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,

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,
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

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 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**.

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.

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.

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.

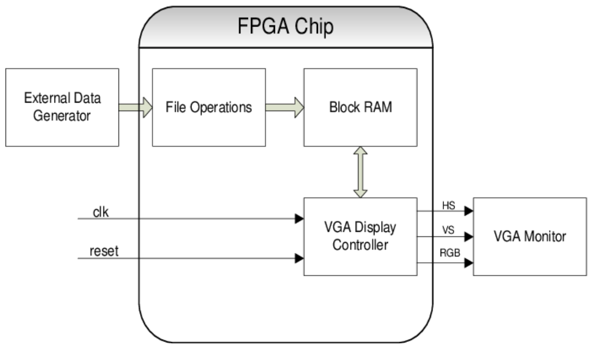
2.2.4 SRAM-Based VGA Controller:

- 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.

CHAPTER 3
PROJECT METHODOLOGY

3. PROJECT METHODOLOGY

3.1 Block Diagram



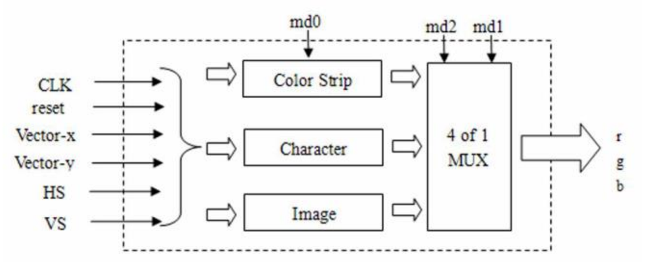


Fig 3.1 Block diagram of VGA Controller in Xilinx Vivado

3.1.1 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**.**

3.1.2 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.

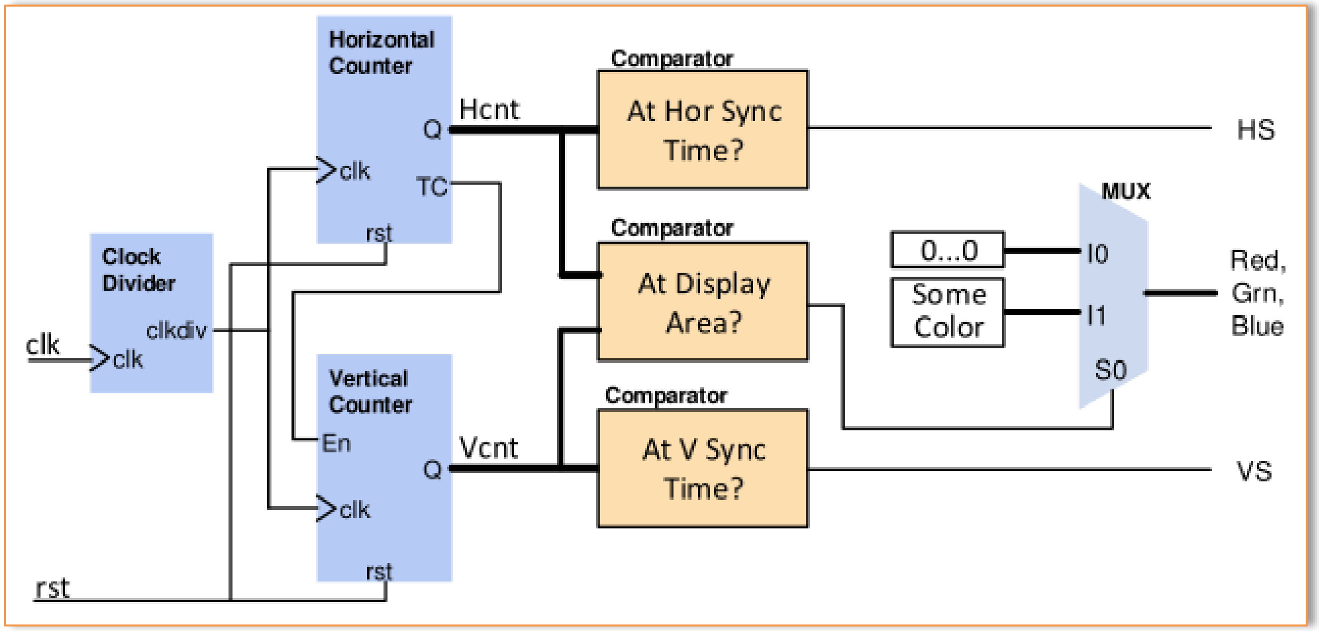


Fig 3.2 Detailed Design of VGA Controller Module

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 & VLSI

1. CONCEPTS 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.

4.1 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.

