**Design and Implementation of VGA Controller**

**A Major Project report submitted to**

##### JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

##### ANANTHAPUR, ANANTHAPURAMU

in partial fulfillment of the requirement for the award of degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

|  |  |
| --- | --- |
| **Sukamanchi Siva Chaitanya** | **(219X1A04J0 )** |
| **Orjineni Veeresh** | **(219X1A04K1)** |
| **Dasa Sai Kiran** | **(219X1A04A9)** |

###### Submitted by

**Under the esteemed guidance of**

**Smt. D. Rohini, M. Tech,**

Assistant Professor, ECE Department



##### DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**G. PULLA REDDY ENGINEERING COLLEGE (AUTONOMOUS): KURNOOL**

**(Accredited by NBA of AICTE and NAAC of UGC with A grade)**

**(Affiliated to JNTUA, ANANTHAPURAMU)**

**2024-2025**

**G. PULLA REDDY ENGINEERING COLLEGE (Autonomous), Kurnool**

**(Accredited by NBA of AICTE and NAAC of UGC with A grade)**

**(Affiliated to JNTUA, Ananthapuramu)**

**Kurnool-518007**

##### DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

****

**CERTIFICATE**

**This is to certify that the major project work entitled**

**Design and Implementation of VGA Controller**

**is the bonafide record of work carried out by**

|  |  |
| --- | --- |
| **Sukamanchi Siva Chaitanya** | **(219X1A04J0)** |
| **Orjineni Veeresh** | **(219X1A04K1)** |
| **Dasa Sai Kiran** | **(219X1A04A9)** |

**Under my guidance and supervision in fulfillment of the requirements for the award of degree**

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**PROJECT GUIDE HEAD OF THE DEPARTMENT**

###### Smt. D. Rohini, M. Tech, Dr. K. SURESH REDDY, M. Tech., Ph. D,

Assistant Professor, Professor and Head of the Department,

Department of ECE, Department of ECE,

G. Pulla Reddy Engineering College (Autonomous), Kurnool.

G. Pulla Reddy Engineering College (Autonomous), Kurnool.

**ACKNOWLEDGEMENT**

We express our sincere thanks to our principal Dr. **B. Sreenivasa Reddy** garu, for providing for the facilities extended to work on the project during the project sessions.

We would like to express our sincere thanks to **Dr. K. Suresh Reddy** garu, Head of the Electronics and Communication Engineering Department, G. Pulla Reddy Engineering College for providing requisite facilities and helping us providing such a good environment.

We are extremely grateful to our project guide **Smt. D. Rohini** garu**,** Assistant Professor, ECE Department, G. Pulla Reddy Engineering College, who has been a source of inspiration throughout the course and for extending all support to us in the form of the technical literature and excellent guidance.

We also extend our sincere thanks to entire faculty and staff members of ECE Department, who have been a source of information throughout the course and for extending all support to us in the form of technical literature and excellent guidance.

**DECLARATION**

We here by declare that the major project titled **“Design and Implementation of VGA Controller”** is an authenticated work carried out by us as the students of **G. PULLA REDDY ENGINEERING COLLEGE**(Autonomous), Kurnool, during 2023- 2024 and has not been submitted for award of any degree or diploma in part or in full to any institute**.**

**Sukamanchi Siva Chaitanya**

**(219X1A04J0)**

**Orjineni Veeresh**

**(219X1A04K1)**

**Dasa Sai Kiran**

**(219X1A04A9)**

**ABSTRACT**

VGA(Video Graphics Array)has been widely used as a standard display interface. The Proposed method is to design and implementation of VGA controller. The project has given its top layer module design and the function simulation. This controller is developed using Verilog HDL. The system can display various color strips and characters, with this proposed algorithm may demonstrate good performance through short processing times, low resource utilization, and minimal power. The design can speed up data processing, improve system reliability.

EDA Tool:

* Xilinx 2023.2

Applications:

* Embedded Systems
* FPGA Prototyping and Development
* Image processing

CHAPTER 1 1

INTRODUCTION

1.1 Background 2

1.2 Problem Statement 3

1.3 Main Objective 3

CHAPTER 2 4

LITERATURE REVIEW 4

2.1 Review of Related Literatures 5

2.2 Review of Exiting system 5

CHAPTER 3 8

PROJECT METHODOLOGY 8

3.1 Block Diagram 9

3.2 Details of each component 9

CHAPTER 4 13

CONCEPTS OF VERILOG HDL 14

4.1 Importance of Verilog HDL 14

4.2 Concept of FPGA Implementation using Verilog HDL 15

4.3 FPGAs Architecture 17

4.4 Components of an FPGA 19

4.5 FPGA Programming 22

4.6 Auxiliary On-Chip Components 23

CHAPTER 5 26

CHAPTER 6 35

SOFTWARE TOOL 35

CHAPTER 7

RESULTS 47

CHAPTER 8 51

ADVANTAGES AND APPLICATIONS 51

8.1 Advantages 52

8.2 Applications 53

CHAPTER 9 54

CONCLUSION AND FUTURE SCOPE 54

9.1 Conclusion 54

9.2 Future Scope 55

REFERENCES 56

REFERENCES 57

##### LIST OF FIGURES PAGE NO

Figure 3.1 Block diagram of VGA Controller in Xilinx Vivado 9

Figure 3.2 Detailed Design of VGA Controller Module 11

Figure 4.1 Architecture of FPGA 17

Figure 4.2 Logic Block 19

Figure 4.3 Routing Architecture 20

Figure 5.1 Zed Board 23

Figure 5.2 Zed Board Block Diagram 27

Figure 6.1 Vivado Startup Window 34

Figure 6.2 Create Project Dialog 34

Figure 6.3 Enter Project Name 35

Figure 6.4 Select ZYNQ 7000 Board 36

Figure 6.5 Create Project Summary 36

Figure 6.6 Vivado Project Window 37

Figure 6.7 Design Source 38

Figure 6.8 Creation of Design Files 39

Figure 6.9 Add the Constraint File 40

Figure 6.10 Constraints Files 40

Figure 6.11 Flow of Synthesis 41

Figure 6.12 Monitor the Synthesis Log 42

Figure 6.13 Flow of Implementation 42

Figure 6.14 Implementing and Monitoring 43

Figure 6.15 Generating the Bit Stream 43

Figure 6.16 Powered on Black Board 44

Figure 7.1 Simulation Waveforms 48

Figure 7.2 Schematic 48

Figure 7.3 : Final Power 49

Figure 7.4 : Timing Analysis Report 49

Figure 7.5 : Utilization Report 50

# CHAPTER 1

# INTRODUCTION

### INTRODUCTION

A VGA (Video Graphics Array) controller is a crucial component in computer graphics systems, responsible for generating video signals that drive display monitors. It acts as an interface between the graphical data stored in memory and the display screen, ensuring the correct timing and synchronization of pixel information. VGA controllers are widely used in embedded systems, gaming consoles, and FPGA-based display applications due to their ability to manage screen resolution, color depth, and refresh rates efficiently.

In digital display systems, the VGA controller generates horizontal and vertical synchronization signals, controls the pixel clock, and processes video data for rendering on a screen. The primary challenge in designing a VGA controller is ensuring precise timing to maintain a stable and flicker-free display.

This project focuses on the **design and simulation** of a VGA controller using **Xilinx Vivado**. The design involves implementing the essential components, such as the synchronization module, video memory interface, and pixel generation logic. The FPGA-based simulation approach offers advantages such as high-speed operation, modular design flexibility, and real-time performance analysis. The implementation is validated through software simulation to verify signal accuracy, pixel positioning, and color rendering.

By leveraging **Xilinx Vivado's** simulation tools, this project provides an in-depth understanding of VGA signal generation and display control techniques, making it a valuable learning experience in digital system design and computer graphics applications.

##### BACKGROUND:

The design and implementation of VGA controllers have gained significant importance in modern digital display systems. VGA controllers are essential for generating video signals, managing display synchronization, and ensuring smooth rendering of graphical content. These controllers are widely used in embedded systems, computer graphics, and real-time visualization applications. The accuracy of signal generation and timing synchronization directly impacts the performance and stability of the display output.

Traditional display controllers rely on dedicated hardware components or microcontrollers, which may introduce limitations in terms of flexibility and scalability. To address these challenges, FPGA-based VGA controllers have emerged as an efficient solution. FPGA implementations allow for high-speed processing, precise control over video timing signals, and adaptability to different screen resolutions and display formats.

The development of digital systems using **Hardware Description Languages (HDLs)**, such as **Verilog and VHDL**, has revolutionized display controller design. Verilog HDL, introduced by **Phil Moorby and Prabhu Goel in 1984**, has become a widely used language for modeling and synthesizing digital circuits. **Xilinx Vivado**, a powerful FPGA design suite, provides an integrated environment for simulating, debugging, and verifying VGA controller designs before hardware implementation.

This project focuses on the **design and simulation** of a VGA controller using **Xilinx Vivado**. By leveraging FPGA-based simulation tools, the project ensures precise signal generation, pixel rendering, and synchronization techniques, making it a valuable study in the field of digital display system design.

##### PROBLEM STATEMENT

Design and simulate a **VGA (Video Graphics Array) Controller** capable of generating synchronization signals and controlling pixel data for display applications. The controller should efficiently handle **color strip and character rendering**, ensuring accurate timing and display stability. The design will be developed using **Verilog HDL** and simulated in **Xilinx Vivado** for functional verification. The implementation aims to demonstrate **optimized processing speed, low resource utilization, and improved system reliability**, making it suitable for real-time digital display applications.

##### MAIN OBJECTIVES

The main objective of this project, **"Design and Simulation of a VGA Controller in Xilinx Vivado,"** is to develop an efficient VGA controller that generates synchronization signals and manages pixel data for display applications. The project involves two key phases: first, the **Verilog HDL** design is simulated in **Xilinx Vivado** to verify functional correctness, signal timing, and waveform outputs. The second phase focuses on evaluating the **processing speed, resource utilization, and system reliability** through simulation results, ensuring optimal performance for digital display applications.

# CHAPTER 2

# LITERATURE REVIEW

### LITERATURE REVIEW

##### REVIEW OF RELATED LITERARURE

#### ****"Design and Implementation of VGA Controller Using FPGA" (IJACT, September 2012)****

**Authors:** Fangqin Ying, Xiaoqing Feng

This paper discusses a structured approach to designing a **VGA controller** on an **Altera FPGA** using **VHDL** as the hardware description language and **Quartus II** for synthesis and implementation. The study highlights key design components, including **synchronization signal generation, color strip rendering, character display, and image processing**. The authors emphasize the importance of **low resource utilization and high processing speed**, making FPGA-based VGA controllers a viable solution for embedded display applications. The paper also explores various optimizations in **timing constraints, memory access, and pixel processing**, ensuring smooth display rendering with minimal hardware overhead.

This research is relevant to our project as it provides insights into **efficient VGA controller design** methodologies, helping us refine our approach using **Xilinx Vivado and Verilog HDL** for simulation and functional verification.

### ****2.1.2 "Design of VGA Display Controller Based on FPGA and VHDL" (IEEE, 2011)****

**Authors:** Hongyan Dong, Hongmin Guo

This paper explores the design and implementation of a **VGA display controller** using **FPGA technology and VHDL programming**, with a focus on optimizing **video data handling** within a single FPGA chip. The authors utilize the **ACEX1K FPGA chip** and **MAX+plus II** software for synthesis and simulation. Their design methodology aims to enhance **performance efficiency** in terms of **processing speed and resource utilization**, making it suitable for real-time display applications.

By streamlining video data management within a single FPGA, the study demonstrates an efficient approach to **display signal processing, synchronization, and pixel rendering**. The research findings provide valuable insights into optimizing FPGA-based **VGA controllers** for **high-speed and low-power** applications.

This paper serves as a significant reference for our project, as it highlights the **FPGA-based VGA controller design flow**, which aligns with our objective of designing and simulating a VGA controller in **Xilinx Vivado** using **Verilog HDL**.

### ****2.1.3. "Design and Implementation of VGA Controller on FPGA" (2015)****

**Authors:** Renuka A. Wasu, Vijay R. Wadhankar

This paper presents a structured approach to designing and implementing a **VGA controller** using **FPGA technology**. The authors employ **Verilog HDL** for hardware description and **Altera Quartus II** for design, simulation, and implementation. The VGA controller is developed for the **DE2-115 FPGA development board**, featuring key functional modules such as **VGA synchronization, address generation, and image data processing**. The design is optimized for a **640×480 resolution** with a pixel clock of **25.175 MHz**, ensuring a stable display output.

The study focuses on the precise **timing and synchronization** required for VGA signal generation while minimizing **hardware resource consumption**. The results demonstrate an efficient approach to achieving **real-time image rendering** using FPGA-based hardware acceleration.

This research provides a useful reference for our project, as it aligns with our focus on **designing and simulating a VGA controller**. Although our project utilizes **Xilinx Vivado** instead of **Altera Quartus II**, the insights into **synchronization techniques and resolution handling** are valuable for optimizing our design for accurate **display signal generation**.

