets, bus tickets), where users input money and select a ticket type.

**4.Smart Vending Machines:** In modern vending systems, Verilog can be used to model systems that interface with IoT devices for inventory management, or even be integrated with credit/debit card readers.

## 5.3 ADVANTAGES

**1.Hardware-Level Design:** Verilog allows you to describe the vending machine at the hardware level, which provides a clear and precise way of simulating physical components like coin insertion, dispensing mechanisms, and user inputs.

**2.Parallelism:** Verilog allows for parallelism in the design, meaning that multiple tasks (e.g., accepting coins, checking button presses, dispensing products) can occur simultaneously. This is essential for the responsive nature of a vending machine.

**3.Accuracy and Reliability:** Once verified and synthesized, Verilog designs are highly reliable when implemented on FPGA or ASIC devices. The digital circuits derived from Verilog descriptions are deterministic and follow well-defined timing constraints.

**4.Synthesis for Hardware:** Verilog designs can be synthesized into hardware for real-world implementation. Once verified via simulation, the code can be directly mapped to hardware devices such as FPGAs, providing a hardware-accelerated solution for the vending machine system.

**5.Simulation and Verification:** With Verilog, you can easily simulate the entire vending machine operation (including different inputs and states) using various simulation tools. This helps in ensuring that the design works as expected before actual deployment.

# CHAPTER 6
# CONCLUSION & FUTURE SCOPE

## CONCLUSION

In this work, we presented the design, implementation, and verification of a vending machine using Finite State Machine (FSM) methodology in Verilog HDL, implemented in Quartus II Software in the STRATIX-II FPGA. The proposed design and implementation provide a reliable and efficient solution for users to purchase items from the vending machine. The FSM model is used to manage the multiple states of the vending machine, including “idle” “accepting coins,” “dispensing item,” and “returning change.” The FSM is implemented as a state diagram in Verilog HDL, which provides a highlevel abstraction of the hardware and makes it easy to simulate, synthesize, and implement the design. The implementation of the vending machine is done using the Quartus synthesis tool. The tool optimizes the design for timing, area, and power, performs physical design, including placement and routing, to meet the design’s timing, power, and area constraints.

The proposed design is validated using a test bench, which simulates the behavior of the vending machine. The simulation results are used to verify that the design meets the required specifications, and that the FSM behaves as expected. The proposed design is also can be implemented on an FPGA to demonstrate its effectiveness in a real-world scenario. The results of the implementation show that the proposed design meets the required specifications and performs well in a real-world scenario.

The design’s performance, power consumption, and area are analyzed, and the results show that the design is efficient and reliable. The proposed design and implementation demonstrate the feasibility and effectiveness of using FSM in Verilog HDL and implementing the design in Genus and Encounter for vending machine applications. The proposed design can be further extended and optimized for more complex vending machines, and it can also be used asa basis for other digital systems that require FSM-based designs. The proposed design runs at 200MHz frequency.

## FUTURE SCOPE

**1. Core Design & Verification:**

**Verilog HDL Implementation:**

- Create a digital circuit model for a vending machine, handling coin inputs, product selection, and dispensing.
- Employ a state machine to manage the vending process (idle, coin insertion, selection, dispensing).
- Implement logic for calculating change.

**Verification:**

- Develop a Verilog testbench to simulate and verify the design's functionality.
- Utilize test cases to cover various scenarios (valid/invalid inputs, edge cases).
- Employ System Verilog for advanced verification (constrained-random testing, assertions).
- Formal Verification to prove correct operation.

**2. Brief Future scope:**

The future of a Verilog HDL vending machine project lies in transforming it from a basic digital circuit into a sophisticated, connected, and intelligent system. Key areas of development include:

**Advanced Payment Systems:**

- Integration of cashless payment options (NFC, credit cards, mobile wallets).
- Secure transaction processing.

**Enhanced User Interface:**

- Touchscreen displays for product selection and information.
- Personalized user experiences.

**IoT Connectivity:**

- Remote monitoring and management of vending machines.
- Real-time data collection and analysis (sales, inventory).
- Predictive maintenance.

**AI Integration:**

- Demand forecasting and inventory optimization.
- Personalized product recommendations.
- Facial recognition for age verification.

**Modular Design & Customization:**

- Create a reusable and scalable hardware platform.
- Allow for easy customization of product offerings and payment options.

**FPGA/SoC Deployment:**

- Implement the design on FPGAs or SoCs for real-world deployment.
- Integrate with embedded systems for advanced functionality.

**Cybersecurity:**

- Implementation of security to prevent fraud.
- Secure communication protocols.

In essence, the project evolves from a hardware design exercise to a platform for innovation in automated retail, leveraging connectivity, data, and AI to create more efficient and user-friendly vending solutions.

# REFERENCE

[1] Chopde, Abhay, et al. "Vending Machine Using Verilog (FPGA)." *2024 4th Asian Conference on Innovation in Technology (ASIANCON)*. IEEE, 2024.

[2] Fuad, Mahamudul Hassan, et al. "Design of a Vending Machine Using Verilog HDL and Implementation in Genus & Encounter." *European Journal of Electrical Engineering and Computer Science* 7.6 (2023): 88-95.

[3] Ravikumar, P., MV Ganeswara Rao, and Vamaraju Nikitha. "Verilog Based Automated Retail System." *2024 OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0*. IEEE, 2023.

