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

Submitted by

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

Under the esteemed guidance of

Smt. D. Rohini, M. Tech(Ph. D),

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

Smt. D. Rohini, M. Tech(Ph. D),
Assistant Professor,
Department of ECE,
G. Pulla Reddy Engineering College (Autonomous), Kurnool.

HEAD OF THE DEPARTMENT

Dr. K. SURESH REDDY, M. Tech., Ph. D,
Professor and Head of the Department,
Department of ECE,
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 Vivado

Applications:

- Embedded Systems
- FPGA Prototyping and Development
- Image processing

CONTENTS

LIST OF FIGURES PAGE NO

CHAPTER 1

INTRODUCTION

1. 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.

1.1 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.

1.2 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.

1.3 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

2. LITERATURE REVIEW

2.1 REVIEW OF RELATED LITERARURE

2.1.1 "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**

2.1.4."FPGA-Based VGA Controller Implementation for Image Display Applications" (IJERT, 2014)

Authors: S. R. Patil, M. V. Dhamane

This paper focuses on designing a VGA controller for image display using an FPGA platform and Verilog HDL. The implementation includes modules for synchronization signal generation, framebuffer interfacing, and pixel-level image rendering. The target resolution is 640×480 at 60 Hz, with a pixel clock of 25.175 MHz. The design emphasizes real-time display capability and efficient memory access using on-chip RAM for storing pixel data. Simulation and timing analysis confirm correct synchronization and stable display output. The system is suitable for low-power, embedded image processing systems where cost and speed are critical factors.

2.1.5. "Implementation of VGA Controller using FPGA for Educational Purposes" (IJRASET, 2016)

Authors: A. R. Bhagat, N. M. Patel

This paper presents a VGA controller project designed primarily for educational and demonstration purposes. The system is implemented on a Spartan-3E FPGA board using VHDL, with the Xilinx ISE toolchain used for synthesis, simulation, and testing. Key components include timing pulse generators, color data registers, and VGA signal drivers. The project targets basic VGA output (640×480), aiming to introduce digital design students to concepts such as clock division, scan line control, and signal synchronization. Despite its simplicity, the controller achieves clear and flicker-free display output, making it a valuable tool for understanding the basics of VGA signal generation and FPGA-based display interfacing.

2.2 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.

2.2.1 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.

Drawback:

Timing Accuracy Issues:

Microcontrollers often fail to maintain the precise timing required for VGA signals, especially at higher resolutions. Since VGA relies on strict synchronization pulses (HSYNC, VSYNC, pixel clock), any timing deviation caused by limited processing speed or interrupt handling can lead to flickering, unstable images, or complete display distortion. This makes microcontrollers unsuitable for applications demanding high-resolution or real-time video performance.

2.2.2 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.

DRAWBACK:

High Power Consumption:

GPUs are designed for complex graphics processing and high-resolution rendering, which

inherently requires substantial power. While they deliver excellent performance, their high energy demands make them unsuitable for low-power embedded systems or battery-operated devices. In such applications, power efficiency is critical, and the excessive consumption by GPUs leads to thermal management issues and reduced system lifespan, making them impractical for compact or portable solutions.

2.2.3 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.

DRAWBACK:

Steep Learning Curve:

FPGA-based VGA controller design involves working with Hardware Description Languages (HDLs) such as Verilog or VHDL, along with understanding digital circuit design, timing constraints, and synthesis processes. For beginners or developers unfamiliar with hardware-level programming, this presents a significant challenge. The complexity of debugging hardware logic, managing timing issues, and optimizing resource utilization adds to the difficulty. As a result, the steep learning curve can slow down development time and limit accessibility for those new to FPGA design.

2.2.4 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.

DRAWBACK:

Limited Scalability:

SRAM-based VGA controllers rely on static RAM to store pixel data for display rendering. While this method allows for fast and smooth image updates, it becomes increasingly constrained as display resolution or complexity grows. High-resolution or dynamic graphics require a larger memory footprint, which can exceed the available SRAM on typical FPGA boards. As a result, the system may struggle to support scalable designs or real-time graphical updates, making it less suitable for applications that demand high-resolution displays or dynamic video content.

CHAPTER 3
PROJECT METHODOLOGY

3. PROJECT METHODOLOGY

Proposed Project

The design and implementation of the VGA Controller in this project follow a structured methodology to ensure accuracy, efficiency, and adaptability. The system is developed using Verilog HDL. The key objective is to overcome the major limitations identified in traditional VGA controller systems and to achieve a low-power, high-performance, and reliable display interface.

3.1.1 Requirement Analysis

The **requirement analysis** phase focuses on determining the necessary **VGA signal parameters** required for achieving a **640×480 @ 60Hz resolution**. This includes setting up accurate timing specifications to ensure proper display synchronization. The goal is to establish the necessary signals and their timing relationships for the VGA controller design.

- **Horizontal sync pulse width**: Defines the duration of the horizontal sync pulse to correctly reset the horizontal scan.
- **Front/back porch**: The time intervals before and after the horizontal sync pulse that help frame the active video data.
- **Vertical sync pulse and scan line definitions**: Ensures the vertical synchronization of the display by setting the sync pulse and scan line timing for the monitor.
- **Calculate pixel clock frequency: 25.175 MHz**: The clock rate required to handle the pixel data and synchronize the screen refresh at the specified resolution.
- **Establish timing diagrams to map sync signals and active display regions**: Draw diagrams representing the timing relationships of horizontal and vertical signals and the active video display areas.

3.1.2 System Design in Verilog HDL:

- Design is structured into modular components:
  - **Horizontal Synchronization Module**

- **Vertical Synchronization Module**
- **Video Enable Logic**
- **Pixel Generation Logic (Color patterns / test outputs)**

This approach ensures accurate timing signals, addressing **microcontroller-based timing issues**.The design uses **parameterized Verilog** for flexible screen resolution and timing reusability. Unlike GPU-based approaches, this HDL-based method is **lightweight and avoids driver complexity**.

3.1.3 Simulation and Functional Verification in Vivado

In this phase, the **Vivado Simulator** is used to validate the functionality of the VGA controller. The simulation process ensures that all timing and synchronization signals are accurately generated. Functional verification checks the **generation of sync pulses**, the **video enable region**, and the **pixel data transitions** to ensure the controller works as intended.

- **Verify the correct generation of h_sync and v_sync pulses**: Ensures that the horizontal and vertical sync pulses are generated correctly to synchronize the display.
- **Confirm proper video enable region**: Checks that the active video region (where actual pixel data is displayed) is correctly enabled during the simulation.
- **Check pixel output transitions and blanking periods**: Verifies that the pixel transitions happen correctly and that the blanking periods are accurately timed during the simulation.

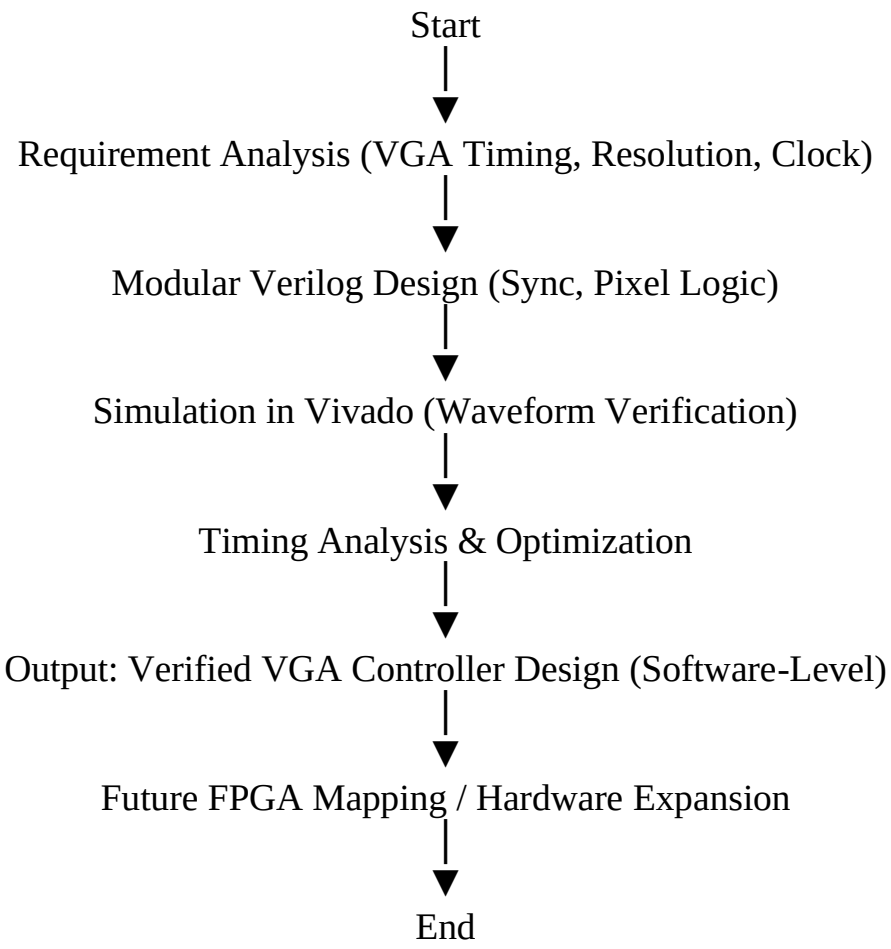
3.1.4 Timing and Optimization Strategy

In this phase, **Vivado** tools are used to validate timing constraints and ensure that the design meets the required timing specifications. The focus is on optimizing key aspects of the design, such as **clock synchronization**, **logic usage efficiency**, and maintaining **design readability** through a hierarchical structure. These optimizations help improve performance, reduce resource usage, and enhance maintainability.