* 1. **Review of Existing System**

The VGA (Video Graphics Array) controller is an essential component in digital display systems, responsible for generating synchronization signals and managing pixel data to render images on a screen. Traditional VGA controller implementations rely on microcontrollers, GPUs, or dedicated hardware circuits, but FPGA-based designs provide greater flexibility, faster processing, and optimized resource utilization. This review explores various VGA controller design methodologies and their implementation strategies.

* + 1. **Microcontroller-Based VGA Controller:**
* Uses a microcontroller to generate VGA signals through software-driven timing loops.
* Limited by processor speed and computational capacity, making it unsuitable for high-resolution displays.
* Often requires external RAM and additional circuitry for framebuffer storage.
  + 1. **GPU-Based VGA Controller:**
* Employs a dedicated graphics processing unit (GPU) to handle display rendering.
* Provides high-quality image processing but requires complex driver software and firmware development.
* Not cost-effective for embedded applications due to high power consumption and hardware complexity.
  + 1. **FPGA-Based VGA Controller:**
* Uses hardware description languages (HDLs) like Verilog or VHDL to generate synchronization signals and control pixel data.
* Offers high-speed parallel processing, enabling efficient video signal generation.
* Supports real-time display modifications with low resource utilization and reduced power consumption.
  + 1. **SRAM-Based VGA Controller:**
* Stores pixel data in Static RAM (SRAM), allowing direct access for fast display updates.
* Provides smooth video rendering but requires large memory space for high-resolution displays.
* Suitable for applications that require fixed image rendering rather than dynamic graphics generation.

# CHAPTER 3

# PROJECT METHODOLOGY

## PROJECT METHODOLOGY

## Block Diagram





###### Fig 3.1 Block diagram of VGA Controller in Xilinx Vivado

###### Block Diagram Description

###### The block diagram represents a VGA display system implemented on an FPGA chip. An External Data Generator supplies input data, such as pixel or image information, which is first processed by the File Operations module. This module formats and transfers the data into the Block RAM, a memory block within the FPGA used to store display data. The VGA Display Controller reads the data from Block RAM and converts it into RGB video signals along with synchronization signals, Horizontal Sync (HS) and Vertical Sync (VS). These signals are required to properly display images on a VGA Monitor. The entire system is driven by a clock (clk) signal and includes a reset input for initialization or restart. The Block RAM allows temporary and efficient storage of video frames. The design ensures the VGA controller receives data in sync with the video timing. This setup enables real-time image display on a VGA screen using FPGA resources.

###### Details of Each Block:

### ****1. Input Signals****

The input section includes essential signals required to operate the display system. **CLK (Clock)**synchronizes all operations across modules, ensuring timing accuracy. The **Reset** signal initializes or resets the system to its default state. **Vector-x** and **Vector-y** represent the current horizontal and vertical pixel coordinates, respectively, and are critical for calculating display content per pixel. The **HS (Horizontal Sync)** and **VS (Vertical Sync)** signals are synchronization pulses required for generating VGA timing to properly place the image on the monitor.

**2.Color Strip Generator**

The **Color Strip** block generates color pattern outputs, usually in vertical or horizontal stripes. It uses the incoming pixel coordinates (vector-x, vector-y) to determine which color should be shown at each location. This module is often used for testing and verifying VGA output functionality, as the color patterns can easily indicate issues in display timing or data path alignment.

**3. Character Generator**

The **Character** block is responsible for rendering text or alphanumeric characters on the VGA screen. It interprets pixel positions using vector-x and vector-y to select the appropriate character from a font table or character memory. This module can be used to display messages, labels, or status information on the screen, making it useful in user interfaces or debug modes.

**4. Image Generator**

The **Image** block outputs pixel data corresponding to a predefined image. It accesses image data from memory or internal ROM using vector coordinates to map pixel colors correctly. This block allows static or dynamic image rendering, depending on whether the image content is fixed or updated during runtime.

**5. 4-to-1 Multiplexer (MUX)**

The **4 of 1 MUX** selects one of the three video outputs (Color Strip, Character, or Image) based on the mode control signals (**md2, md1, md0**). Although only three display blocks are used, the 4-input MUX allows for future expansion or default modes. The MUX forwards the selected RGB pixel data to the VGA output, ensuring only one display mode is active at a time.

**6. Output RGB Signals**

The final **RGB output** (Red, Green, Blue) from the MUX is sent to the VGA monitor. These signals determine the color intensity of each pixel on the display. Combined with the synchronization signals (HS and VS), the RGB data completes the video signal required for displaying visuals on a VGA screen.

A computer screen with different colored squares

Description automatically generated with medium confidence

**Fig 3.2 Detailed Design of VGA Controller Module**

**1. Clock Divider**

The **Clock Divider** takes the high-frequency system clock (clk) and reduces its frequency to a suitable level for VGA timing using the clkdiv output. This is essential because VGA monitors operate at a specific pixel clock frequency (e.g., 25 MHz for 640x480 resolution). The divided clock drives the horizontal and vertical counters for synchronized timing.

**2. Horizontal Counter**

The **Horizontal Counter** counts the number of clock cycles corresponding to each horizontal scan line. It resets after completing one full line (including visible area, front porch, sync pulse, and

back porch). The output Hcnt represents the current horizontal pixel position, and the terminal

count (TC) indicates the end of a line, enabling the vertical counter.

**3. Vertical Counter**

The **Vertical Counter** increments once per horizontal line (enabled by TC from the horizontal counter). It keeps track of the current vertical position on the screen, including visible rows and vertical sync periods. When it reaches the end of a full frame, it resets to begin a new frame cycle.

**4. Comparators**

There are three **Comparators** used for decision-making. The first comparator checks if the horizontal counter is within the **horizontal sync time** and generates the **HS (Horizontal Sync)** signal. The second comparator determines if both Hcnt and Vcnt are within the **display area**, controlling whether the screen should show pixel data or remain blank. The third comparator checks for **vertical sync timing.**

**5. Multiplexer (MUX)**

The **MUX** selects between two outputs: either a pixel color value (Some Color) when inside the visible display area or a blanking signal (all zeros) otherwise. The selection is based on the result from the "At Display Area?" comparator. The final output goes to the **Red, Green, and Blue (RGB)** VGA signals, controlling the color shown on the monitor.

# CHAPTER 4

**CONCEPTS OF VERILOG HDL & FPGA**

### CONCEPTS OF VERILOG HDL

Verilog is a hardware description language used to design digital logic. It is mostly used in designing [digital circuits](https://en.wikipedia.org/wiki/Digital_electronics) at the [register-transfer level.](https://en.wikipedia.org/wiki/Register-transfer_level) This language can also be used to design Analog circuits and mixed logic circuits i.e., combination of analog and digital logic.

Verilog HDL is a hardware description language, unlike other high-level software programming languages the sole purpose of coding in Verilog HDL is to generate the desired circuit, whether it is a simple logic gate IC or a complex CPU design.

The syntax of the Verilog HDL is similar to the C programming language. For instance, the concept of looping, creating and calling user-defined functions (modules in case of Verilog HDL), data types, etc. are very synonymous amongst these two languages.

HDLs are specialized computer languages used to program electronic and digital logic circuits. High level languages with which we can specify our hardware to analyze its design before actual fabrication.

###### Different Levels of Abstraction in Verilog HDL:

* + - 1. Gate-Level modelling
      2. Dataflow modelling
      3. Behavioral modelling

###### IMPORTANCE OF Verilog HDL:

**Mixed Abstraction Levels**:

Verilog allows different levels of abstraction to co-exist within the same model. Designers can describe hardware using switches, gates, RTL (Register Transfer Level), or behavioral code.

###### Unified Language for Design and Stimulus:

Verilog allows both stimulus generation and hierarchical design. Designers can express their ideas consistently, whether specifying behavior or defining the hardware Structure.

###### Automation and Conversion:

Verilog enhances the design process by allowing engineers to describe desired hardware functionality. Automation tools then convert this behavioral description into actual hardware elements (combinational gates, sequential logic, etc.). It bridges the gap between high-level design into low-level implementation.

###### Simulation Acceleration and Time-to-Market:

Verilog accelerates simulation, reducing the time required for design validation. Faster simulations lead to quicker iterations and faster product development. Ultimately, this contributes to shorter time-to-market for electronic products.

##### CONCEPT OF FPGA IMPLEMENTATION USING VERILOG HDL:

###### FPGA Basics:

An FPGA is a reconfigurable integrated circuit that allows designers to create custom digital logic circuits. Unlike fixed-function ASICs (Application-Specific Integrated Circuits), FPGAs can be programmed and reprogrammed after manufacturing. Verilog HDL (Hardware Description Language) is commonly used for designing and describing the behavior of digital circuits within FPGAs.

###### Verilog HDL and FPGA Design:

**Logic Synthesis**:

Logic synthesis translates high-level descriptions (written in Verilog) into a netlist of gates and flip-flops. It optimizes the design for area, speed, or power, considering the specific FPGA architecture.

###### Architecture-Specific Optimization:

FPGAs have unique architectures with configurable logic blocks (CLBs), routing resources, and I/O blocks. Verilog designs can be optimized specifically for these FPGA architectures. Register transfer level (RTL) optimizations and logic-level optimizations are crucial.

###### Hierarchical Design:

Verilog allows hierarchical design, where modules can be instantiated within other modules. This modularity simplifies complex designs and promotes reusability.

###### Behavioral Modelling:

Verilog enables behavioral modeling, where we describe the intended functionality of a module. Behavioral code focuses on what the design should do, rather than how it should be implemented.

###### Applications of FPGA Implementation:

FPGAs find applications in various domains:

1. **Digital Signal Processing (DSP)**: Implementing filters, transforms, and modulation/demodulation schemes.
2. **Embedded Systems**: Creating custom processors, memory controllers, and peripheral interfaces.
3. **High-Performance Computing**: Accelerating specific algorithms or computations.
4. **Prototyping and Rapid Development**: Quickly testing and iterating hardware designs.

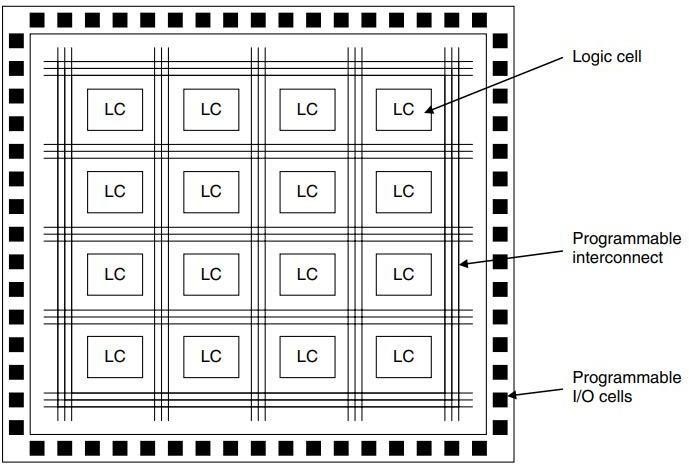
###### Field Programmable Gate Arrays (FPGAs)

Field Programmable Gate Arrays (FPGAs) are digital ICs (Integrated Circuits) that enable the hardware design engineer to program a customized Digital Logic as per his/her requirements. The term “Field Programmable” implies that the Digital Logic of the IC is not fixed during its manufacturing (or fabrication) but rather it is programmed by the end-user (designer). In order to provide this programmability, an FPGA consists of Configurable (or Programmable) Logic Blocks and configurable interconnects between these blocks. This configurable Logic and the Interconnections (Routing) of FPGAs makes them general purpose and flexible but at the same time, it also makes them slow and power hungry when compared to a similar caliber ASIC with Standard Cells.

###### FPGAs Architecture:

FPGA from Xilinx are hybrid computation systems with Block RAMs, programmable fabric, DSP Slices, and PCI Express support. Just Because all of these compute resources can be accessed at the same time, they enable scalability and pipelining of applications throughout the entire platform. The architecture of an FPGA is completely different as it consists of programmable Logic Cells, programmable interconnects and programmable IO blocks.

Field Programmable Gate Arrays or FPGAs in short are pre-fabricated Silicon devices that consists of a matrix of reconfigurable logic circuitry and programmable interconnects arranged in a two- dimensional array. The programmable Logic Cells can be conFigured to perform any digital function and the programmable interconnects (or switches) provide the connections among different logic cells.



###### Fig 4.1: Architecture of FPGA

Using an FPGA, we can implement any custom design by specifying the logic or function of each logic block and setting the connection of each programmable switch. Since this process of designing a custom circuit is done in the field rather than in a fab, the device is known as “Field Programmable”.

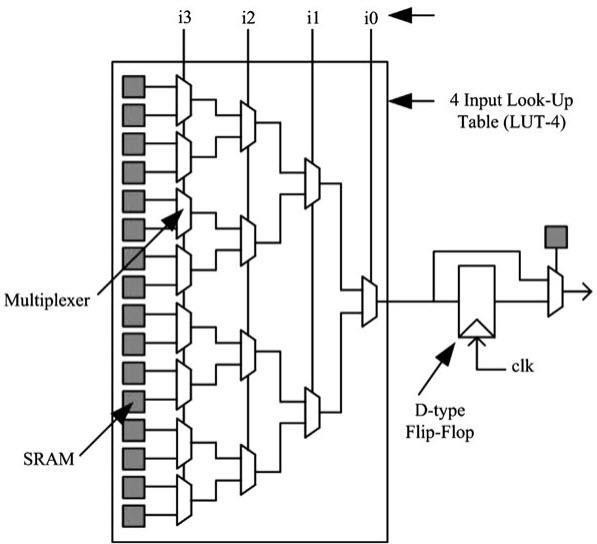
###### Components of an FPGA:

FPGA consists of three basic components They are:

* + 1. Programmable Logic Cells (or Logic Blocks) – responsible for implementing the core logic functions.
    2. Programmable Routing – responsible for connecting the Logic Blocks.
    3. IO Blocks – which are connected to the Logic Blocks through the routing and help to make external connections.