4.2 CONCEPT VLSI:

4.2.1 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.

4.2.2 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.

4.2.3 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

4.2.4 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.

4.2.5 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.

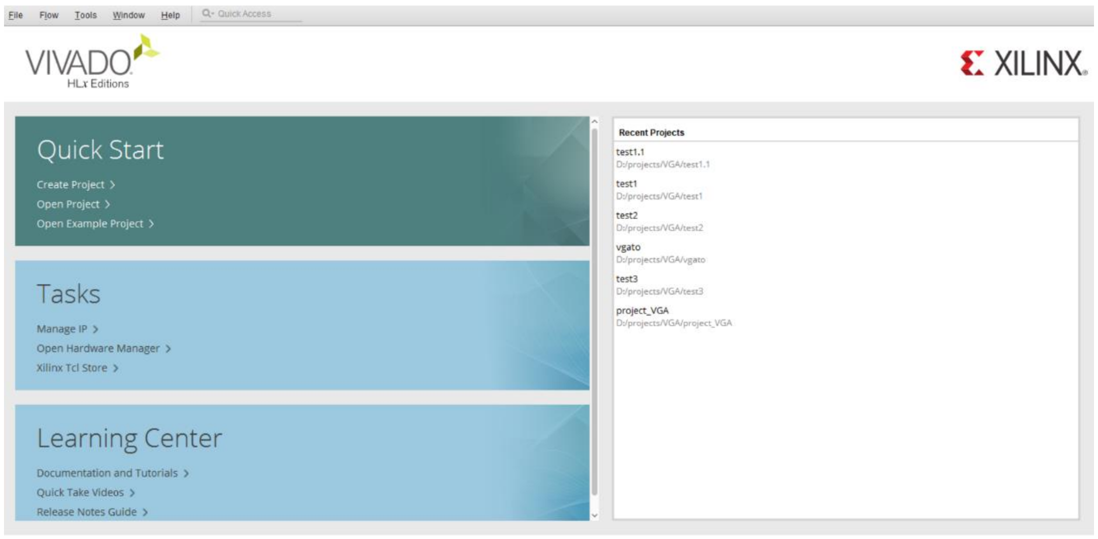
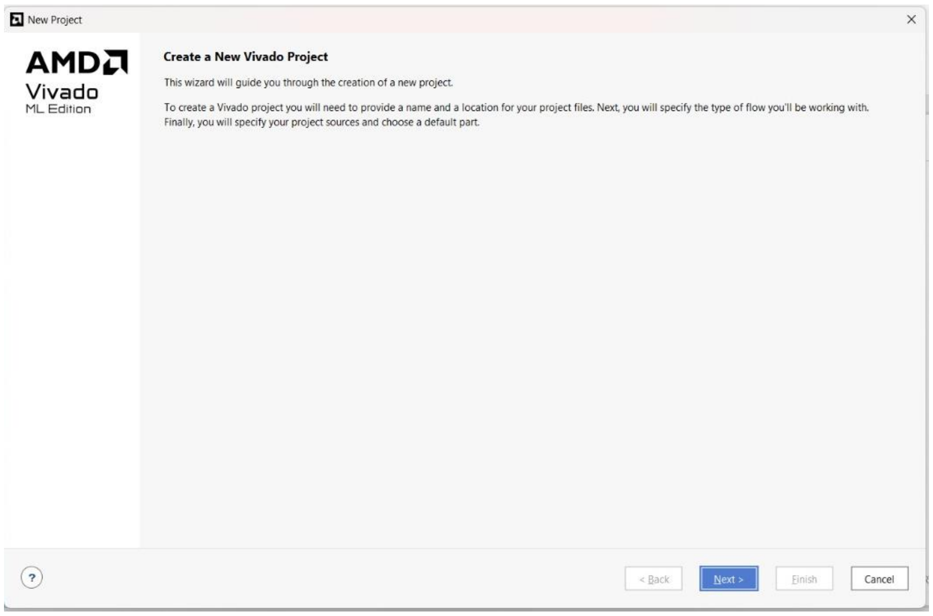


Figure 5.1 Vivado Start-Up Window

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.