- **Use Vivado tools for timing constraint validation**: Vivado's timing analysis tools ensure that the timing requirements of the VGA controller are met and that no timing violations occur in the design.

- **Optimize clock synchronization**: Ensures that the system clock and sync signals (horizontal and vertical) are synchronized accurately to prevent glitches or display errors.
- **Optimize logic usage efficiency**: The design is optimized to use fewer logic resources without compromising performance, ensuring an efficient and cost-effective solution.
- **Ensure hierarchical design readability**: The project's structure is kept modular and easy to read, allowing for easier debugging, future modifications, and scalability.

3.1.5 Methodology Flowchart:



3.2 Block Diagram

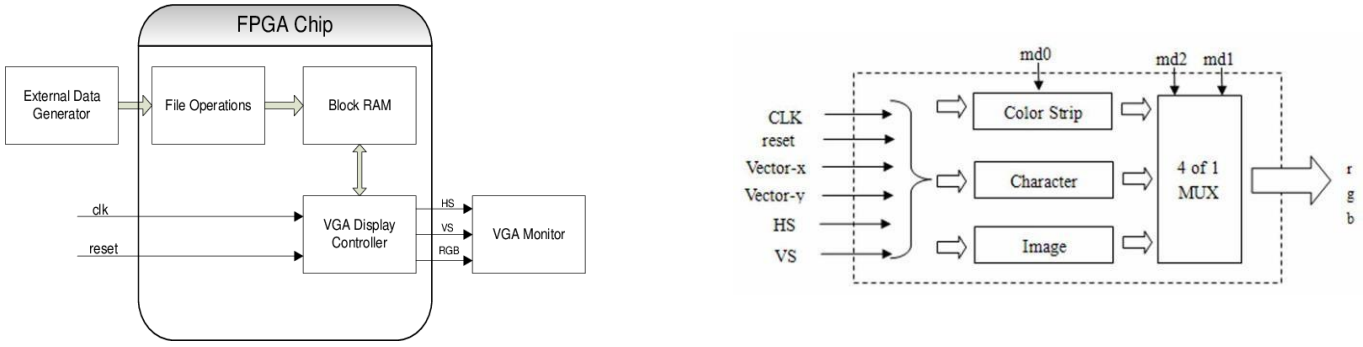


Fig 3.2.1 Block diagram of VGA Controller in Xilinx Vivado

3.2.1 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**.**

3.3 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.

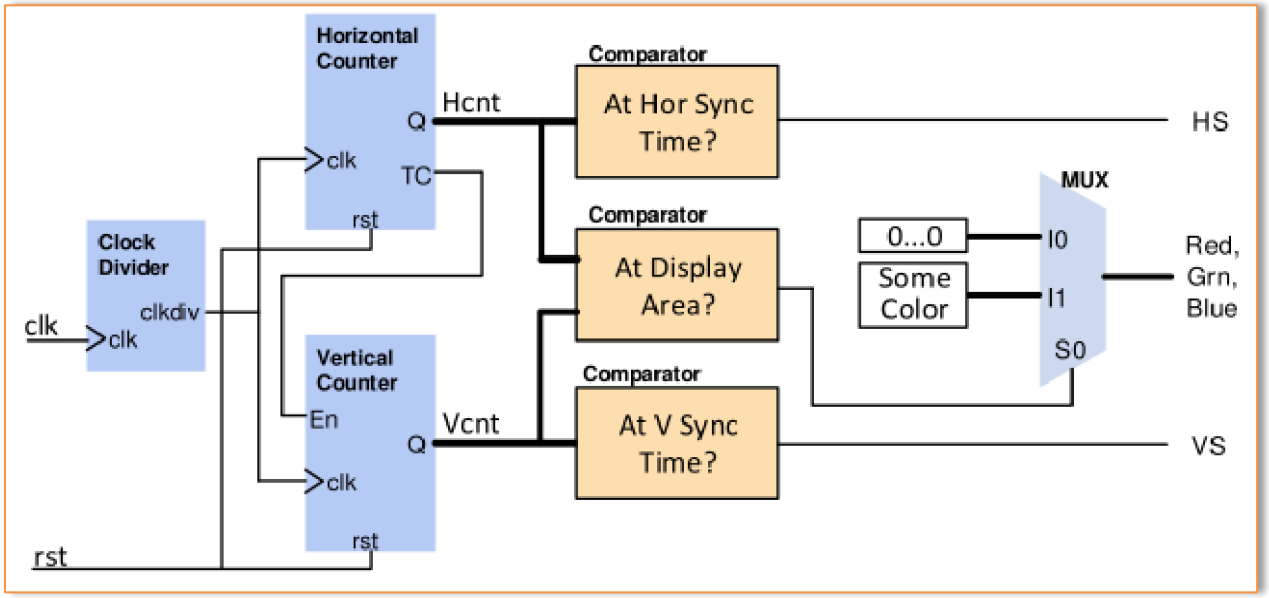


Fig 3.3.1 Detailed Design of VGA Controller Module

Figure Description:

This figure provides a comprehensive visual representation of the VGA controller module's architecture. It illustrates the intricate connections and internal components involved in generating video signals.

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
INTRODUCTION TO VERILOG HDL & VLSI

4.1 FUNDAMENTALS OF VERILOG HDL

Verilog is a hardware description language used to design digital logic. It is mostly used in designing digital circuits at the 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:

I. Gate-Level modelling

II. Dataflow modelling

III. Behavioral modelling

1. Gate-Level Modelling

Gate-level modelling represents the circuit in terms of logic gates and their interconnections. It closely resembles a schematic and is the **lowest level** of abstraction in Verilog.

- It uses **primitive gates** such as and, or, not, nand, nor, xor, etc.
- Designers must manually connect inputs and outputs, mimicking real hardware behavior.
- It is best suited for **small-scale designs** or to represent already synthesized logic.
- Simulation at this level is **slower and more complex**, but it provides a very **accurate** hardware representation.

2. Dataflow Modelling

Dataflow modelling describes the circuit behavior in terms of data movement between registers and how data is processed, using continuous assignments.

- It uses the assign statement along with logical, bitwise, and arithmetic operators.
- Offers medium-level abstraction, focusing on how data flows through the system rather than specific gates.
- Easier to read and write compared to gate-level modelling.
- Suitable for implementing combinational logic with moderate complexity.

3. Behavioral Modelling

Behavioral modelling is the highest level of abstraction in Verilog. It describes what the circuit should do rather than how it should do it.

- It uses constructs like always, if-else, case, for loops, and begin-end blocks.
- Supports sequential logic using clock and reset signals.
- Ideal for modeling complex systems, state machines, counters, controllers, and testbenches.
- Focuses on algorithmic behavior rather than hardware structure.

4.2 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.

4.3 SIGNIFICANCE OF VLSI:

4.3.1 Verilog HDL for RTL Design:

- Verilog HDL is used in VLSI design to describe hardware behavior at multiple abstraction levels—from gate-level to behavioral.
- In this project, the VGA controller is developed using Verilog HDL at the Register Transfer Level (RTL), enabling structured, modular, and synthesizable code.
- The design is verified through simulation and then synthesized onto the FPGA, achieving a working visual interface.

4.3.2 Timing and Synchronization:

- Precise timing is crucial in VGA controller design to ensure correct generation of horizontal and vertical sync pulses, pixel data timing, and frame rendering.
- The VGA controller includes logic to manage pixel clocks, sync counters, and data enable signals, synchronized to the VGA 640x480 @ 60Hz standard (using a 25.175 MHz pixel clock).
- Accurate timing ensures stable and flicker-free video output on external monitors.

4.3.3 Modular Design and Hardware Reusability:

- The project adopts a modular design approach, with separate functional blocks for synchronization, video signal generation, color pattern rendering, and character display.
- This modularity supports hardware reusability, where each module can be reused or modified independently, simplifying design updates and enhancements

4.3.4 Optimized Resource Utilization:

- Efficient utilization of FPGA resources such as LUTs (Look-Up Tables), Flip-Flops, and Block RAM is a key aspect of this project.
- Logic is optimized to minimize hardware usage while maintaining functional accuracy, making the design suitable for mid-range FPGA platforms.

- The display logic is designed to operate with minimal switching activity, indirectly contributing to reduced power consumption.

4.3.5 Low-Power Considerations:

- Although VGA controllers are not power-intensive by design, certain VLSI power optimization strategies are considered:
    - Clock Gating is selectively used to reduce switching in idle modules.
    - Minimal memory access and simplified logic reduce unnecessary transitions and help in lowering dynamic power.
- The overall architecture ensures reliable performance with efficient power management on the FPGA board.

CHAPTER 5
SOFTWARE TOOL

5. 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

1. 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.