###### Logic Block:

The Logic Block in Xilinx based FPGAs are called as Configurable Logic Blocks or CLB while the similar structures in Altera based FPGAs are called Logic Array Blocks or LAB. Let us use the term CLB for this discussion. A CLB is the basic component of an FPGA, which provides boththe logic and storage functionalities. The basic logic block can be anything like a transistor, a NAND gate, Multiplexors, Look-up Table (LUT), a PAL-Like structure or even a processor. BothXilinx and Altera use Look-up Table (LUT) based logic blocks to implement the logic as well as the storage functionalities.



###### Fig 4.2: Logic block

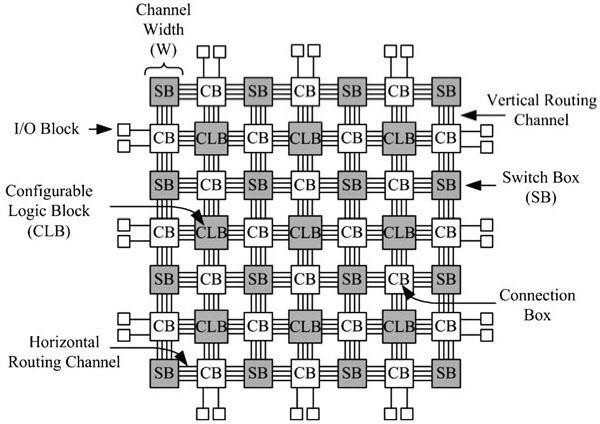
**Routing:**

If the computational functionality is provided by the Logic Blocks, then the programmable routing network is responsible for interconnection these logic blocks. The Routing Network provides interconnections between one logic block to other as well as between the logic block and the IO Block to completely implement a custom circuit.

Basically, the routing network consists of connecting wires with programmable switches, which can be configured using any of the programming technologies. There are basically two types of routing architectures. They are:

* + 1. Island Style Routing (also known as Mesh Routing)
    2. Hierarchical Routing

In island style routing architecture, the logic blocks are arranged in a two-dimensional array and are interconnected using a programmable routing network. This type of routing is widely used in commercial FPGAs.



###### Fig 4.3: Routing Architecture

The main features of modern FPGA structure are combined with extra computational and data storage blocks to enhance the device’s computational density and efficiency.

* + - 1. Embedded memories for distributed data storage
      2. PLLs or Phase-locked loops for driving the FPGA fabric at different clock rates
      3. High-speed serial transceivers
      4. Off-chip memory controllers
      5. Multiply-accumulate block

###### FPGA Programming:

To illustrate a typical FPGA design flow, we will examine the tools of the Xilinx tool chain, as well as their intermediate circuit representations. The most important steps and tools of the design flow to produce an FPGA-circuit are depicted. Programming on FPGA is same as connecting wires in circuit this is done by using a hardware description language (HDL) such as VHDL or Verilog. Synthesizer converts HDL into gate level netlist, native generic circuit (NGC) format mapped to technology library provided by Xilinx.

###### Translate

Tool ngbuild merge and translate all input netlists and constrain into a single netlist and save it as native generic database (NGD) file. User constrain file (UCF) is specify by the FPGA designer at time of manufacturing constrained are used for assigning special physical element of FPGA (e.g.: I/O pin, clock etc.) in the design as well as timing require for design. NGC netlist is based on UNISIM library whereas NGD netlist based on SIMPRIM library. NGC allow behavioral simulation and NGD allow timing simulation.

###### Map

The map tool maps the SIMPRIM primitives in an NGD netlist to specific device resources such as logic islands, I/O blocks, etc. The map tool then generates a native circuit description (NCD) file that describes the circuit, now mapped to physical FPGA components.

###### Place and Route

The par tool performs placement and routing. The physical element specified in NCD file is specify at particular location on FPGA and interconnected. Place and route are most time-

consuming process in design flow it is based on simulated annealing algorithms. The par tool takes the mapped NCD file and generates routed NCD files which also contain the routing information. **Bitstream Generation**

The implemented design has to dump on FPGA readable format. This is done by the bitgen tool, it encodes design in binary known as Bitstream. Then Bitstream is loaded on FPGA using JTAG cable. Inside FPGA a finite state machine control by Bitstream which extract configuration data from Bitstream.

###### Auxiliary On-Chip Components

The logic resources of FPGAs discussed so far are in principle sufficient to implement a wide range of circuits. However, to address high-performance and usability needs of some applications, FPGA vendors additionally intersperse FPGAs with special silicon components, such as dedicated RAM blocks (BRAM), multipliers and adders (DSP units), and in some cases even full-edged CPU cores. Hence, [HGV+08] observed that the model for FPGAs has evolved from a bag of gates to a bag of computer parts.

###### Block RAM (BRAM)

The BRAM is a configurable module, which is attached with variety of BRAM interface controllers.

###### Main features

1. Fully automated generation and configuration of HDL through EDK Platgen/Simgen tools.
2. Number of BRAM primitives utilized is a function of the configuration parameters for: memory address range, number of byte-write enables, the data width, and the targeted architecture.
3. Both Port A and Port B of the memory block can be connected to independent BRAM
4. Interface Controllers: LMB (Local Memory Bus), OPB (On-chip Peripheral Bus), PLB (Processor Local Bus), and OCM (On-Chip Memory).
5. Supports byte, half-word, word, and double word transfers provided the correct number of byte-write enables have been configured.

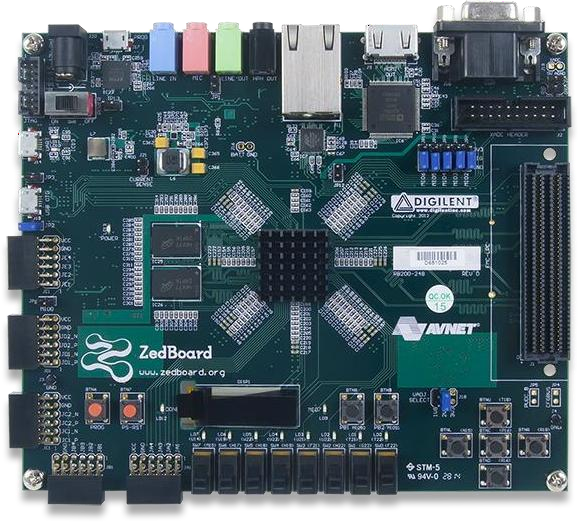
###### Digital Signal Processing (DSP) Units

FPGA is very widely used for digital signal processing (filter design, Fourier transform, convolution etc.), these all are mathematical based expression. Large number of adder and multiplier is incorporated in FPGA (hundred too few thousand). Adder and multiplier can be realizing using LUTs and input-output in FPGA. These all are power efficient high performance and minimum space circuit. Similar to other components of FPGA, the digital signal processing unit also needed and combine with the adjacent DSP section. For example, we consider Xilinx DSP$\*E slice which has three input ports of different width (25bits,18 bits and 48 bits) and it provide 25\*18bit multiplier with a pipelined second stage that can perform as subtractions or adder (48 bit) also with optimal accumulation feedback. The DSP unit of FPGA is used for various application like multiply, multiply and accumulate, multiply and add/subtract, three input addition, barrel shifting, wide bus multiplexing etc. are various operation which can perform on FPGA. It can also be used in different modes. These all function can be implemented in one or two clock cycle one over a one wide input.

# CHAPTER 5

**DESCRIPTION OF ON BOARD COMPONENTS**

###### ZED Board (Zynq Evaluation and Development Board)

****

**Fig 5.1 Zed board**

The features provided by the Zed Board consist of:

* Xilinx XC7Z020-1CLG484C Zynq-7000 AP SoC
  + Primary configuration = QSPI Flash
  + Auxiliary configuration options
    1. Cascaded JTAG
    2. SD Card

#### Memory

* + 512 MB DDR3 (128M x 32)
  + 256 Mb QSPI Flash

###### Interfaces

* USB-JTAG Programming using Diligent SMT1-equivalent circuit
  1. Accesses PL JTAG
  2. PS JTAG pins connected through PS Pmod
* 10/100/1G Ethernet
* USB OTG 2.0
* SD Card
* USB 2.0 FS USB-UART bridge
* Five Diligent Pmod™ compatible headers (2x6) (1 PS, 4 PL)
* One LPC FMC
* One AMS Header
* Two Reset Buttons (1 PS, 1 PL)
* Seven Push Buttons (2 PS, 5 PL)
* Eight dip/slide switches (PL)
* Nine User LEDs (1 PS, 8 PL)
* DONE LED (PL)

###### On-board Oscillators

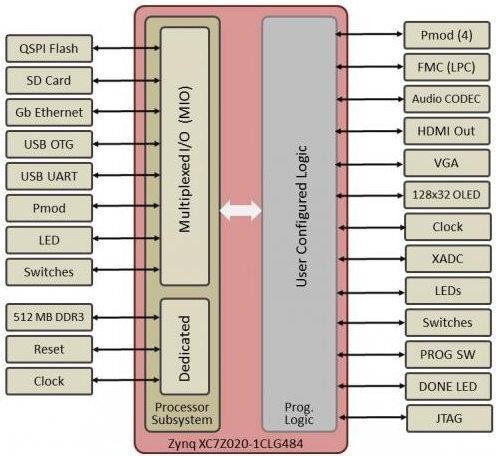
* 33.333 MHz (PS)
* 100 MHz (PL)

###### Display/Audio

* HDMI Output
* VGA (12-bit Color)
* 128x32 OLED Display

###### Audio Line-in, Line-out, headphone Power

* On/Off Switch
* 12V @ 5A AC/DC regulator



###### Fig 5.2 Zed Board Block diagram

* + 1. **Clock sources:**

The EPP’s PS subsystem uses a dedicated 33.3333 MHz clock source, IC18, Fox 767- 33.333333- 12, with series termination. The PS infrastructure can generate up to four PLL-based clocks for the PL system. An on-board 100 MHz oscillator, IC17, Fox 767-100-136, supplies the PL subsystem clock input on bank 13, pin Y9.

###### Reset Sources

* + - * **Power‐on Reset (PS\_POR\_B)**: The Zynq PS supports external power-on reset signals. The power-on reset is the master reset of the entire chip. This signal resets every register in the device capable of being reset. Zed Board drives this signal from a comparator that holds the system in reset until all power supplies are valid. Several other ICs on Zed Board are reset by this signal as well
      * **Program Push Button**: Switch A PROG push switch, BTN6, toggles EPP PROG\_B. This initiate reconfiguring the PL subsection by the processor.
      * **Processor Subsystem Reset**: Power-on reset, labeled PS\_RST/BTN7, erases all debug configurations. The external system reset allows the user to reset all of the functional logic within the device without disturbing the debug environment. For example, the previous break points set by the user remain valid after system reset. Due to security concerns, system reset erases all memory content within the PS, including the OCM. The PL is also reset in system reset. System reset does not re-sample the boot mode strapping pins.

#### User I/O

* + - * **User Push Buttons**: The Zed Board provides 7 user GPIO push buttons to the EPP; five on the PL-side and two on the PS-side. Pull-downs provide a known default state, pushing each button connects to Vcco.

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Subsection** | **Zynq pin** |
| BTNU | PL | T18 |
| BTNR  BTND | PL  PL | R18  R16 |
| BTNC | PL | P16 |
| BTNL | PL | N16 |
| PB1  PB2 | PS  PS | D13(MIO50)  C10(MIO51) |

###### Table 5.3 Push buttons connections

* + - * **User DIP Switches**: The Zed Board has eight user dip switches, SW0-SW7, providing user input. SPDT switches connect the I/O through a 10kΩ resistor to the VADJ voltage supply or GND.

|  |  |
| --- | --- |
| **Signal Name** | **Zynq pin** |
| SW0 | F22 |
| SW1 | G22 |
| SW2 | H22 |
| SW3 | F21 |
| SW4 SW5 SW6  SW7 | H19 H18 H17  M15 |
| **Table 5.1.3.2 DIP switch connects** | |

**User LEDs:** The Zed Board has eight user LEDs, LD0 – LD7. A logic high from the Zynq-7000 AP SoC I/O causes the LED to turn on. LEDs are sourced from 3.3V banks through 390Ω resistors.