New Project

AMD
Vivado
ML Edition

Create a New Vivado Project

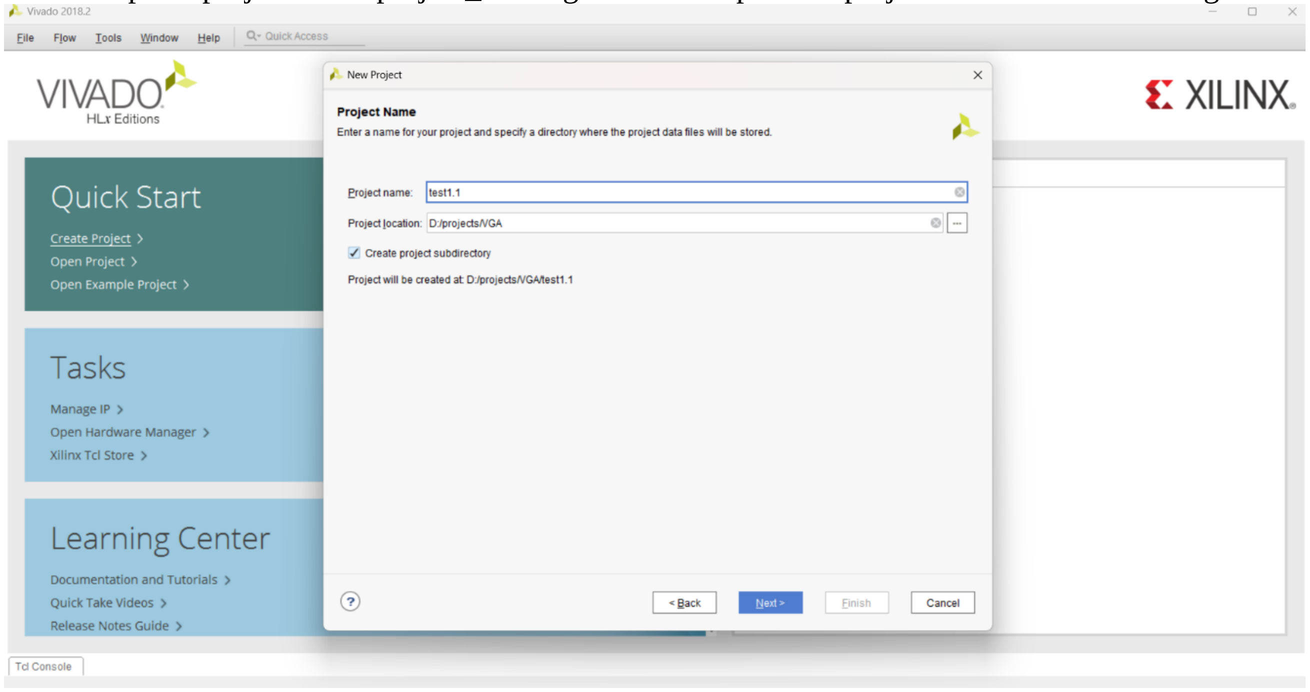
This wizard will guide you through the creation of a new project.

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

Figure 5.2 Create Project Dialog

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.