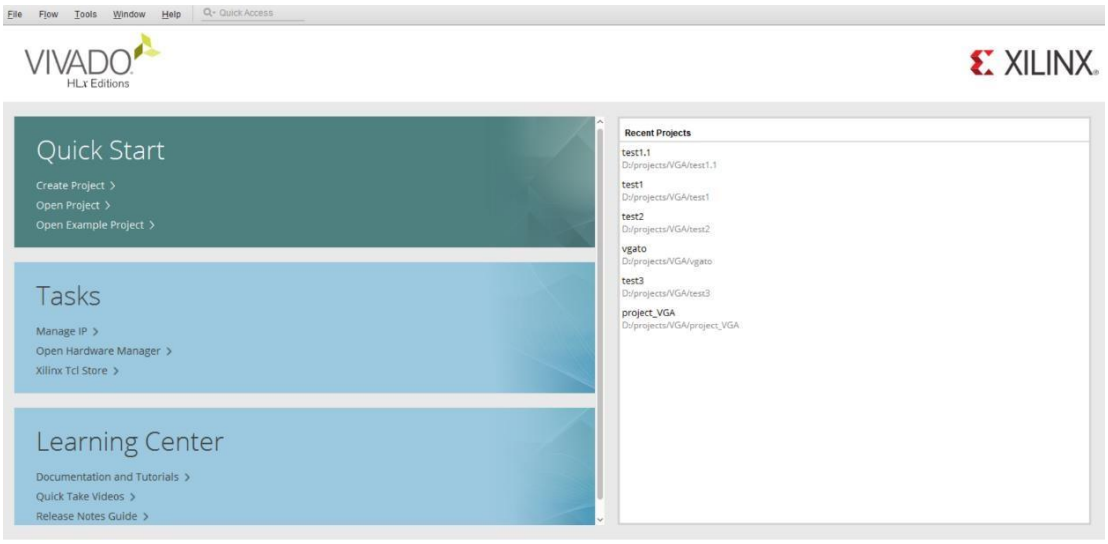


Figure 5.1 Vivado Start-Up Window

Figure Description:

Vivado Start-Up Window This window is the initial interface displayed upon launching Vivado. It provides access to recent projects and various creation or opening options.

2. 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.

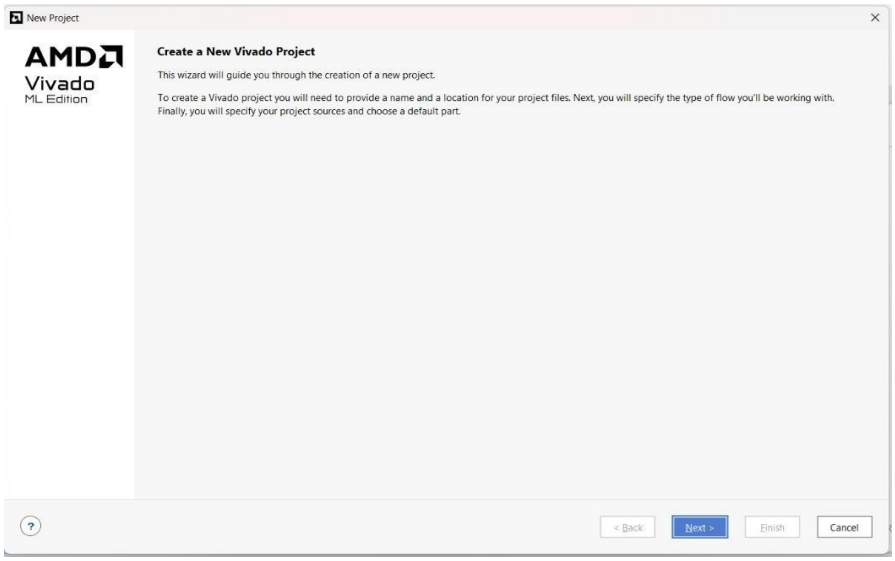


Figure 5.2 Create Project Dialog

Figure Description:

This dialog initiates the process of creating a new project in Vivado. It guides users through setting up design parameters and files.

3. Set Project Name and Location

In this step our project name project_3 was given at the option of project name as shown in Figure.

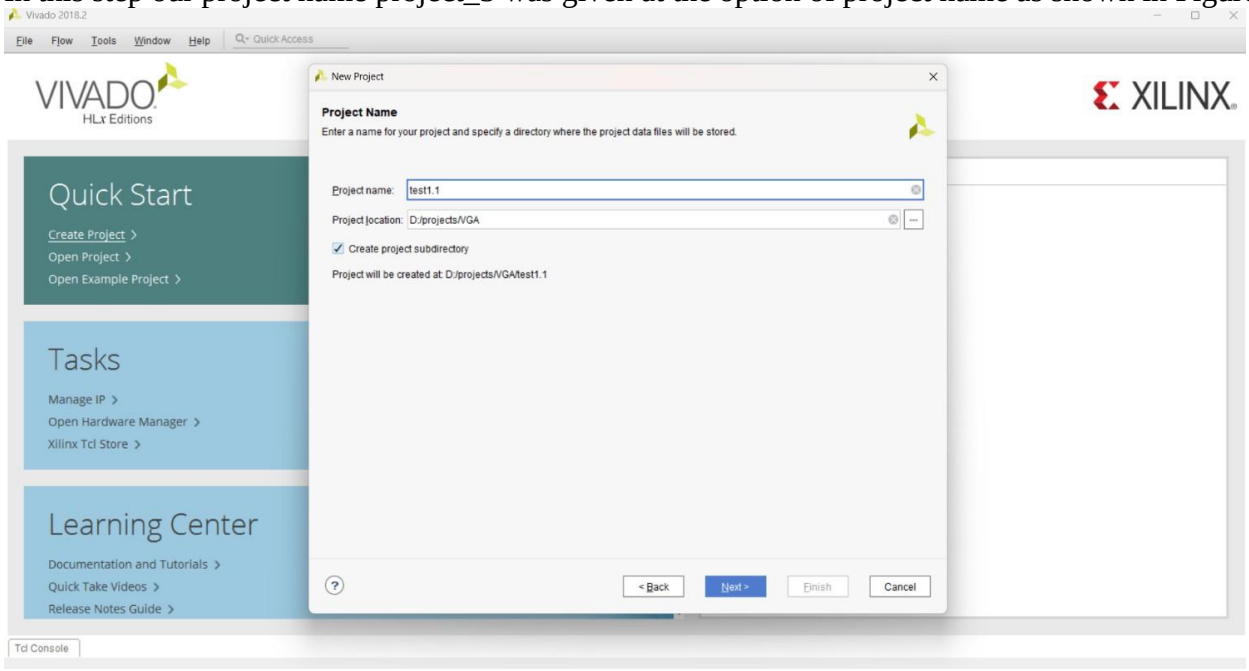


Figure 5.3. Enter Project Name

Figure Description:

Enter Project Name In this step, the user specifies the name and location for the new project. It helps organize and save the project under a defined directory.

4. 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	**xc7a100tcsg324-1**
Family	**Artix-7**

Package	**Clg324**
Speed Grade	**-1**

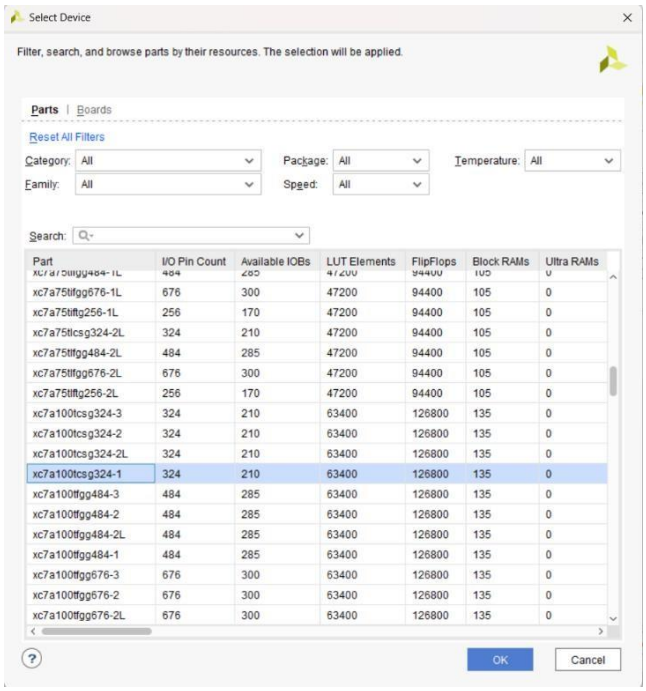


Figure 5.4. Select Board Evaluation and development kit

Figure Description:

This interface allows users to select the target FPGA board. It ensures compatibility by auto-configuring the project to match board specifications.

5. 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.