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Subsection** | **Zynq pin** |
| LD0 | PL | T22 |
| LD1 LD2 LD3 LD4 LD5 LD6 LD7  LD9 | PL PL PL PL PL PL PL  PS | T21 U22 U21 V22 W22 U19 U14  D5(MIO7) |

**Table 5.4 LED connections**

#### Power

* + - * **Primary Power Input:** The board’s primary input is through a 12V barrel jack. A compatible AC/DC converter will have a 2.5mm inner diameter, 5.5mm outer diameter, center positive connection. The total power budget is based on 4A from an AC/DC wall wart supply. This rail is protected with a TE 0603SFF600F/24-2. A 10mΩ, 1W current sense resistor exists in series with the 12V input power supply. Header J21 straddles this resistor to measure the voltage across this resistor for calculating Zed Board power. Power is the measured voltage squared divided by the 10mΩ resistance.
      * **On/Off Switch**: Zed Board includes an on/Off switch, SW8, to gate 12V... When SW8 is in the OFF position, 12V is disconnected from the board.

##### LED:

A light-emitting diode (LED) is a semiconductor device that emits light when an electric current flows through it. When current passes through an LED, the electrons recombine with holes emitting light in the process. LEDs allow the current to flow in the forward direction and blocks the current in the reverse direction.

The Zed Board is a development board based on the Xilinx Zynq-7000 All Programmable SoC (System on Chip). It is designed for embedded systems development and prototyping. The Zynq- 7000 SoC combines a dual-core ARM Cortex-A9 processor with Xilinx's programmable logic (FPGA) on a single chip.

Here are some key features of the Zed Board:

* + - 1. **Zynq-7000 SoC**: The heart of the Zed Board is the Xilinx Zynq-7000 All Programmable SoC, which combines a dual-core ARM Cortex-A9 processor with programmable logic (FPGA).
      2. **Memory:** The board typically comes with DDR3 memory for the ARM processor subsystem and programmable logic, as well as other memory components like Flash memory for storing the bootloader and configuration bitstreams.
      3. **FPGA**: The programmable logic (FPGA) on the Zynq-7000 allows for customizable hardware acceleration and interfacing with external devices. Users can design and implement custom digital circuits to meet specific application requirements.
      4. **Peripherals:** The Zed Board includes a variety of peripherals, such as USB ports, Ethernet, HDMI, audio ports, and more, providing flexibility for a range of embedded system applications.
      5. **Connectivity:** It offers multiple connectivity options for interfacing with other devices and systems. This includes Ethernet for networking, USB for connecting peripherals, and various GPIO (General Purpose Input/Output) pins for interfacing with external hardware.
      6. **Expansion Headers:** The Zed Board has expansion headers, such as the Pmod connectors and FMC (FPGA Mezzanine Card) connector, allowing users to attach additional hardware modules and daughter cards for added functionality.
      7. **Development Tools**: Xilinx provides development tools, such as Vivado Design Suite, to program and configure the Zynq-7000 SoC. These tools enable users to design, simulate, implement, and debug their hardware and software.
      8. **Community and Documentation**: The Zed Board has an active user community, and there is documentation and example projects available to help developers get started with their projects.

The Zed Board is commonly used for a variety of applications, including but not limited to embedded systems, signal processing, image processing, and industrial automation. It serves as a valuable platform for learning and prototyping with Xilinx's Zynq-7000 SoC technology.

# CHAPTER 6

# SOFTWARE TOOL

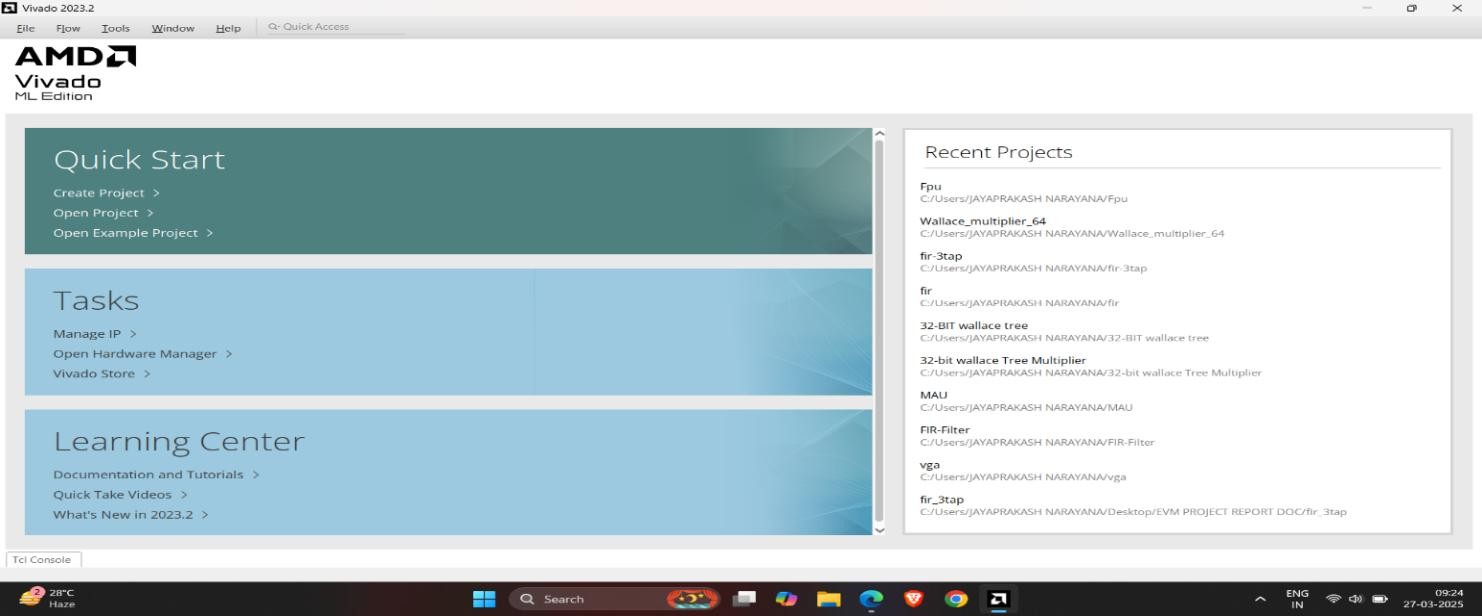
##### SOFTWARE TOOL

Vivado, developed by Xilinx, is a robust and integrated design environment for FPGA and SoC development. Serving as a comprehensive IDE, it facilitates hardware description, synthesis, implementation, and debugging. Supporting a broad range of Xilinx devices, Vivado enables designers to optimize their designs for specific FPGA and SoC architectures. With high-level design entry options such as Verilog, VHDL, and graphical block diagram entry through IP Integrator, it accommodates diverse design methodologies. The tool incorporates an extensive IP library, streamlining the integration of intellectual property blocks into designs to reduce development time. Vivado's synthesis and optimization capabilities enhance performance, area utilization, and power efficiency. Through its place-and-route tools, it maps designs onto FPGA resources, optimizing for timing and meeting specified constraints. The tool also features power analysis tools, aiding in the optimization of power consumption. With debugging tools, logic analyzers, and hardware co-simulation, Vivado facilitates efficient design verification and debugging. Furthermore, it supports advanced features like partial reconfiguration, allowing specific regions of an FPGA to be reprogrammed without affecting the entire device. Available on both Windows and Linux platforms, Vivado caters to a broad user base, regularly updating to incorporate new features and device support.

##### STEP 1: CREATE A VIVADO PROJECT

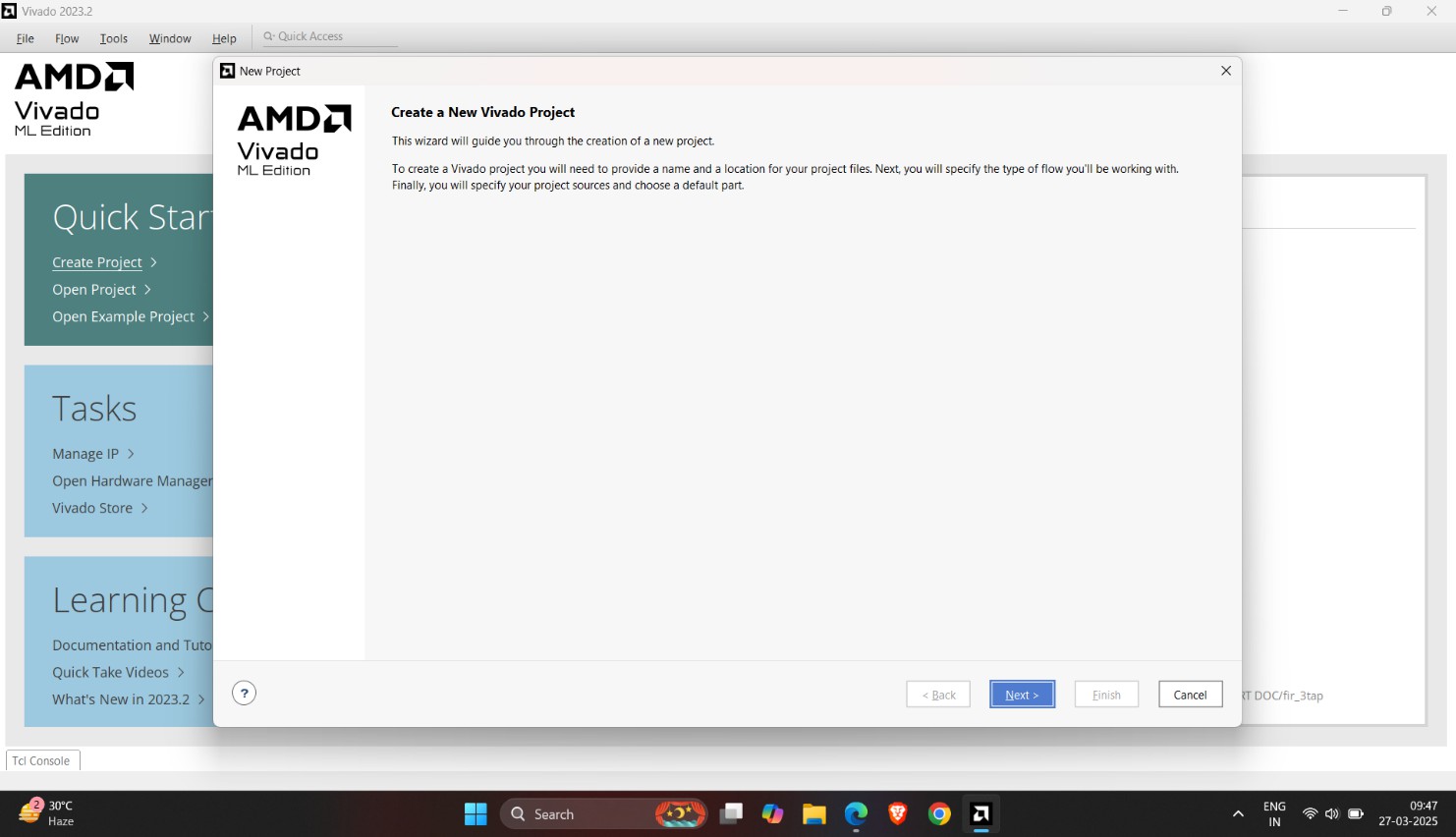
###### Start Vivado

In Windows, Vivado was started by clicking the shortcut on the desktop. After Vivado was started, the window should look similar to the picture in Figure 1.