Project Name

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

Project name: test1.1

Project location: D:/projects/VGA

✓ Create project subdirectory

Project will be created at: D:/projects/VGA/test1.1

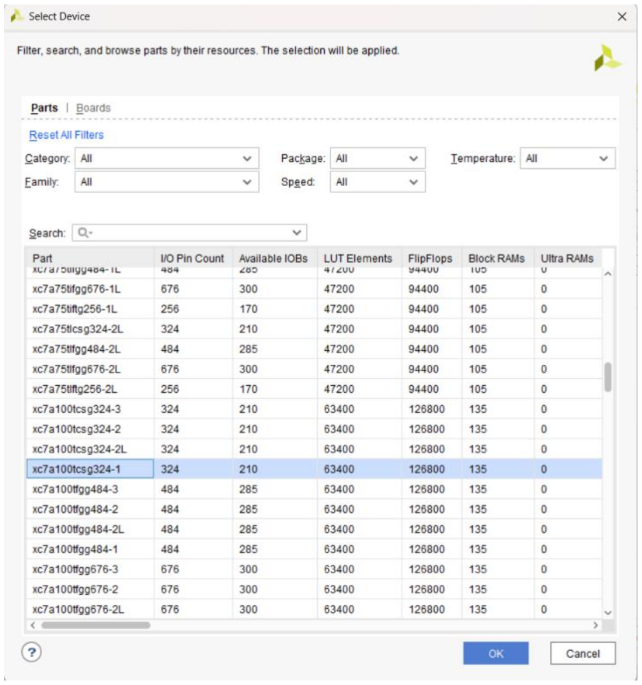
Figure 5.3. Enter Project Name

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**


Select Device

Filter, search, and browse parts by their resources. The selection will be applied.

Parts | Boards

Reset All Filters

Category: All   Package: All   Temperature: All
Family: All   Speed: All

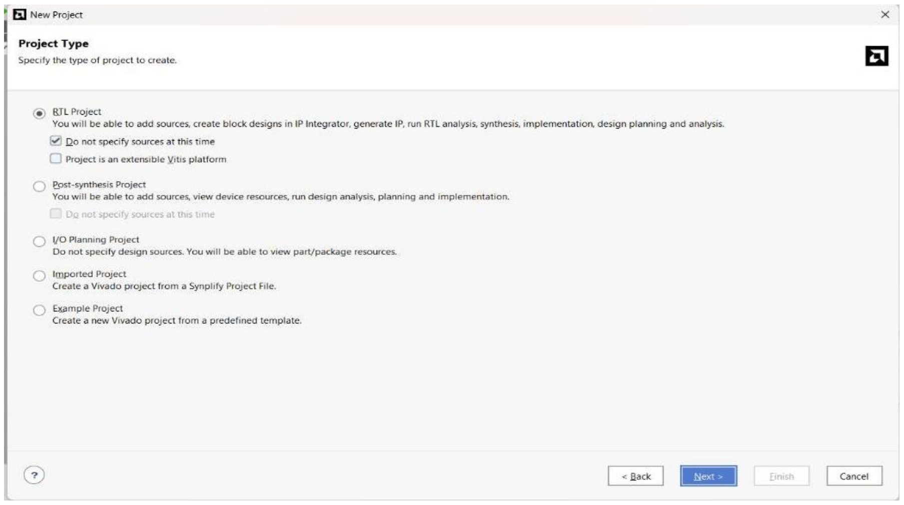
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs
xc7a75tifgg484-1L	484	285	47200	94400	105	0
xc7a75tifgg676-1L	676	300	47200	94400	105	0
xc7a75tiftg256-1L	256	170	47200	94400	105	0
xc7a75tlcsg324-2L	324	210	47200	94400	105	0
xc7a75tlfgg484-2L	484	285	47200	94400	105	0
xc7a75tlfgg676-2L	676	300	47200	94400	105	0
xc7a75tlftg256-2L	256	170	47200	94400	105	0
xc7a100tcsg324-3	324	210	63400	126800	135	0
xc7a100tcsg324-2	324	210	63400	126800	135	0
xc7a100tcsg324-2L	324	210	63400	126800	135	0
xc7a100tcsg324-1	324	210	63400	126800	135	0
xc7a100tfgg484-3	484	285	63400	126800	135	0
xc7a100tfgg484-2	484	285	63400	126800	135	0
xc7a100tfgg484-2L	484	285	63400	126800	135	0
xc7a100tfgg484-1	484	285	63400	126800	135	0
xc7a100tfgg676-3	676	300	63400	126800	135	0
xc7a100tfgg676-2	676	300	63400	126800	135	0
xc7a100tfgg676-2L	676	300	63400	126800	135	0

Figure 5.4. Select Board Evaluation and development kit

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.