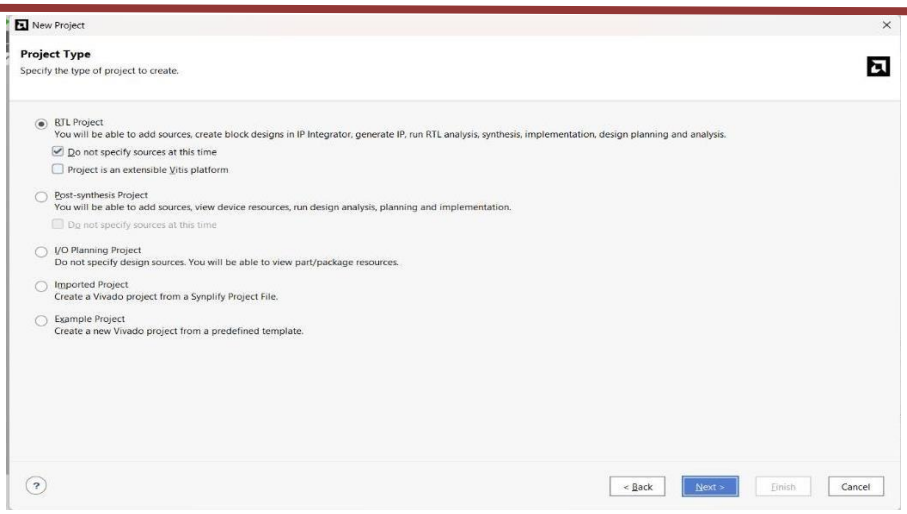


Figure 5.5. Create Project Summary

Figure Description:

Displays a summary of the selected project settings. It confirms the final configuration before project creation is completed.

6. 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.

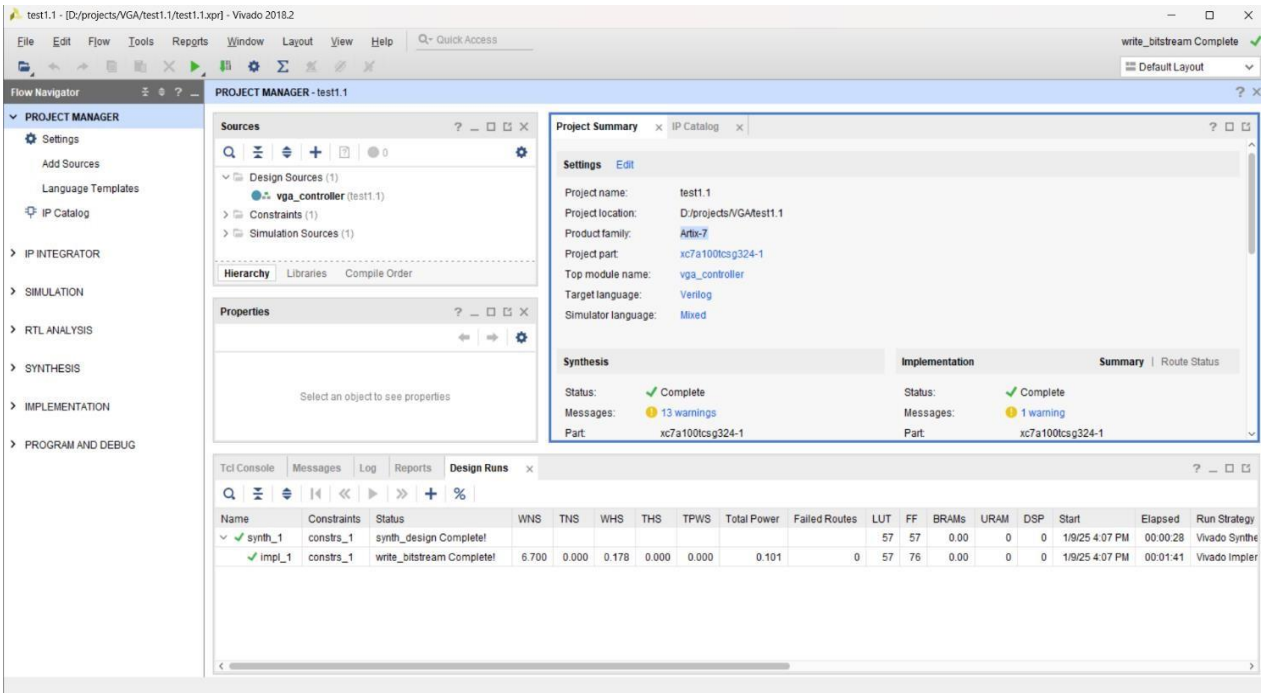


Figure 5.6 Vivado Project Window

Figure Description:

This is the main working environment of Vivado. It includes design sources, simulation tools, and flow navigation.

Step 2: Edit The Project - Create source files

1. 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.

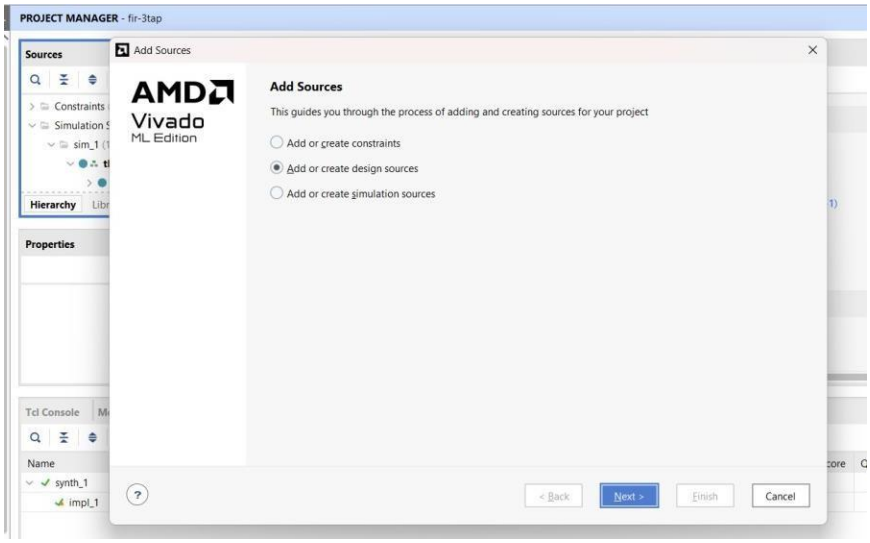


Figure 5.7. Add or create design sources using Add Source Dialog

Figure Description:

Users can add new or existing Verilog/VHDL files here. It serves as the primary method to import HDL code into the project.

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.

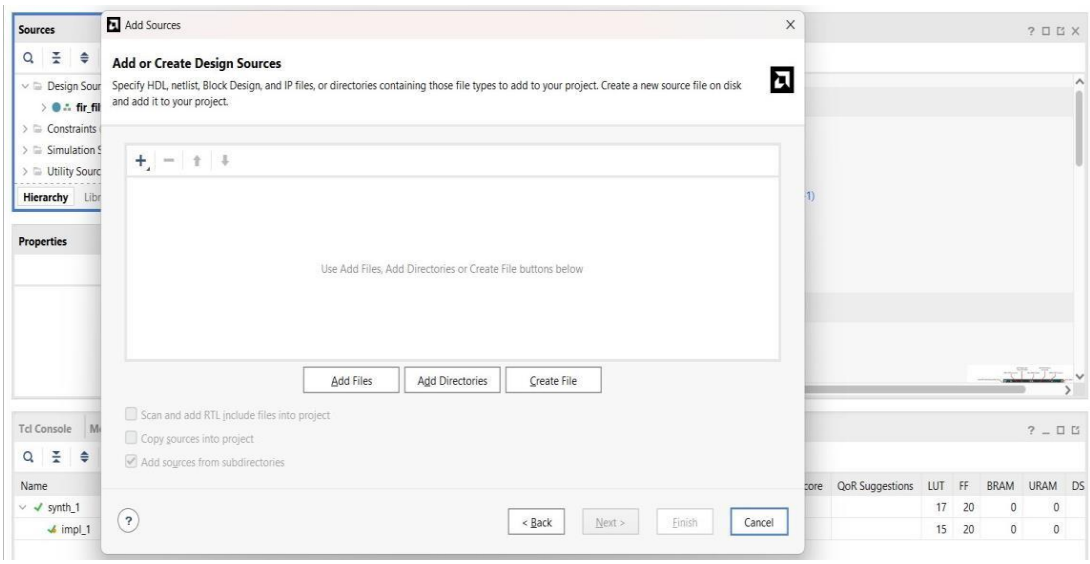


Figure 5.8. Creation of Design Source

Figure Description:

Demonstrates the successful creation or addition of a design file. The file is now ready for synthesis and implementation.

FileSkip the Define Module dialog by clicking OK to continue.

2. 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.