###### Figure 6.1 Vivado Start-Up Window

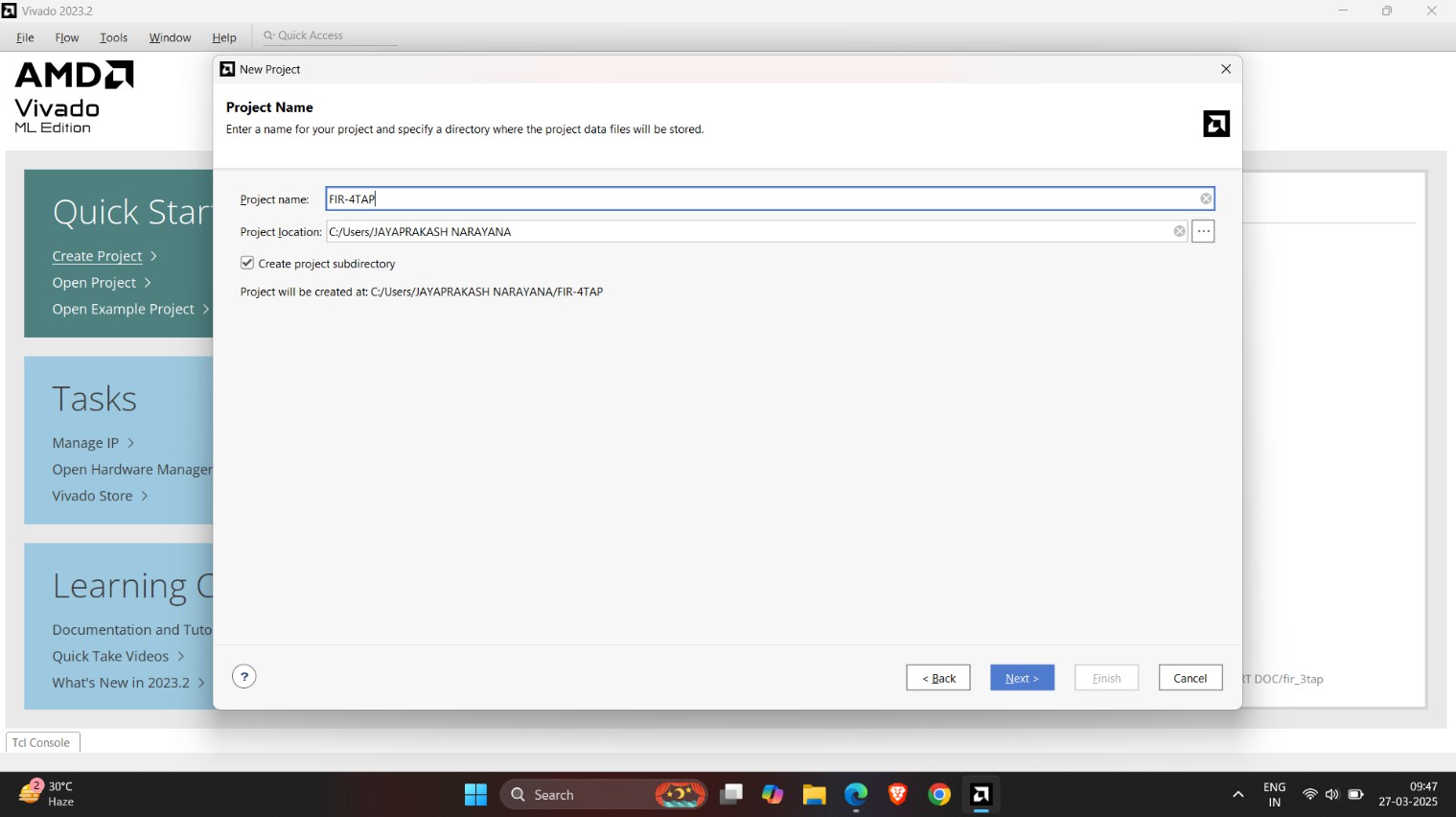
1. **Open Create Project Dialog**

Click on “Create Project” in the Quick Start panel. This will open the New Project dialog as shown in Figure 2. Click Next to continue.

###### Figure 6.2 Create Project Dialog

1. **Set Project Name and Location**

In this step our project name FIR-4TAP was given at the option of project name as shown in Figure.



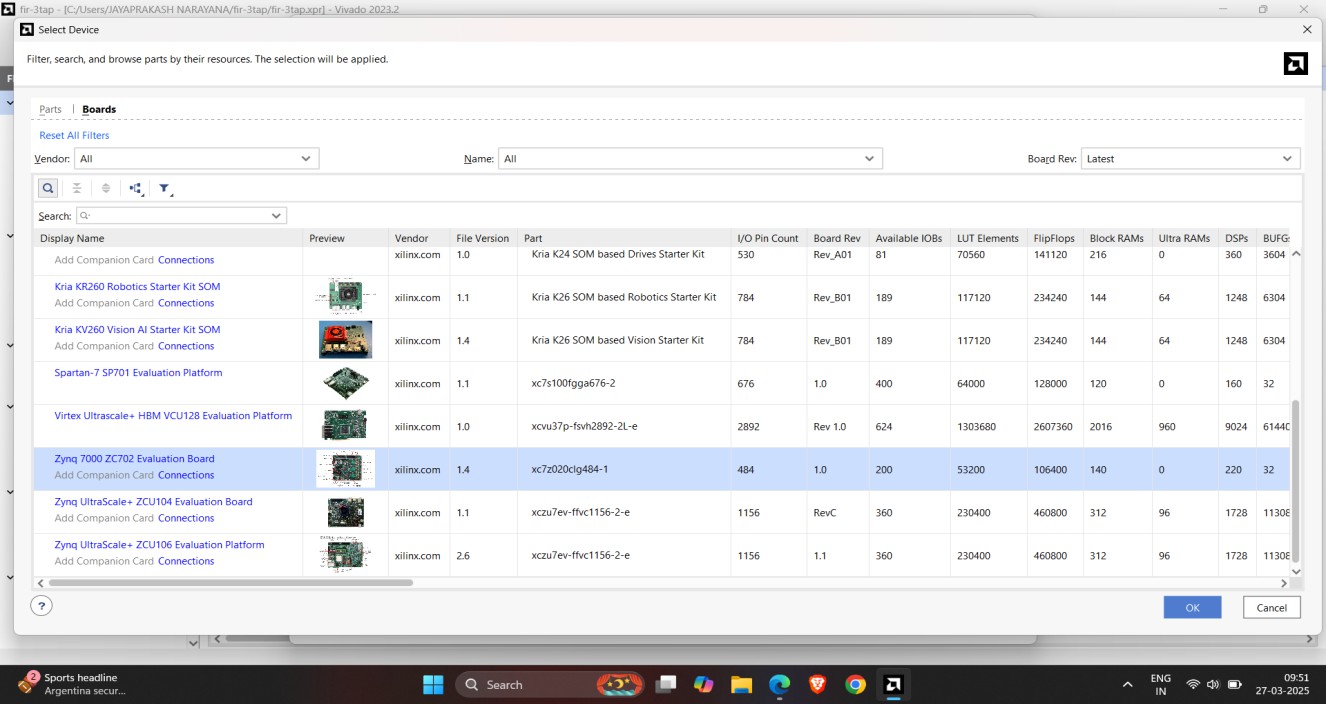
###### Figure 6.3. Enter Project Name

1. **Select Parts**

Xilinx produces many different parts, and the synthesizer needs to know exactly what part you are using so it can produce the correct programming file. For our project we selected the following parts of the FPGA to meet our requirements.

For example, the Blackboard uses a Zynq device with the following attributes:

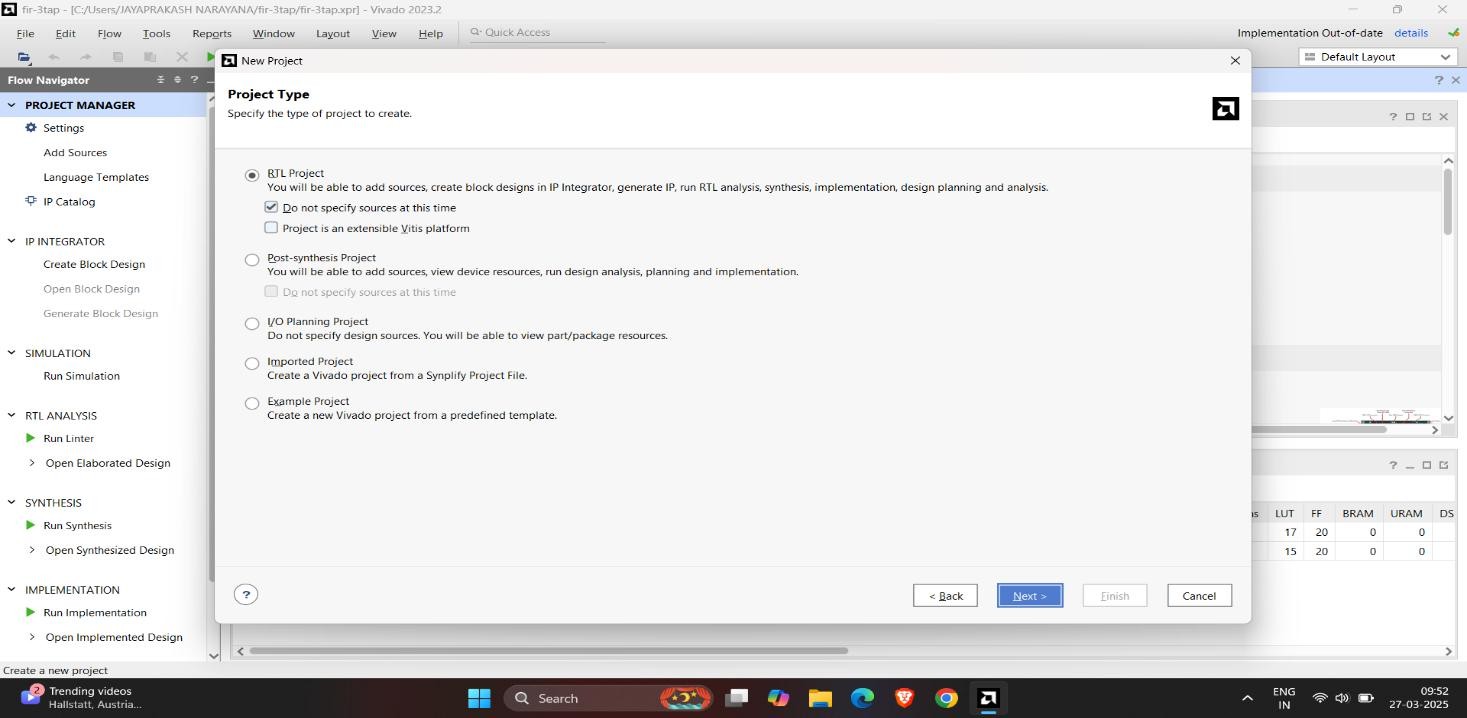
|  |  |
| --- | --- |
| **Part Number** | **xc7z060clg484** |
| **Family** | **Zynq-7000** |
| **Package** | **Clg324** |
| **Speed Grade** | **-1** |



###### Figure 6.4. Select Zed Board Evaluation and development kit

1. **Check Project Configuration Summary**

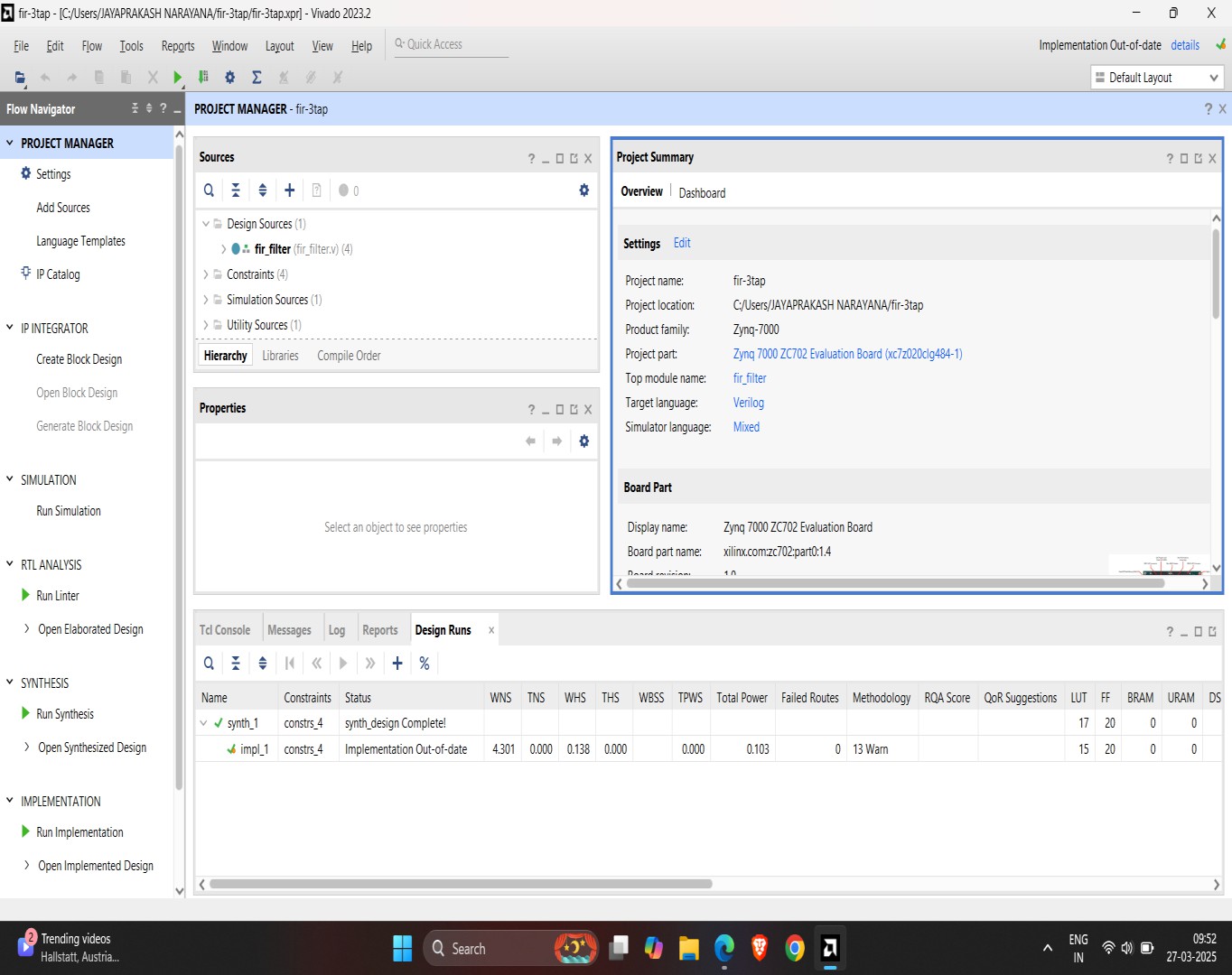
On the last page of the Create Project Wizard a summary of the project configuration is shown.We Verified all the information in the summary and made sure the correct FPGA was selected.