New Project

Project Type

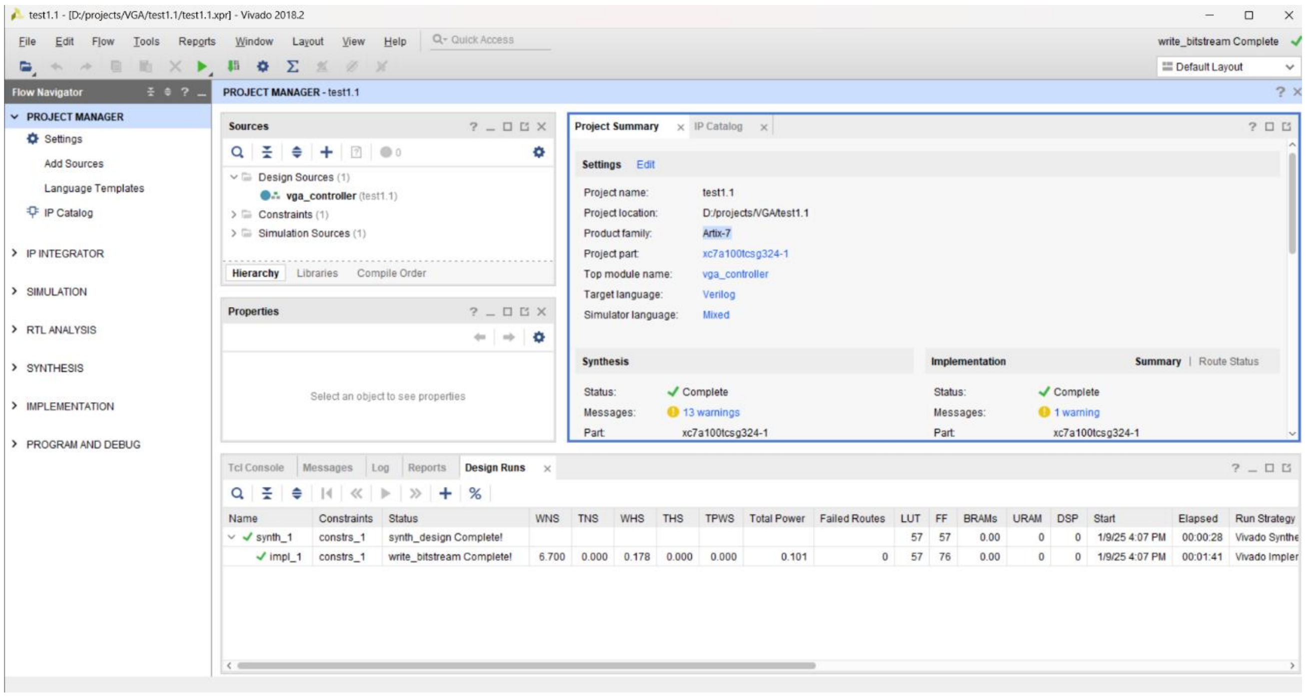
Specify the type of project to create.

- [✓ RTL Project]
  You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
  - [✓ Do not specify sources at this time]
  - Project is an extensible Vitis platform
- Post-synthesis Project
  You will be able to add sources, view device resources, run design analysis, planning and implementation.
  - Do not specify sources at this time
- I/O Planning Project
  Do not specify design sources. You will be able to view part/package resources.
- Imported Project
  Create a Vivado project from a Synplify Project File.
- Example Project
  Create a new Vivado project from a predefined template.

Figure 5.5. Create Project Summary

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.



Settings Edit

Project name:	test1.1
Project location:	D:/projects/VGA/test1.1
Product family:	Artix-7
Project part:	xc7a100tcsg324-1
Top module name:	vga_controller
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	✓ Complete
Messages:	13 warnings
Part:	xc7a100tcsg324-1

Implementation Summary | Route Status

Status:	✓ Complete
Messages:	1 warning
Part:	xc7a100tcsg324-1

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
✓ synth_1	constrs_1	synth_design Complete!								57	57	0.00	0	0	1/9/25 4:07 PM	00:00:28	Vivado Synthe
✓ impl_1	constrs_1	write_bitstream Complete!	6.700	0.000	0.178	0.000	0.000	0.101	0	57	76	0.00	0	0	1/9/25 4:07 PM	00:01:41	Vivado Impler