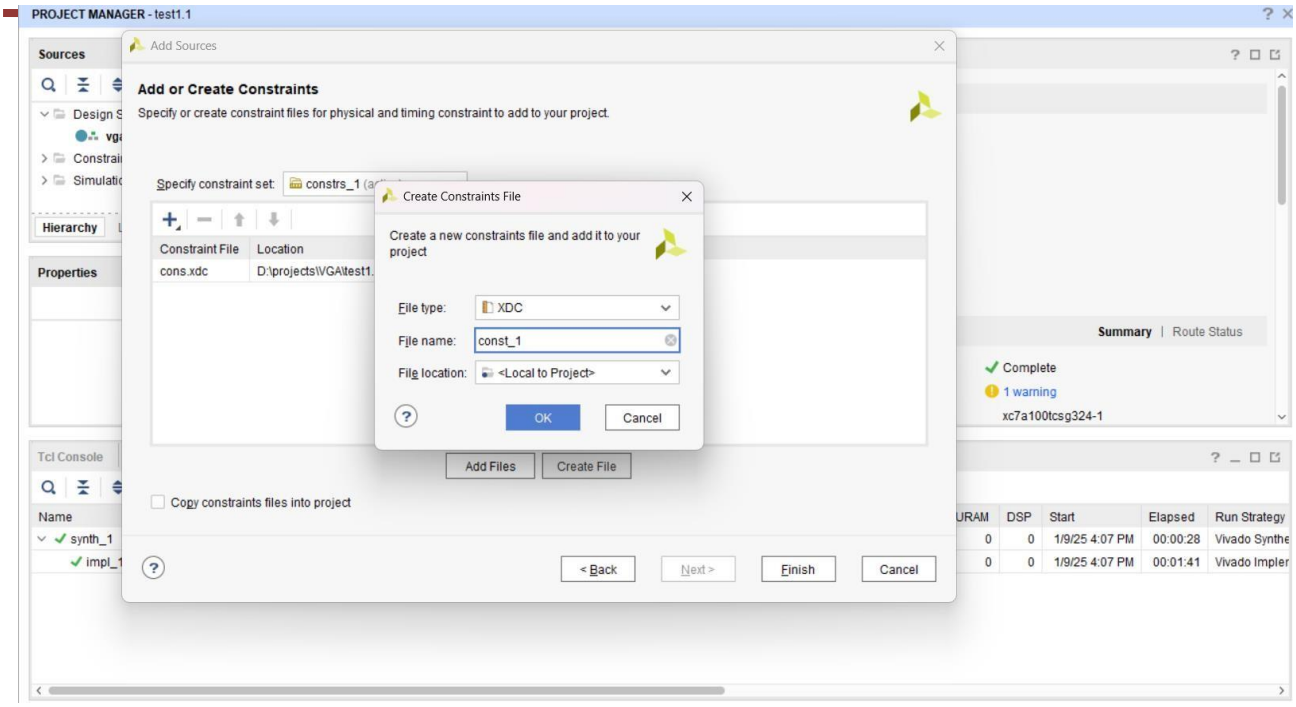


Figure 5.9. Add Source to Design Constraints

Figure Description:

Allows users to define timing, pin assignments, and other design constraints. Ensures proper operation on the selected FPGA board.

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.

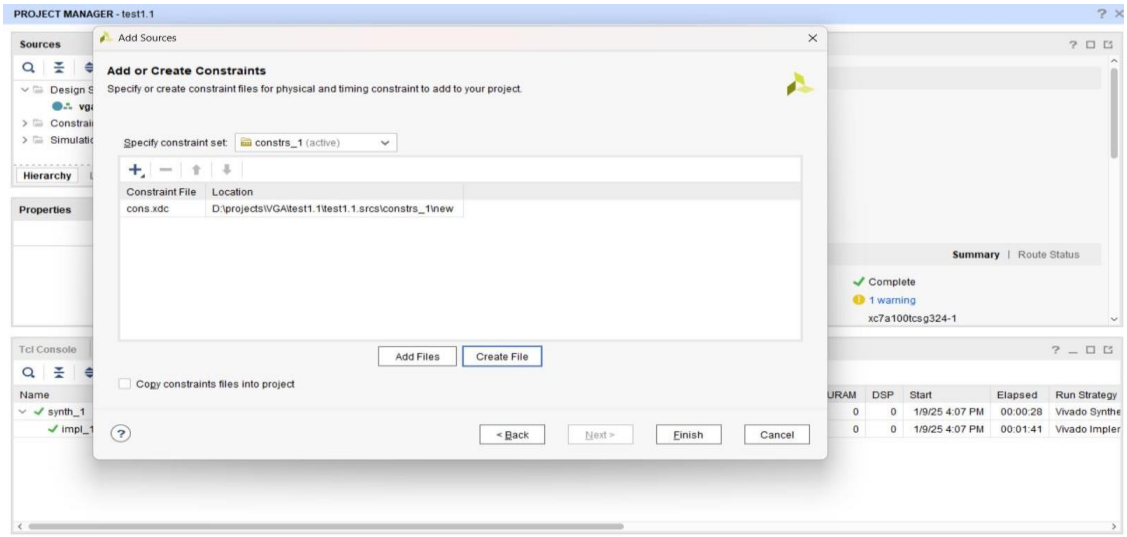


Figure 5.10. Creation of Constraints File

Figure Description:

Shows a newly created XDC (constraints) file. This file links logical ports to physical FPGA pins.

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.

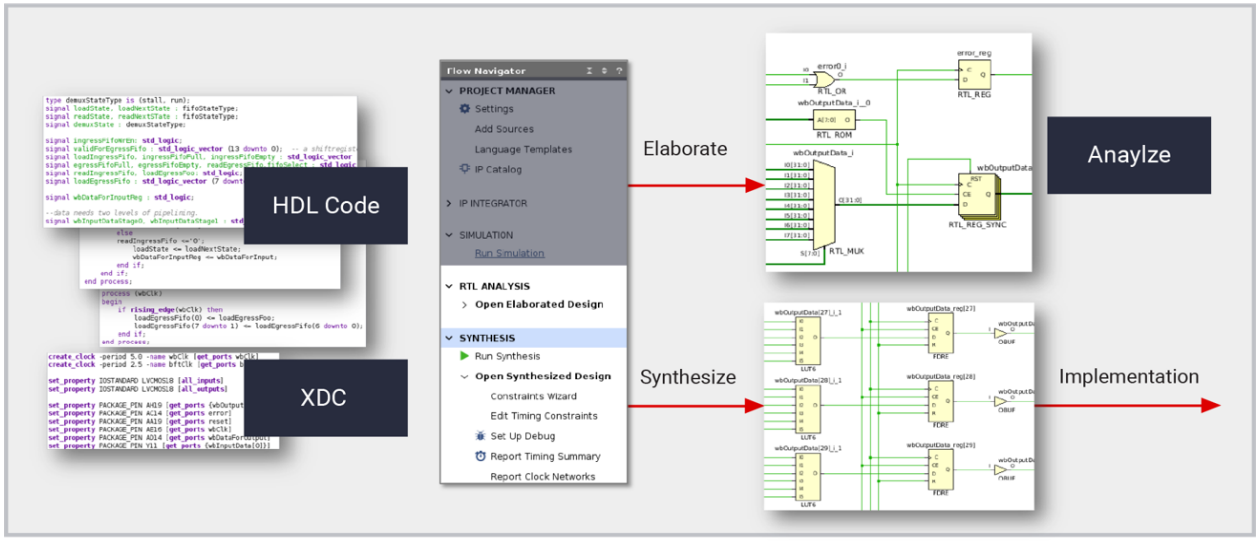


Figure 5.11. Flow Of Synthesis

Figure Description:

Displays the process of translating HDL code into a gate-level netlist. It's the first step before implementation.

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.

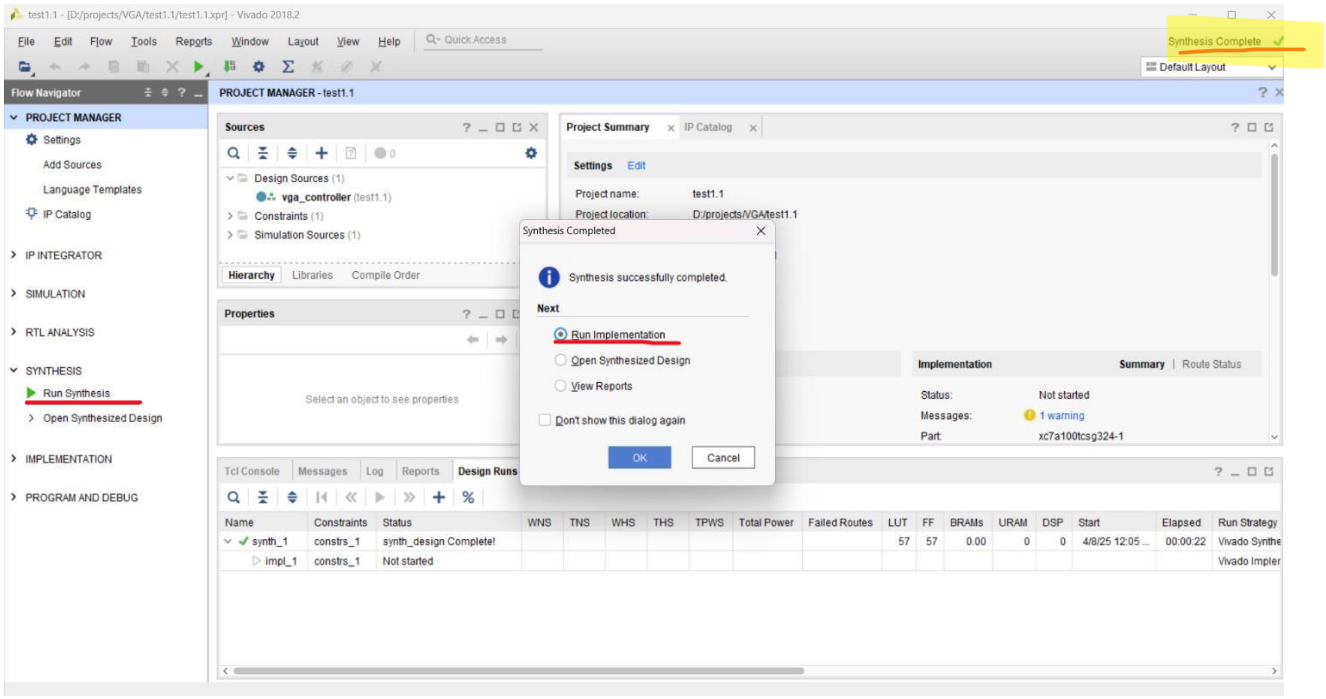


Figure 5.12. Start Synthesis process and monitor the synthesis log

Figure Description:

Illustrates the initiation of the synthesis process for the design. It also depicts the subsequent monitoring of the synthesis log for progress and potential issues.