###### Figure 6.5. Create Project Summary

1. **Vivado Project Window**

After having finished with the Create Project Wizard, the main IDE window was displayed. This is the main “working” window where you entered and simulate our Verilog code,launch the synthesizer, and program on board. The left-most pane is the flow navigator that shows all the current files in the project, and the processes you can run on those files. To the right of the flow navigator is the project manager window where we enter source code, view simulation data, and interact with our design. The console window across the bottom shows a running status log.



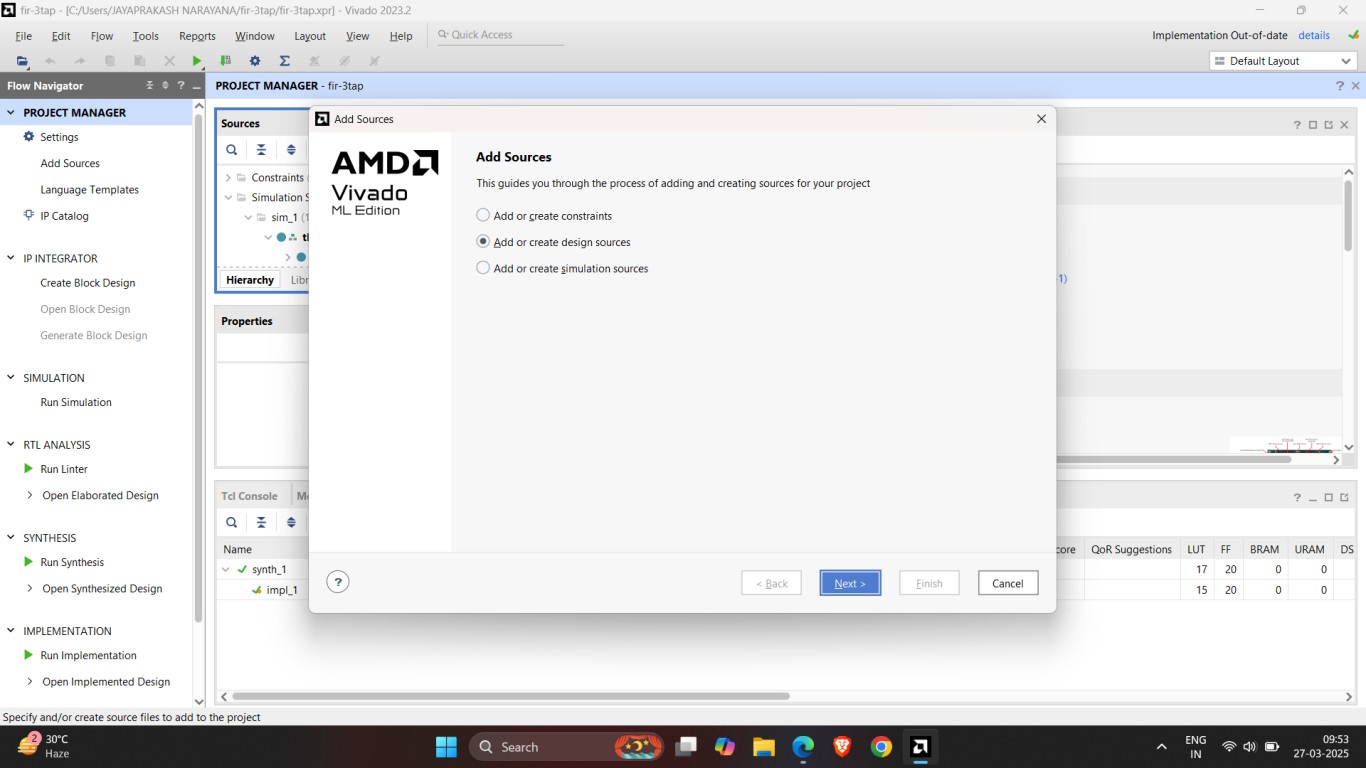
**Figure 6.6 Vivado Project Window**

#### Step 2: Edit The Project - Create source files

###### Design Sources

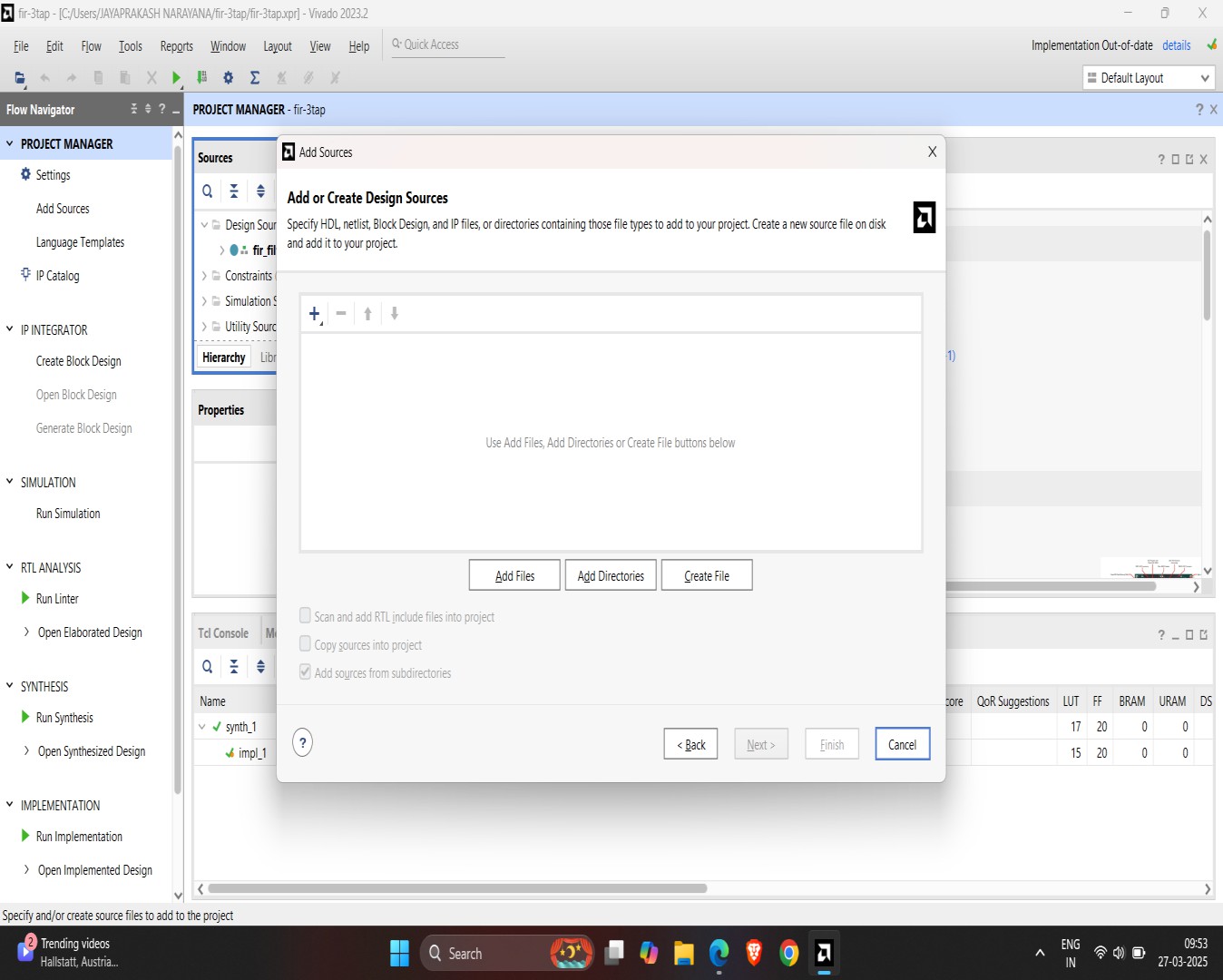
There are many ways to define a logic circuit, and many types of source files including VHDL, Verilog, EDIF and NGC netlists, DCP checkpoint files, TCL scripts, System C files, and many others. We had used the Verilog language in this project

To create a Verilog source file for the project, right-click on “Design Sources” in the Sources panel, and select Add Sources. The Add Sources dialog box will appear as shown – select “Add or create design sources” and click next.



###### Figure 6.7. Add or create design sources using Add Source Dialog

In the Add or Create Design Sources dialog, click on Create File, enter “FIR-4tap” as filename, and click OK. The newly created file will appear in the list as shown. Click Finish tomove to the next step.

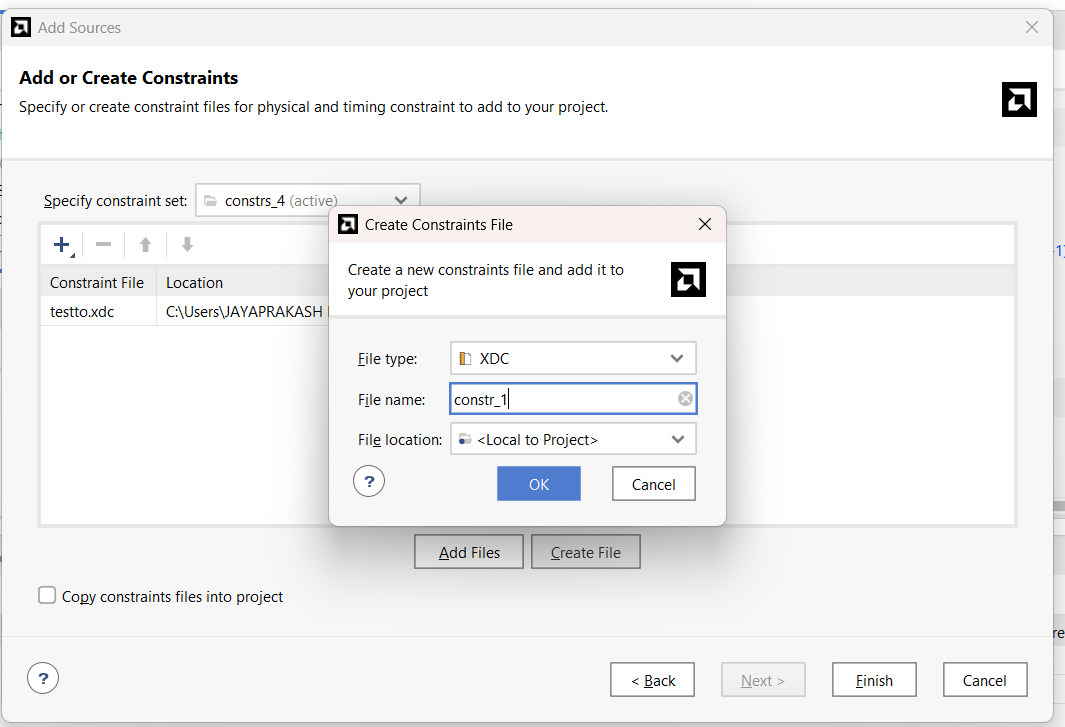


###### Figure 6.8. Creation of Design Source

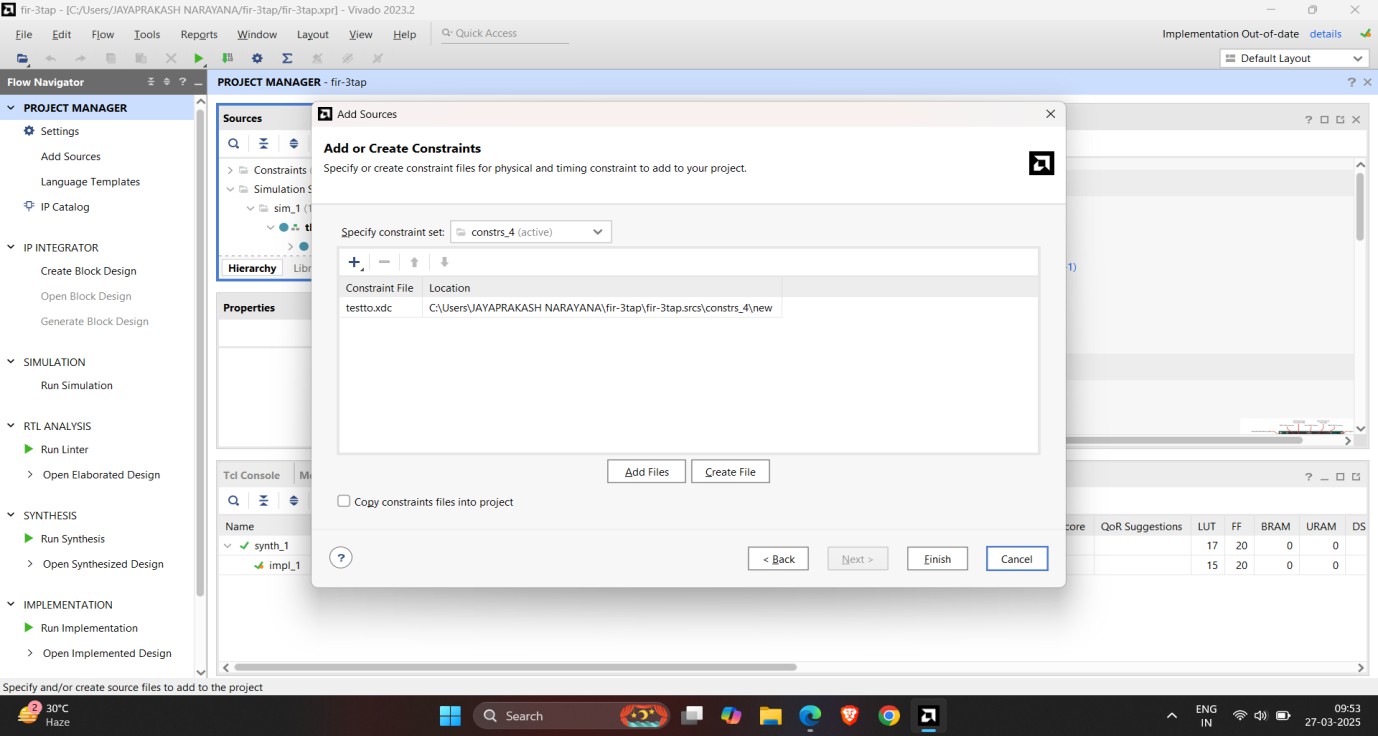
FileSkip the Define Module dialog by clicking OK to continue.