Figure 5.6 Vivado Project Window

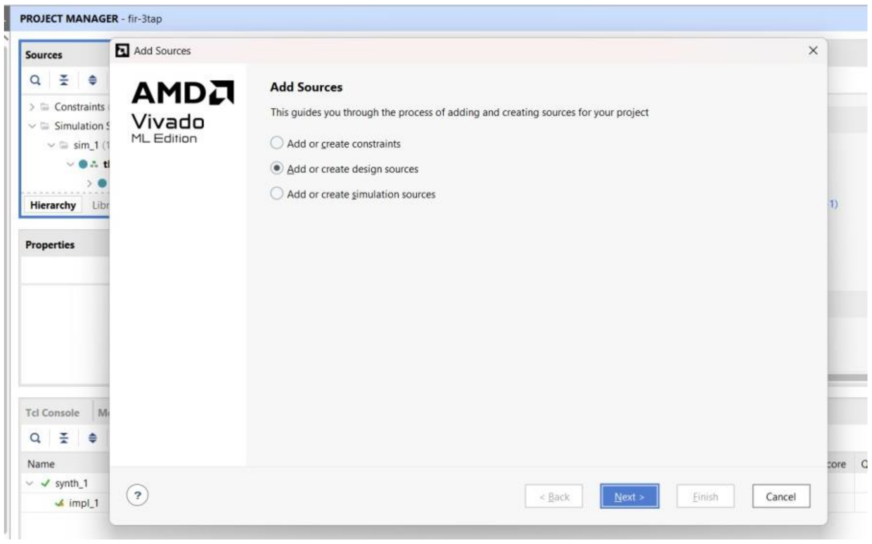
Step 2: Edit The Project - Create source files

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.

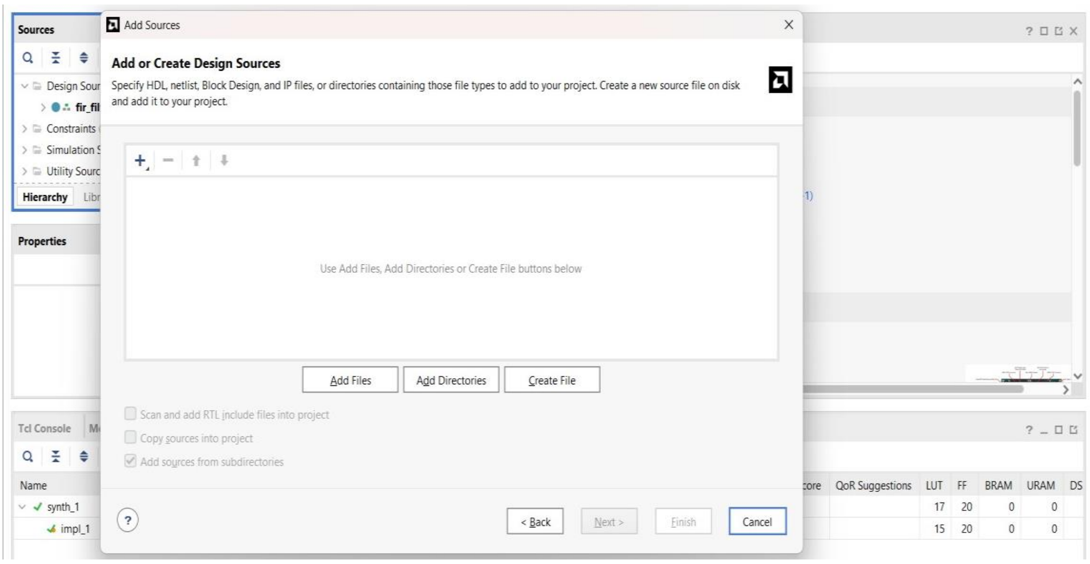
Figure 5.7. Add or create design sources using Add Source Dialog


Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
(●) Add or create design sources
( ) Add or create simulation sources

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.