2. 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.

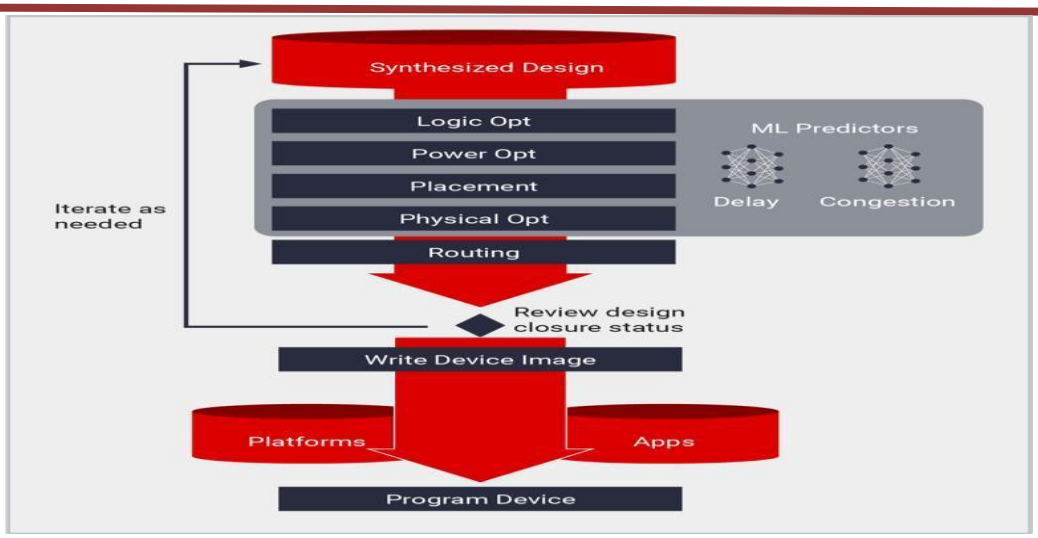


Figure 5.13. Flow of Implementation

Figure Description:

Illustrates placement and routing of logic on FPGA. Ensures the design is physically mapped for hardware execution**.**

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

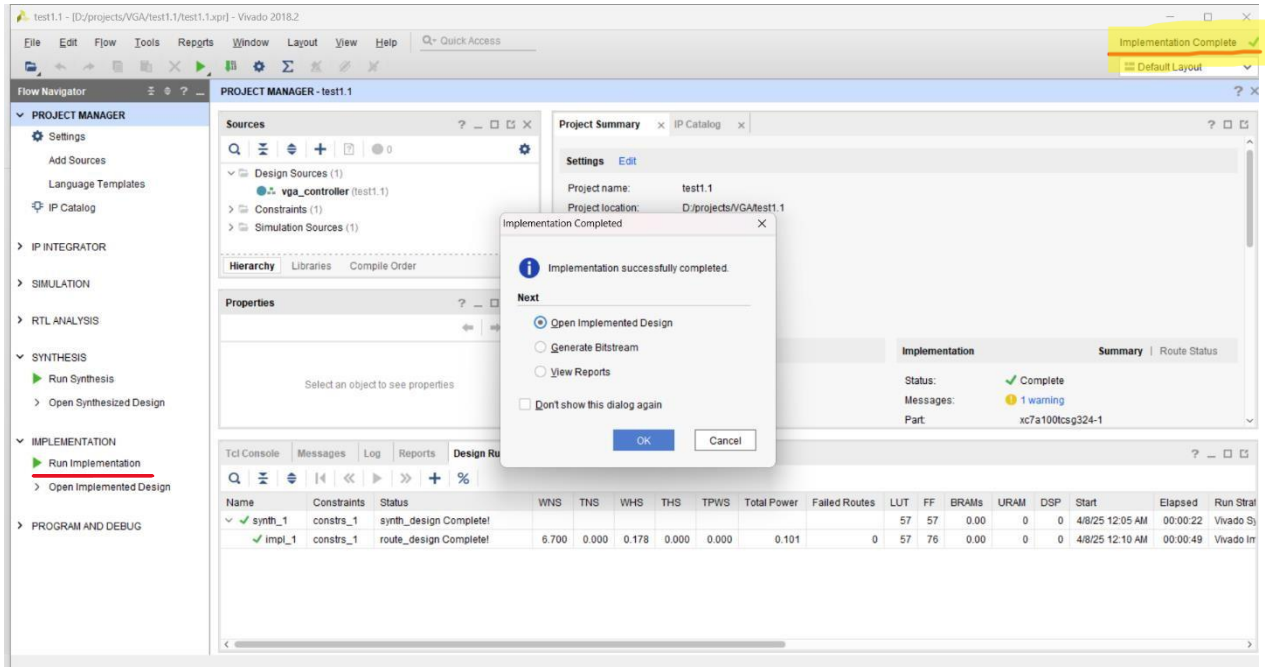


Figure 5.14. Start Implementation process and monitor the implementation log

Figure Description:

shows the commencement of the implementation process for the design. It further depicts the necessary monitoring of the implementation log to track progress and identify any errors.

3. Generate Bitstream.

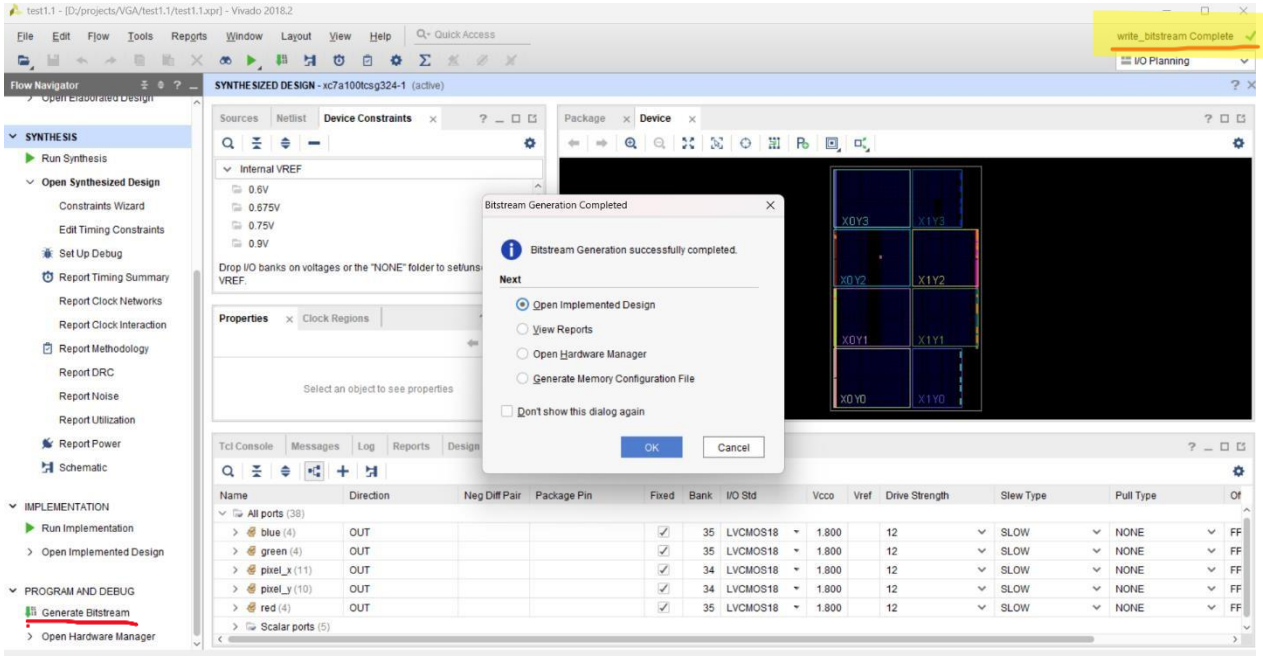


Figure 5.15. Generate Bitstream

Figure Description:

Depicts the final stage of the design flow: bitstream generation. This process creates the configuration file that can be loaded onto the target hardware.