###### Constraints

Design sources, like Verilog HDL files, only describe circuit behavior. We must also provide a constraints file to map your design into the physical chip. In order to create a constraint file, expand the Constraints heading in the Sources panel, and right-click on constrs\_1, andselect Add Sources.



###### Figure 6.9. Add Source to Design Constraints

Click on Create File, enter project1 for the filename and click OK. The newly created file will appear in the list as shown. Click Finish to move to the next step.

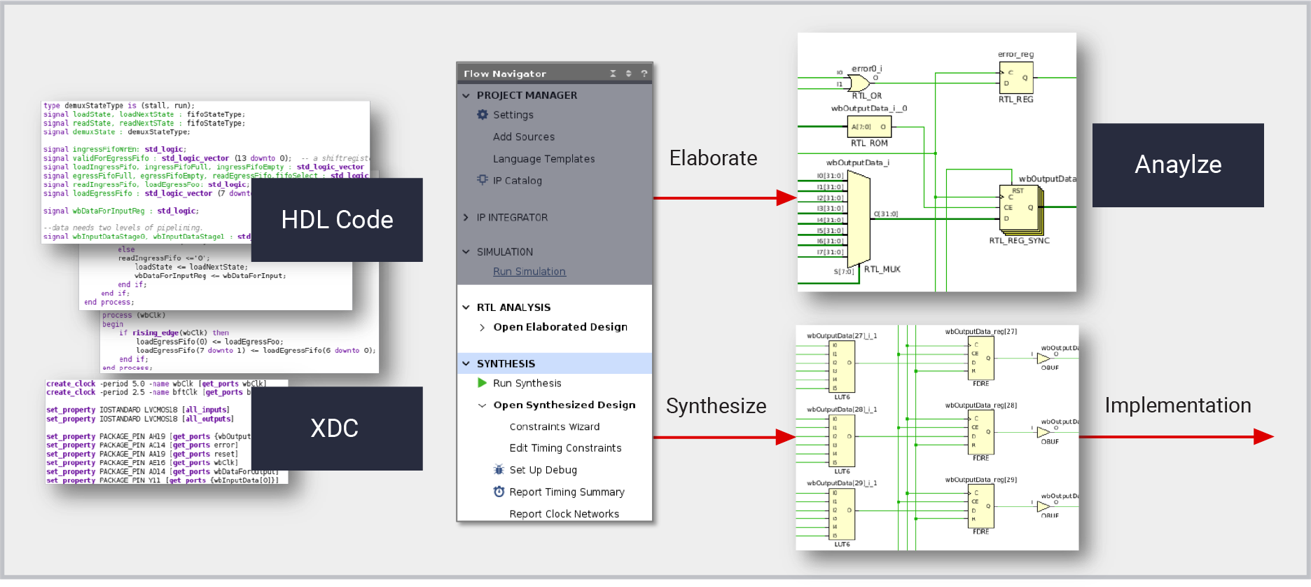
###### Figure 6.10. Creation of Constraints File

Double click “mbv1.xdc” to open the file, and replace the contents with the code below:

###### Step 3: Synthesize, Implement, and Generate Bitstream

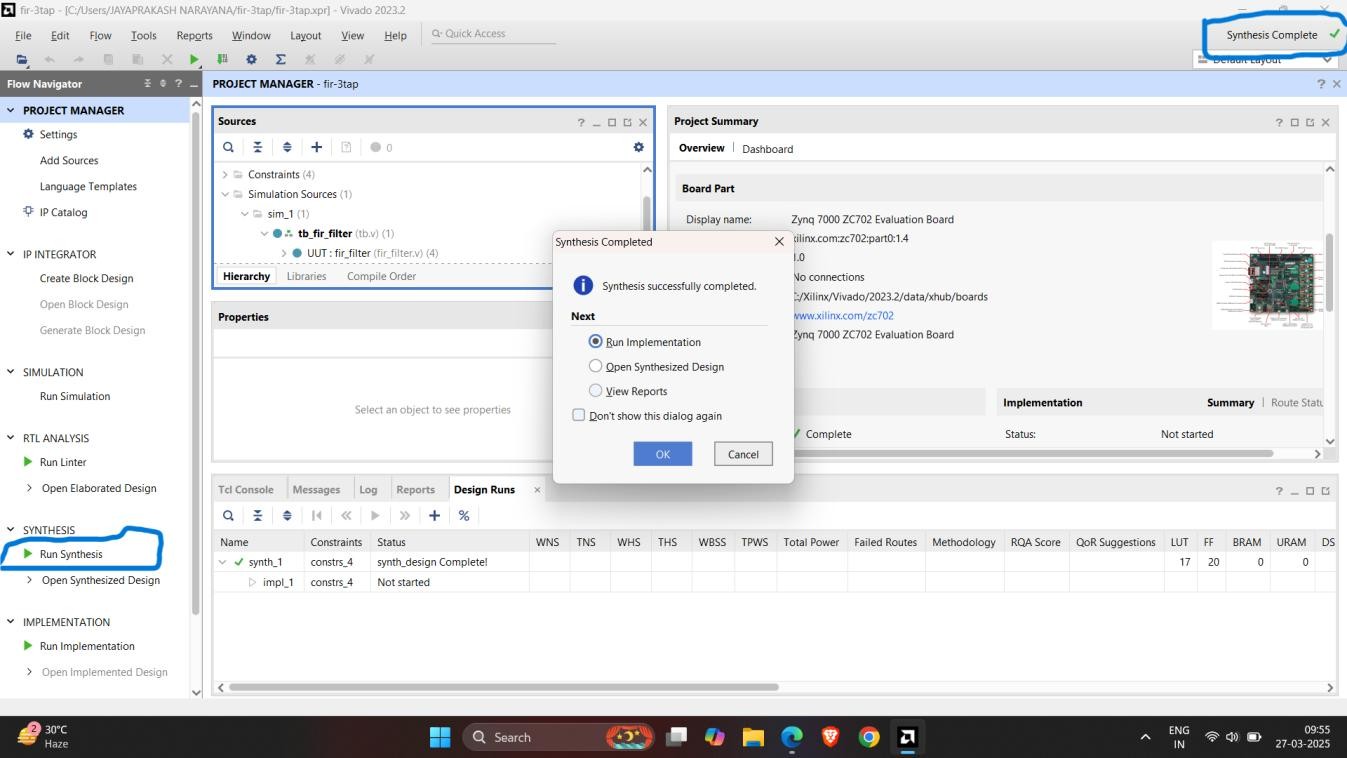
1. **Synthesis**

After Verilog code and constraint files were completed, we can Synthesize the design project. In thesynthesis process, Verilog code is translated into a “netlist” that defines all the required circuit components needed by the design (these components are the programmable parts of the targeted logic device - more on that later). Then Synthesize process will be started by clicking on Run Synthesis button in the Flow Navigator panel as shown.



###### Figure 6.11. Flow Of Synthesis

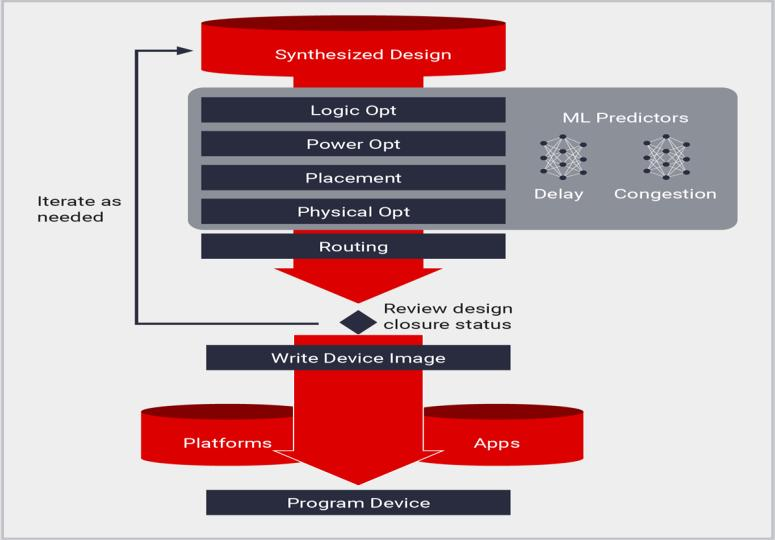
When synthesis is running, you can select the log panel located at the bottom of Project Manager to see a log of the currently running processes. Any errors that occur during the synthesis process will be described in the log.



**Figure 6.12. Start Synthesis process and monitor the synthesis log**

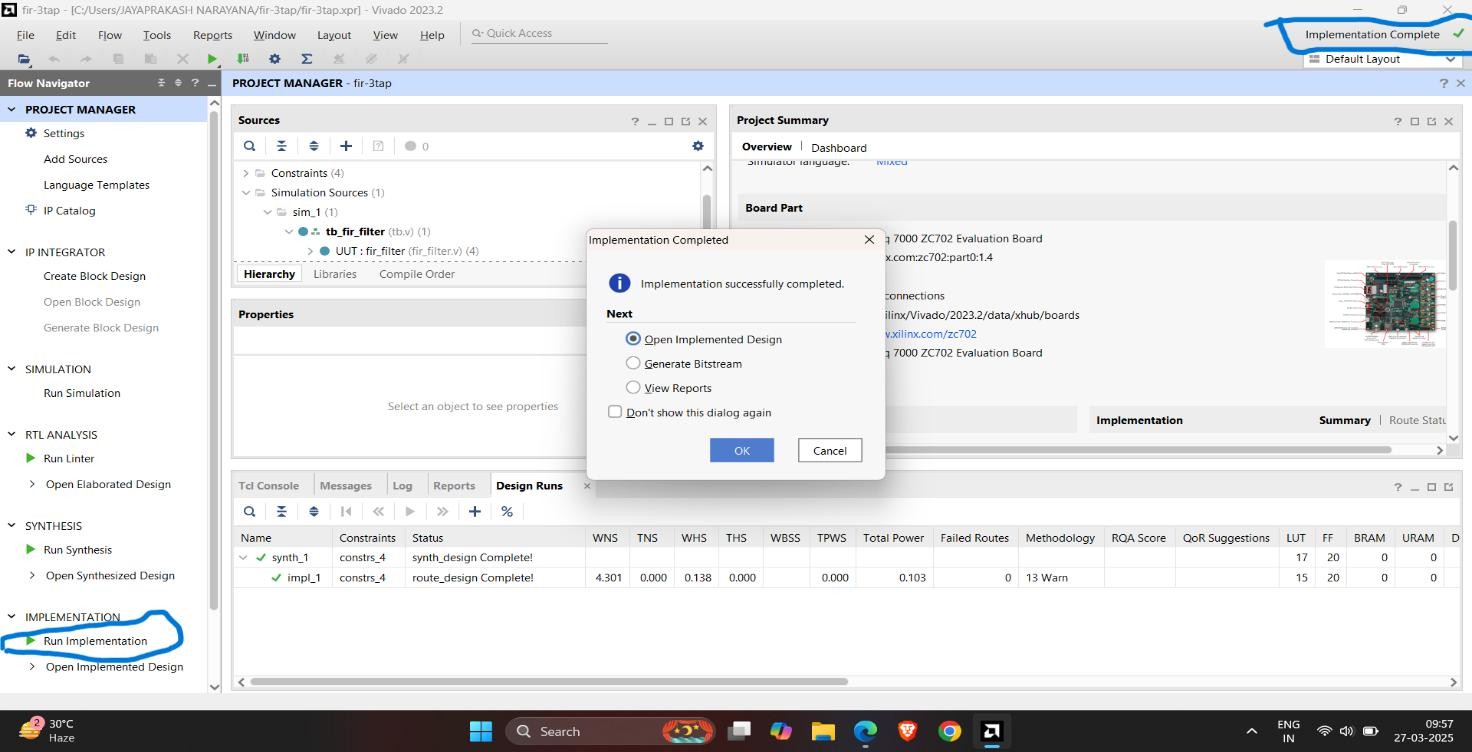
###### Implementation

After the design is synthesized, it must run the Implementation process. The implementation process maps the synthesized design onto the Xilinx chip targeted by the design. Click the Run Implementation button in the Flow Navigator panel as shown.