Add Sources

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [x] Add sources from subdirectories

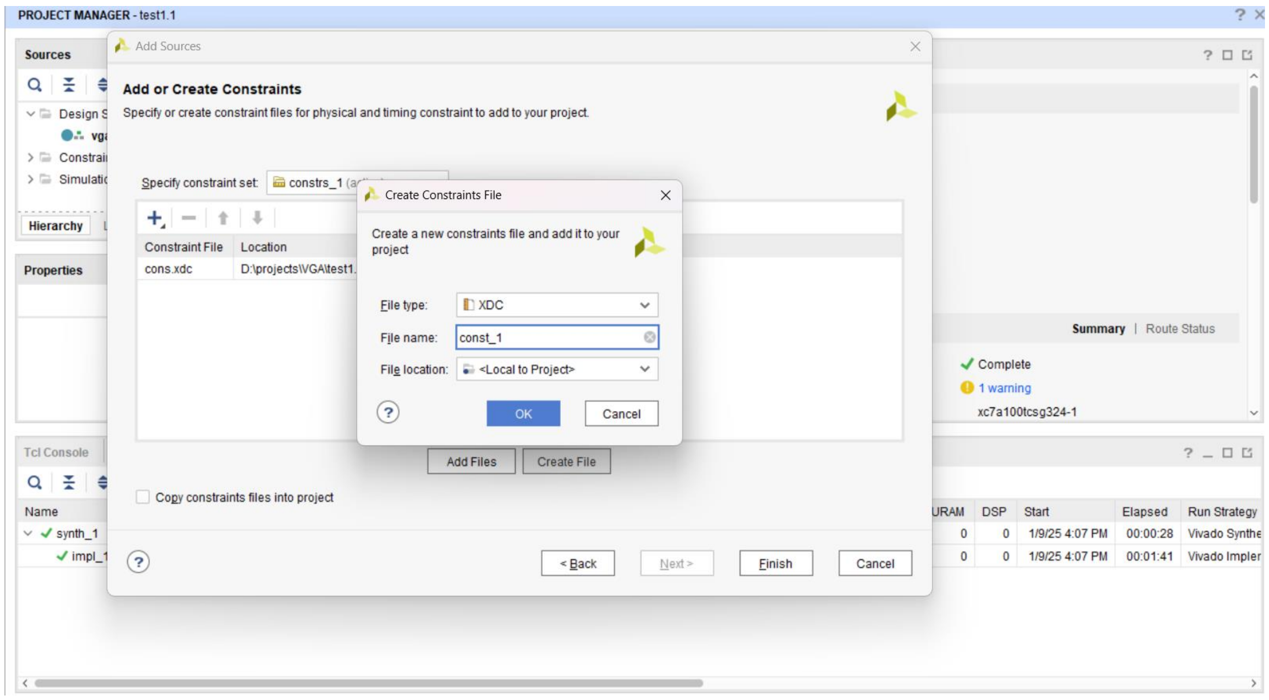
Name	QoR Suggestions	LUT	FF	BRAM	URAM
synth_1		17	20	0	0
impl_1		15	20	0	0

Figure 5.8. Creation of Design Source

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.


PROJECT MANAGER - test1.1

Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Constraint File	Location
cons.xdc	D:\projects\VGA\test1.