[4] Rakshith, S., and S. Niranjana. "User-friendly Vending Machines: A Moore FSM Perspective." *2023 5th IEEE Global Conference for Advancement in Technology (GCAT)*. IEEE, 2023.

[5] Barra, Marco Antonio Quispe, et al. "Modeling and Simulation of a Vending Machine Through FSM Using VHDL." *Brazilian Technology Symposium*. Cham: Springer Nature Switzerland, 2022.

# APPENDIX

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:18:00 03/22/2025
// Design Name:
// Module Name:    vm1
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module vm1(candy,cake,cooldrink,change,clk,rst,coin,item);

output reg candy,cake,cooldrink;
output reg [3:0]change;//0101-5;1010-10;1111-15;
reg [1:0]p_candy;
reg [1:0]p_cake;
reg [1:0]p_cooldrink;
reg [1:0]p_extra;
input [2:0]item;//000-no item,001-candy,010-cake,011-cooldrink
input clk,rst;
input [1:0]coin;//00-5rs,01-10rs,10-15rs,11-20rs;
parameter s0 = 3'b000;
parameter s1 = 3'b001;
parameter s2 = 3'b010;
parameter s3 = 3'b011;

initial
begin
p_candy=2'b00;
p_cake=2'b01;
p_cooldrink=2'b10;
p_extra=2'b11;
end

always @(posedge clk)
begin
if (rst==1)begin
candy=0;
cake=0;
cooldrink=0;
change=0;end

else begin
case(item)
//..........................................candy
s1:begin if(coin==p_candy)begin
candy=1;
cake=0;
cooldrink=0;
change=0;end
else if(coin==p_cake)begin
candy=1;
cake=0;

cooldrink=0;
change=4'd5;end
```

```
else if(coin==p_cooldrink)begin
candy=1;
cake=0;
cooldrink=0;
change=4'd10;end
else if(coin==p_extra)begin
candy=1;
cake=0;
cooldrink=0;
change=4'd15;end
else begin
candy=0;
cake=0;
cooldrink=0;
change=4'd0;end
end
//.......................................
//.......................................cake
s2:begin if(coin==p_candy)
begin
candy=0;
cake=0;
cooldrink=0;
change=0;
end
else if(coin==p_cake)
begin
candy=0;
cake=1;
cooldrink=0;
change=0;
end
else if(coin==p_cooldrink)
begin
```

```
candy=0;
cake=1;
cooldrink=0;
change=4'd5;
end
else if(coin==p_extra)
begin
candy=0;
cake=1;
cooldrink=0;
change=4'd10;
end
else
begin
candy=0;
cake=0;
cooldrink=0;
change=4'd0;
end
end
//.......................................
//.......................................cooldrink
s3:begin if(coin==p_candy)
begin
candy=0;
cake=0;
cooldrink=0;
change=0;
end
else if(coin==p_cake)
begin
candy=0;
cake=0;
cooldrink=0;
```

```
change=0;
end
else if(coin==p_cooldrink)
begin
candy=0;
cake=0;
cooldrink=1;
change=0;
end
else if(coin==p_extra)
begin
candy=0;
cake=0;
cooldrink=1;
change=4'd5;
end
else
begin
candy=0;
cake=0;
cooldrink=0;
change=4'd0;
end
end
//..............................................
default: begin
candy=0;
cake=0;
cooldrink=0;
change=4'd0;
end
//..............................................
endcase
end
```

```
end
endmodule
```

**Test bench**

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   20:05:26 03/22/2025
// Design Name:   vm1
// Module Name:   F:/backppp/BACKP/vendingmachine1/tb_vm1.v
// Project Name:  vendingmachine1
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: vm1
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module tb_vm1;
	// Inputs
	reg clk;
	reg rst;

	reg [1:0] coin;
	reg [2:0] item;
```

```
// Outputs
wire candy;
wire cake;
wire cooldrink;
wire [3:0] change;

// Instantiate the Unit Under Test (UUT)
vm1 uut (
        .candy(candy),
        .cake(cake),
        .cooldrink(cooldrink),
        .change(change),
        .clk(clk),
        .rst(rst),
        .coin(coin),
        .item(item)
);

initial begin
        // Initialize Inputs
        clk = 1;
        rst = 1;
        coin = 2'b00;
        item = 3'b000;

        // ................................candy
        #20;
        rst = 0;
        item = 3'b001;
        coin = 2'b00;

        #20;
        coin=2'b01;
```

```
                #20;
                coin=2'b10;
                #20;
                coin=2'b11;
//................................cake
                 #20;
                item = 3'b010;
                coin = 2'b00;
                #20;
                coin=2'b01;
                #20;
                coin=2'b10;
                #20;
                coin=2'b11;
                 //.................................cooldrink
                 #20;
                item = 3'b011;
                coin = 2'b00;
                #20;
                coin=2'b01;
                #20;
                coin=2'b10;
                #20;
                coin=2'b11;
 //.......................................no item
                  #20;
                item = 3'b000;
                coin = 2'b00;
                #20;
                coin=2'b01;


                #20;
                coin=2'b10;
```

```
                    #20;
                    coin=2'b11;
        //..................................
                      #20;
                    // Add stimulus here

            end
      always #10 clk=~clk;
endmodule
```