CHAPTER 6
RESULTS

6.1 SIMULATION RESULTS

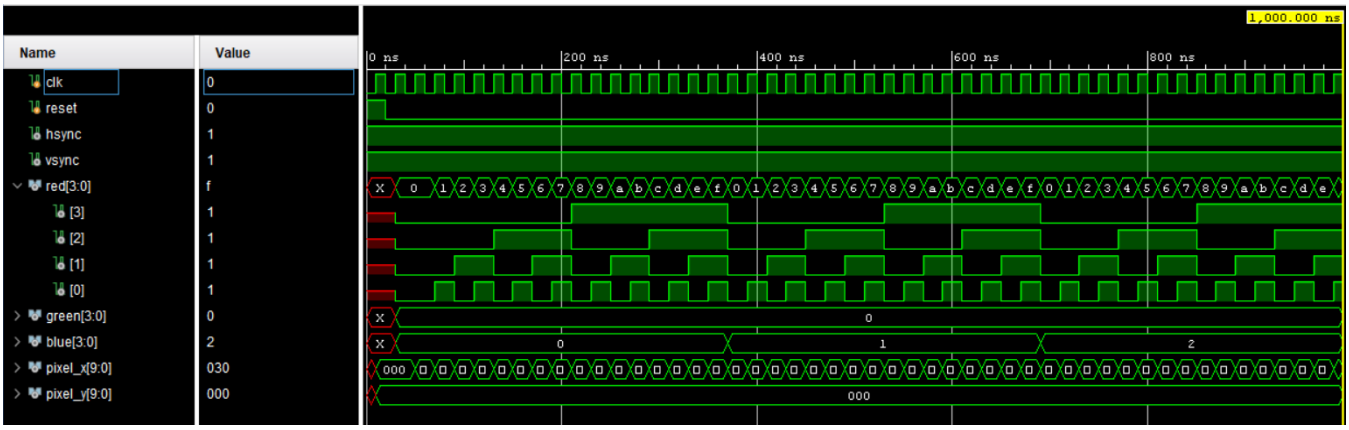


Fig6.1 Simulation output

Figure Description And I/P's:

Input signal:

Signal	Value at 1,000,000 ns
clk	0
reset	1

Output signal:

Signal	Value at 1,000,000 ns	Value Type	Description
hsync	1	Binary (Single bit)	Horizontal Synchronization signal
vsync	1	Binary (Single bit)	Vertical Synchronization signal
red[3:0]	f (hexadecimal)	4-bit Vector (Hexadecimal representation)	Red color component
green[3:0]	0 (hexadecimal)	4-bit Vector (Hexadecimal representation)	Green color component
blue[3:0]	2 (hexadecimal)	4-bit Vector (Hexadecimal representation)	Blue color component
pixel_x[9:0]	030 (hexadecimal)	10-bit Vector (Hexadecimal representation)	Horizontal pixel coordinate
pixel_y[9:0]	000 (hexadecimal)	10-bit Vector (Hexadecimal representation)	Vertical pixel coordinate

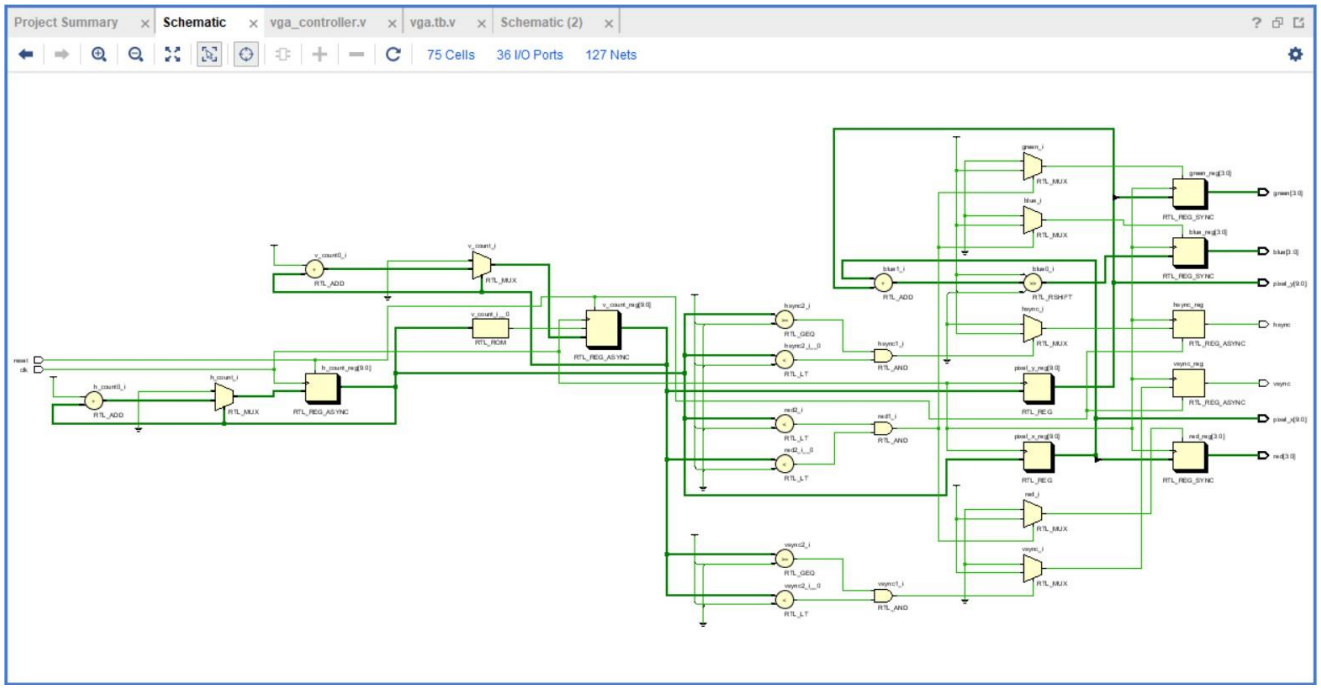


Fig 6.2 schematic

Figure Description:

This image shows a schematic of a digital circuit, likely part of a VGA controller, visually representing interconnected logic gates (AND, OR, inverters) and registers (like FDRE). It displays input/output ports, with this section appearing to process pixel coordinate-related inputs to generate the red color component (red_o[3:0]). Schematics are crucial for designing, understanding, implementing, verifying, documenting, and synthesizing digital circuits for hardware like PCBs or FPGAs. In this VGA controller, it's essential for generating the correct red intensity for each pixel.

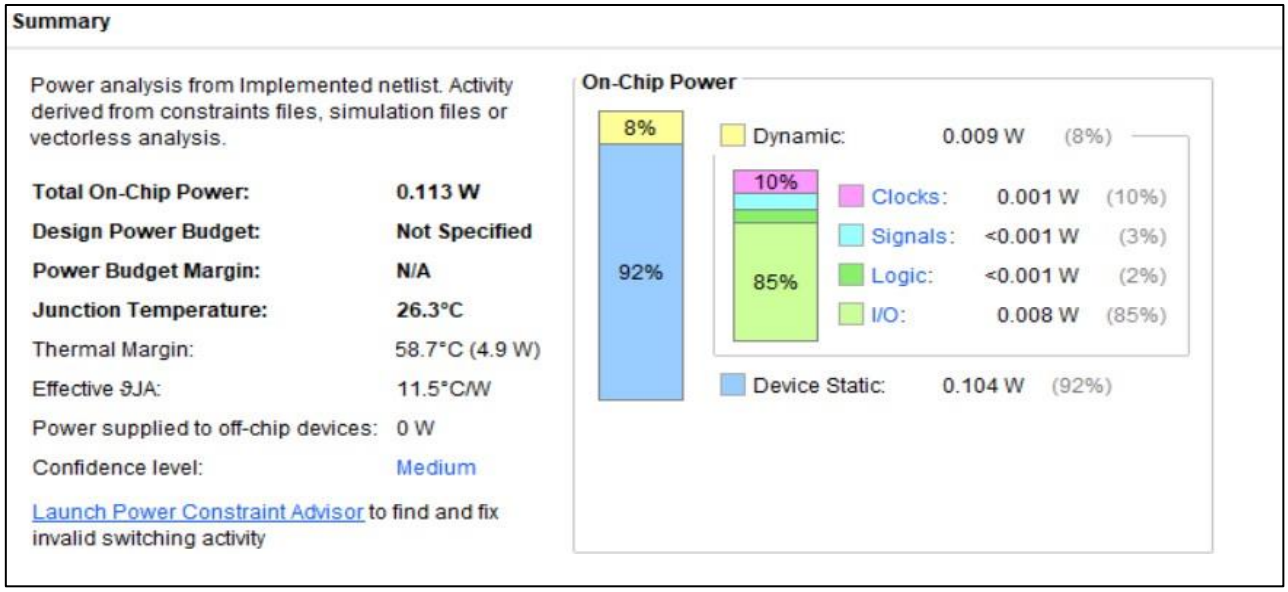


Fig 6.3 : Final Power

Figure Description:

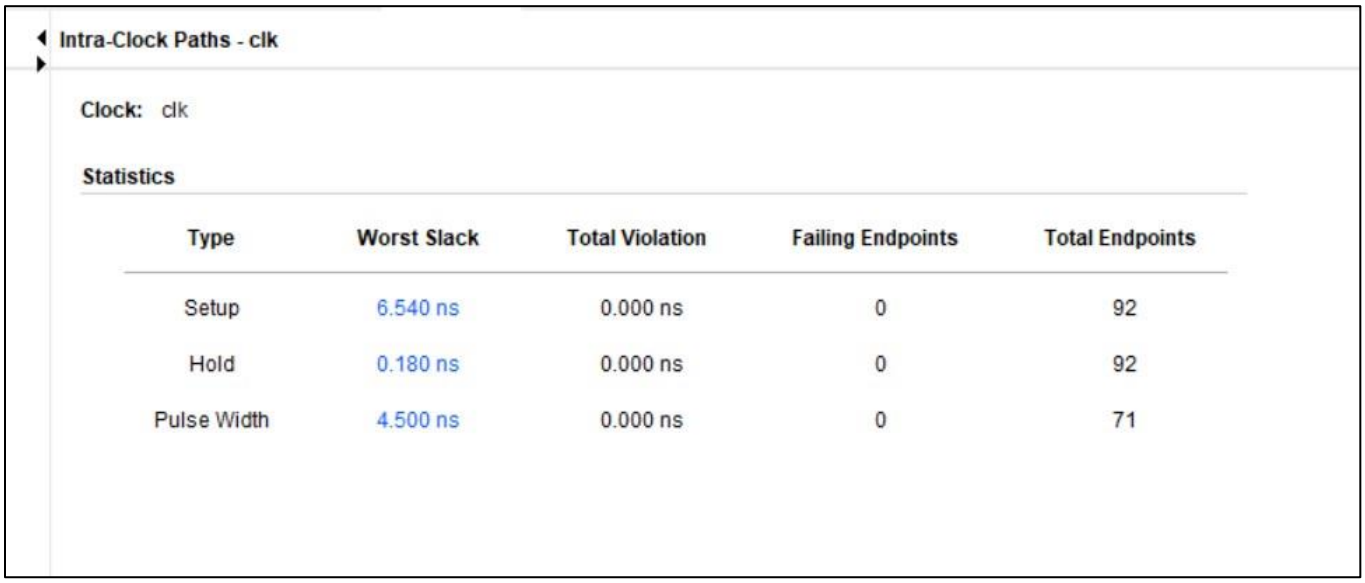
This report represents the on-chip power consumption of the implemented VGA controller design in Xilinx Vivado. The total on-chip power is 0.113 W, which is very low, making the design highly efficient and suitable for embedded systems.

- Device Static Power: 0.104 W (92% of total power)
- Dynamic Power: Only 0.009 W (8% of total power)

  I/O: 0.008 W (85% of dynamic power)

  Clocks, Signals, Logic: Each contributes < 0.001 W

The extremely low dynamic power usage (just 0.009 W) and overall power (only 0.113 W) show that the design is optimized and energy-efficient—ideal for power-constrained applications.



Intra-Clock Paths - clk

Clock: clk

Statistics

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	6.540 ns	0.000 ns	0	92
Hold	0.180 ns	0.000 ns	0	92
Pulse Width	4.500 ns	0.000 ns	0	71

Fig 6.4 : Timing Analysis Report

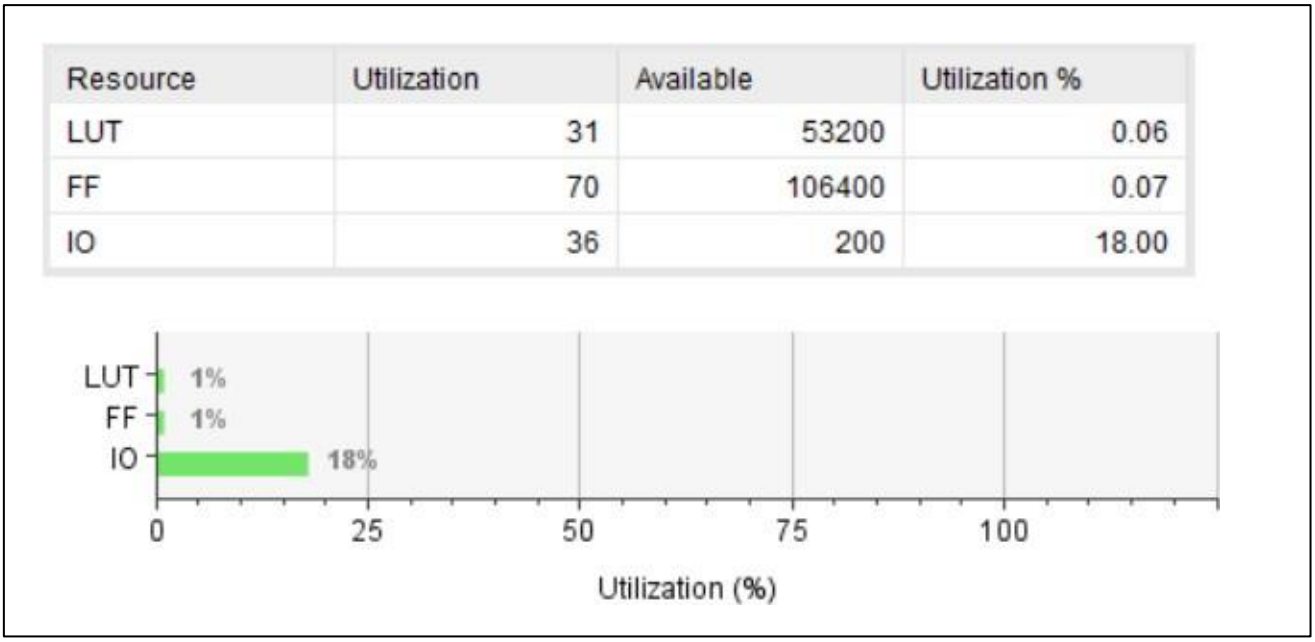
Figure Description:

This report analyzes critical timing constraints such as setup, hold, and pulse width to ensure reliable system operation.

Key Observations:

- Setup Slack: 6.540 ns
- Hold Slack: 0.180 ns
- Pulse Width Slack: 4.500 ns
- Total Violations: 0 for all paths
- Failing Endpoints: 0

There are no violations, and all slacks are positive, indicating the design is timing-safe and meets all constraints. The healthy setup slack of 6.540 ns ensures stable data transfer without timing failures.



Resource	Utilization	Available	Utilization %
LUT	31	53200	0.06
FF	70	106400	0.07
IO	36	200	18.00

Fig 6.5 : Utilization Report

Figure Description:

The design shows minimal usage of internal FPGA resources:

- LUT Utilization: 31 out of 53,200 → 0.06%
- Flip-Flops (FF): 70 out of 106,400 → 0.07%
- I/O Pins: 36 out of 200 → 18.00%

LUTs and FFs are used sparingly, indicating a highly optimized and lightweight design. I/O utilization is relatively higher but still within safe limits, reflecting the VGA interface's dependency on external connections.

CHAPTER 7
ADVANTAGES AND APPLICATIONS

Advantages and applications

7.1 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.

7.2 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 8

CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION

The VGA controller was successfully designed and implemented using Xilinx Vivado with Verilog HDL, enabling real-time video signal generation for display applications. The design incorporated precise timing control for synchronization signals and pixel data output, ensuring stable and clear VGA output. Functionality was verified through simulation and testbenches

Key achievements of this project include:

- **Precise Timing Control**: The controller accurately generated horizontal and vertical synchronization signals, adhering to VGA standards for 640×480 resolution at 60Hz.
- **Efficient Resource Utilization**: Synthesis reports indicated minimal usage of FPGA resources, with low utilization percentages for Look-Up Tables (LUTs) and Flip-Flops (FFs), demonstrating the design's efficiency.
- **Low Power Consumption**: Power analysis revealed that the design consumed only 0.113 W of on-chip power, making it suitable for power-sensitive applications.
- **Timing Optimization**: Timing analysis showed positive slack values with no violations, confirming that the design met all timing constraints and operated reliably.
- **Scalability and Flexibility**: The modular design approach allows for easy scalability to support higher resolutions or additional features in future enhancements.

This project demonstrates the feasibility and effectiveness of implementing a VGA controller on FPGA hardware, providing a foundation for further developments in embedded display systems and educational tools.

8.2 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

I. **Renuka A. Wasu, Vijay R. Wadhankar**, "Design and Implementation of VGA Controller on FPGA," *International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)*, Vol. 3, Issue 7, July 2015, ISSN: 2320-9801.

II. **Fangqin Ying, Xiaoqing Feng**, "Design and Implementation of VGA Controller Using FPGA," *International Journal of Advancements in Computing Technology (IJACT)*, Vol. 4, No. 17, September 2012, pp. 458–465.

III. **Hongyan Dong, Hongmin Guo**, "Design of VGA Display Controller Based on FPGA and VHDL," *IEEE*, 2011.

IV. **S. R. Patil, M. V. Dhamane**, "FPGA-Based VGA Controller Implementation for Image Display Applications," *International Journal of Engineering Research and Technology (IJERT)*, Vol. 3, Issue 5, May 2014.

V. **A. R. Bhagat, N. M. Patel**, "Implementation of VGA Controller using FPGA for Educational Purposes," *International Journal of Research in Applied Science and Engineering Technology (IJRASET)*, Vol. 4, Issue 6, June 2016.

VI. **B. Yuvaraj, S. Muruganand**, "Design and Implementation of VGA Controller using FPGA," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)*, Vol. 5, Issue 3, March 2016, ISSN (Online): 2278–8875.

VII. **V. Ashwini, T. Ranjith Kumar**, "Real-Time Image Display using FPGA VGA Controller," *International Journal of Computer Applications (IJCA)*, Volume 180 – No.47, June 2018, ISSN: 0975–8887.

VIII. **Shashank Gupta, Rahul Mital**, "FPGA Based VGA Controller Design and Implementation," *International Journal of Engineering Research and Applications (IJERA)*, Vol. 3, Issue 1, Jan-Feb 2013, ISSN: 2248-9622.