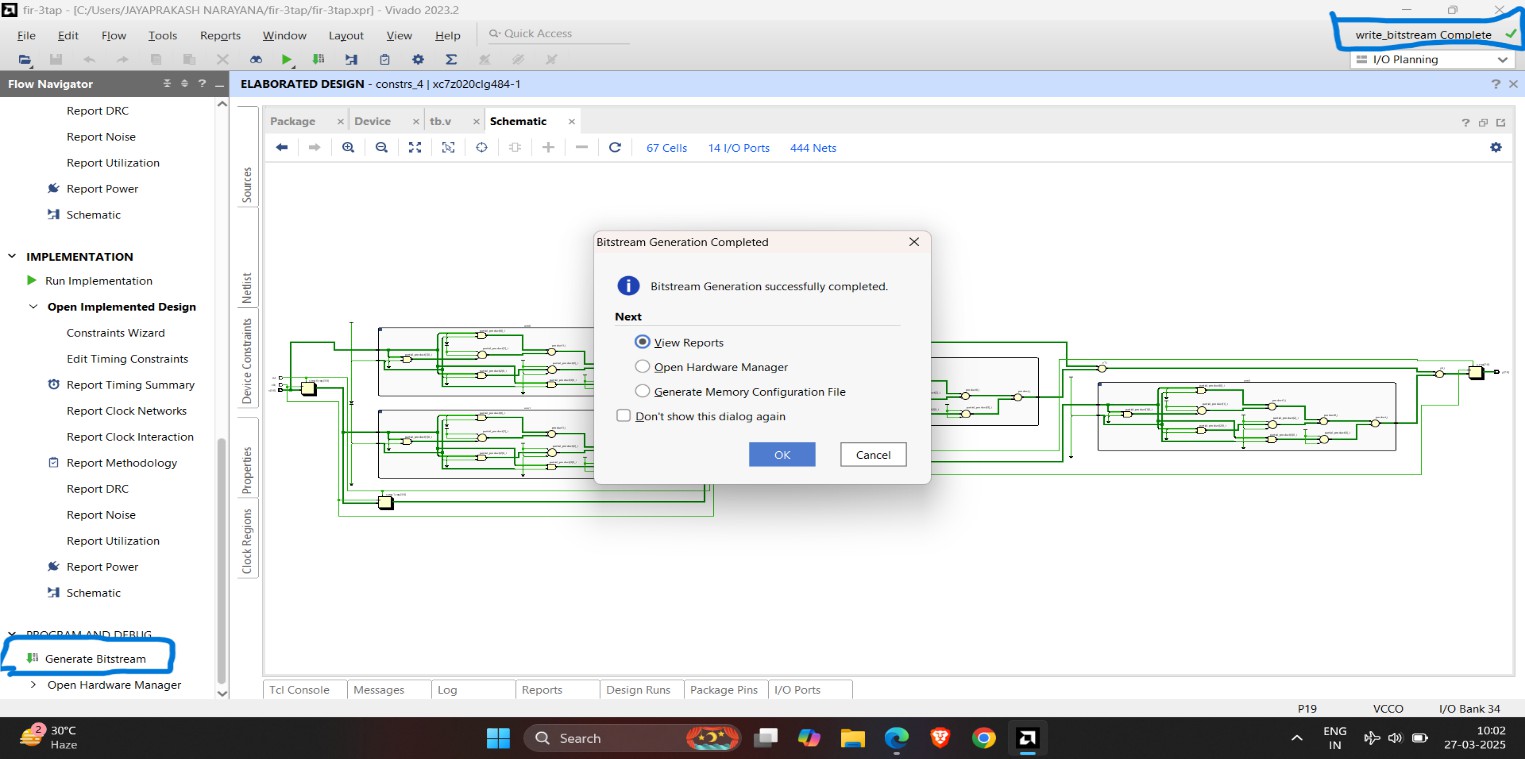
###### Figure 6.13. Flow of Implementation

When the implementation process is running, the log panel at the bottom of Project Manager will show details about any errors that occur.



**Figure 6.14. Start Implementation process and monitor the implementation log**

1. **Generate Bitstream**.



**Figure 6.15. Generate Bitstream**

**CHAPTER 7**

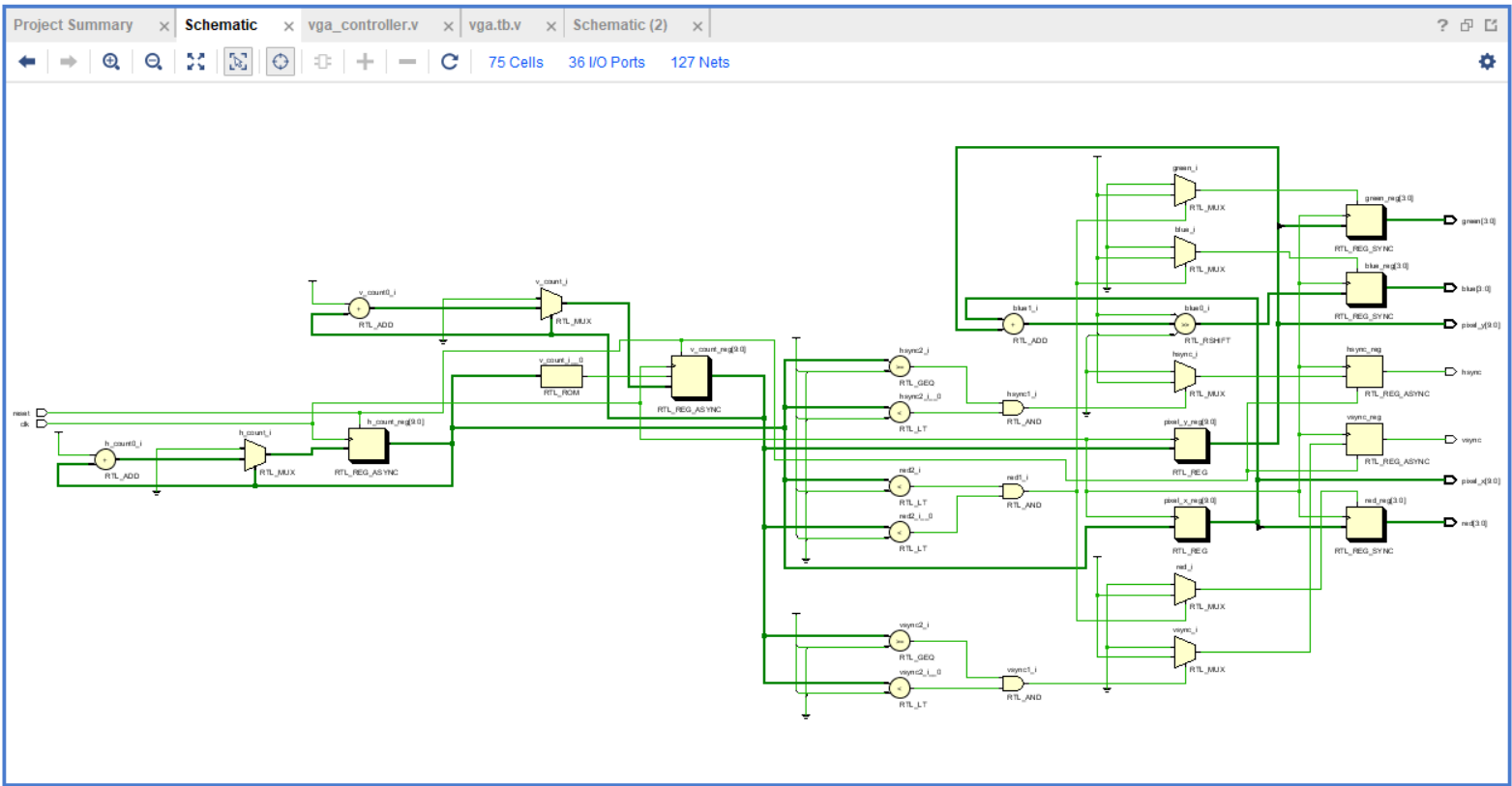
**RESULTS**

##### 7.1 SIMULATION RESULTS

**A green and black screen

Description automatically generated**

**Fig7.1 Simulation output**

****

**Fig 7.2 schematic**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Fig 7.3 : Final Power**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Fig 7.4 : Timing Analysis Report**

**A screenshot of a graph

AI-generated content may be incorrect.**

**Fig 7.5 : Utilization Report**

# CHAPTER 8

# ADVANTAGES AND APPLICATIONS

## Advantages and applications

###### Advantages

**1.Saves Time and Money**

Using Xilinx Vivado for VGA controller design helps reduce both development time and cost by providing a powerful simulation environment, reusable IP cores, and automation tools. Designers can quickly prototype and test their controller without needing multiple hardware iterations, saving on physical resources.

**2.Effective Learning**

Vivado offers a hands-on environment that enhances conceptual understanding through practical implementation. Designing a VGA controller reinforces knowledge in digital design, timing constraints, and hardware-software interaction, making the learning experience more comprehensive and impactful.

**3.24/7 Access to Learning**

Vivado and supporting documentation are available anytime, allowing learners and developers to work on their VGA controller design at their own pace. This flexibility supports asynchronous learning and helps accommodate different schedules or time zones.

**4.Access to Updated Content**

Xilinx regularly updates Vivado with the latest features, libraries, and IP cores. This ensures that your VGA controller project can leverage modern design techniques, optimized logic, and compatibility with newer FPGA devices.

**5.Scalable**  
The VGA controller design can start with a basic model and scale up with additional features like color depth, text overlays, or sprite handling. Vivado’s modular design flow supports such scalability, allowing for incremental development and integration into larger systems.

###### 8.2 Applications

###### 1.Medical Imaging Systems

###### A VGA controller can display high-resolution images from medical devices like X-ray or MRI machines. Implementing it in Vivado allows real-time visualization and accurate pixel mapping on monitors.

###### 2.Industrial Automation

###### VGA controllers are used to display system statuses or machine diagnostics on industrial HMI screens. Using Vivado enables efficient integration with FPGAs for fast, reliable visual feedback.

###### 3.Digital Signal

###### In signal processing projects, a VGA controller helps visualize waveforms or spectrum outputs. Vivado enables precise timing control needed for real-time signal display on VGA-compatible screens.

###### 4.Gaming Consoles

###### Basic games developed on FPGA platforms rely on VGA controllers to render graphics. Vivado supports this by offering IP cores and simulation tools for smooth video output.

###### 5.Surveillance Systems

###### Video feeds in surveillance can be output to screens using VGA controllers. With Vivado, designers can create low-latency, customized display modules for real-time monitoring.

# CHAPTER 9

# CONCLUSION AND FUTURE SCOPE

##### 9.1 CONCLUSION

The VGA controller was successfully designed and implemented using Xilinx Vivado with Verilog, enabling real-time video signal generation for display applications. The design included precise timing control for synchronization signals and pixel data output, ensuring stable and clear VGA output. Functionality was verified through simulation and testbenches, and the design was deployed on the ZedBoard FPGA for real-time validation. The project demonstrates efficient hardware-level VGA signal control, making it suitable for embedded display systems and educational applications.

##### 

##### 9.2 FUTURE SCOPE

##### The VGA controller design can be extended in future by integrating with AI and image processing algorithms to enable intelligent display systems, such as adaptive brightness or real-time object tracking on screen. It holds potential for use in embedded vision systems, educational tools, and human-computer interaction interfaces. Additionally, the controller can be enhanced to support higher resolutions and HDMI output for modern display requirements. Its integration with machine learning models could enable smart visual feedback in robotics and IoT applications, expanding its relevance in advanced embedded and display technologies.

# REFERENCES

### REFERENCES

* 1. V.VittalReddy, Udathu Jaya Madhuri, Vutukuri Sailaja,Zubera Farheen Khatun,Velpula Jagadish Kumar,” Implementation of Digital Filter Using Verilog HDL”, IJRASET ISSN: 2321-9653,2024
  2. L. Y. Akshitha V. Ramesh, Apeksha Ravi Kumar, Amulya S. Iyengar, “Implementation and Design of FIR Filters using Verilog HDL and FPGA,” 2020.
  3. V. Thamizharasan and N. Kasthuri, “FPGA implementation of high performance digital FIR filter design using a hybrid adder and multiplier,” Int. J. Electron., vol. 110, no. 4, pp. 587–607,Apr.2023,doi:10.1080/00207217.2022. 2098387.
  4. Swati Chulet, Himanshu Joshi “FIR Filter Designing Using Wallace Multiplier”,IJETR ISSN: 2321-0869, Volume-3, Issue-6, June 2015.
  5. O. Venkata Krishna “Design and Implementation of FIR Filter using Low Power and High- Speed Multiplier and Adders” CVR Journal ISSN:2277-3916.
  6. G. Appala Naidu, “Implementation of Wallace tree multiplier using Xilinx with Verilog” – IERJ, E-ISSN No: 2454-991, Vol. 9, Issue 8, August 2023.
  7. A. K. Jameil, Y. A. Ahmed, and S. Albawi, “Efficient FIR Filter Architecture using FPGA,” Recent Adv. Comput. Sci. Commun., vol. 13, no.1, pp. 91–98, Mar. 2020, doi:10.2174/2213 275912666190603115506.