Create Constraints File

Create a new constraints file and add it to your project

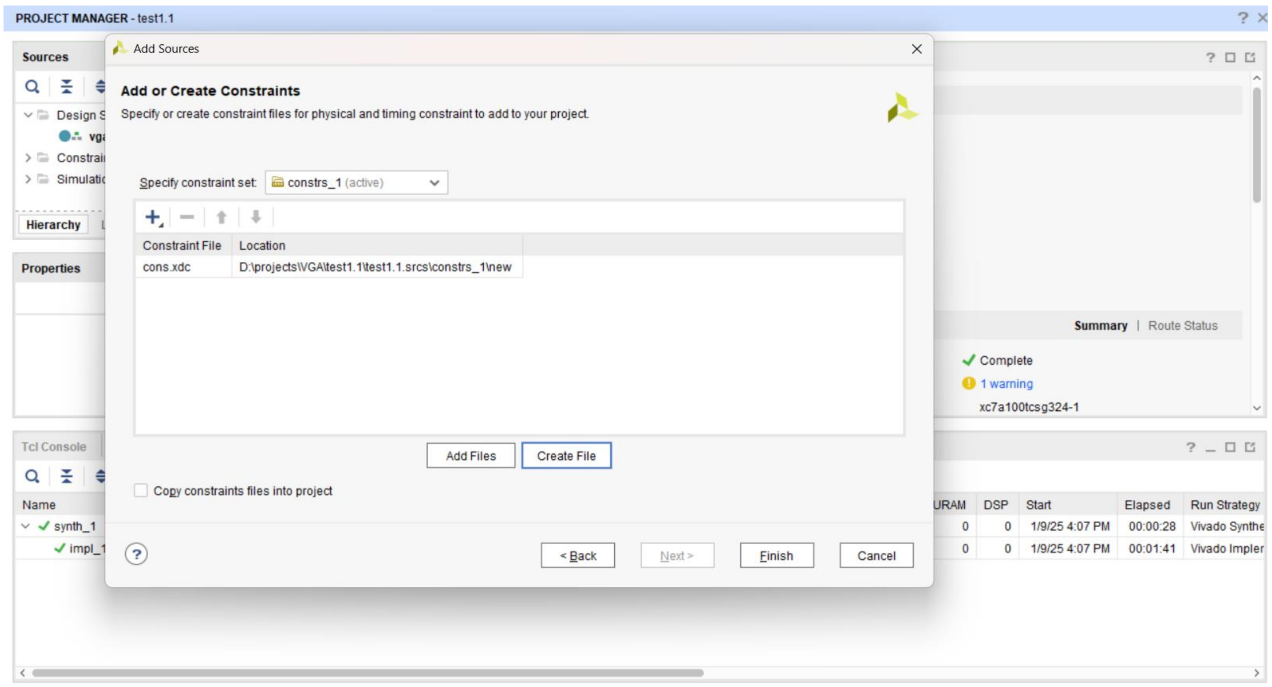
File type: XDC

File name: const_1

File location: <Local to Project>

Figure 5.9. 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.


PROJECT MANAGER - test1.1

Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Constraint File	Location
cons.xdc	D:\projects\VGA\test1.1\test1.1.srcs\constrs_1\new

Figure 5.10. 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.

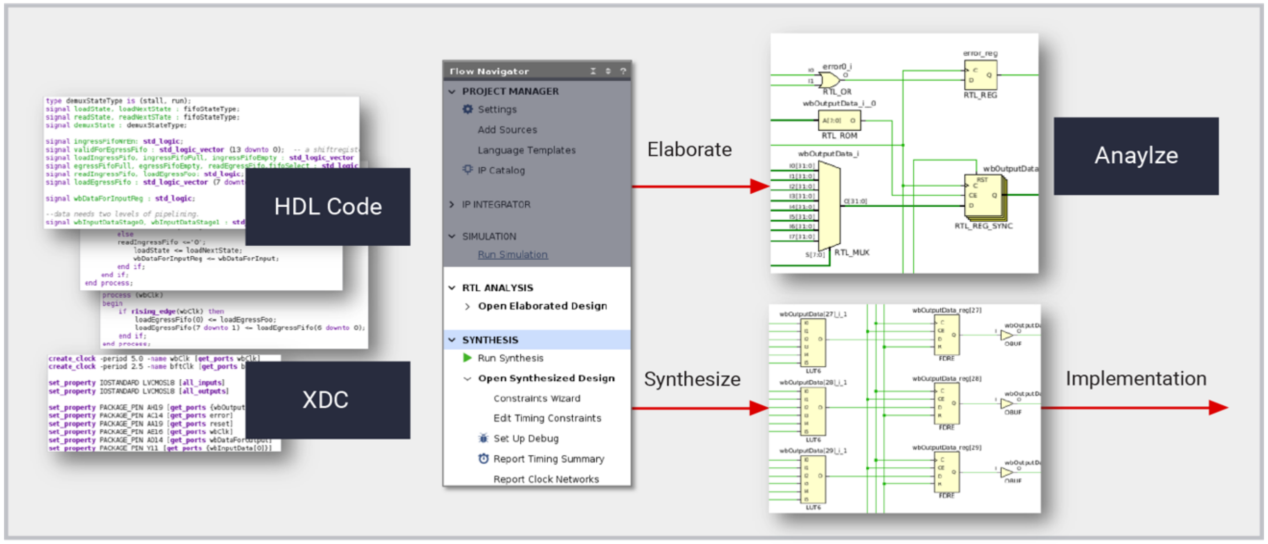
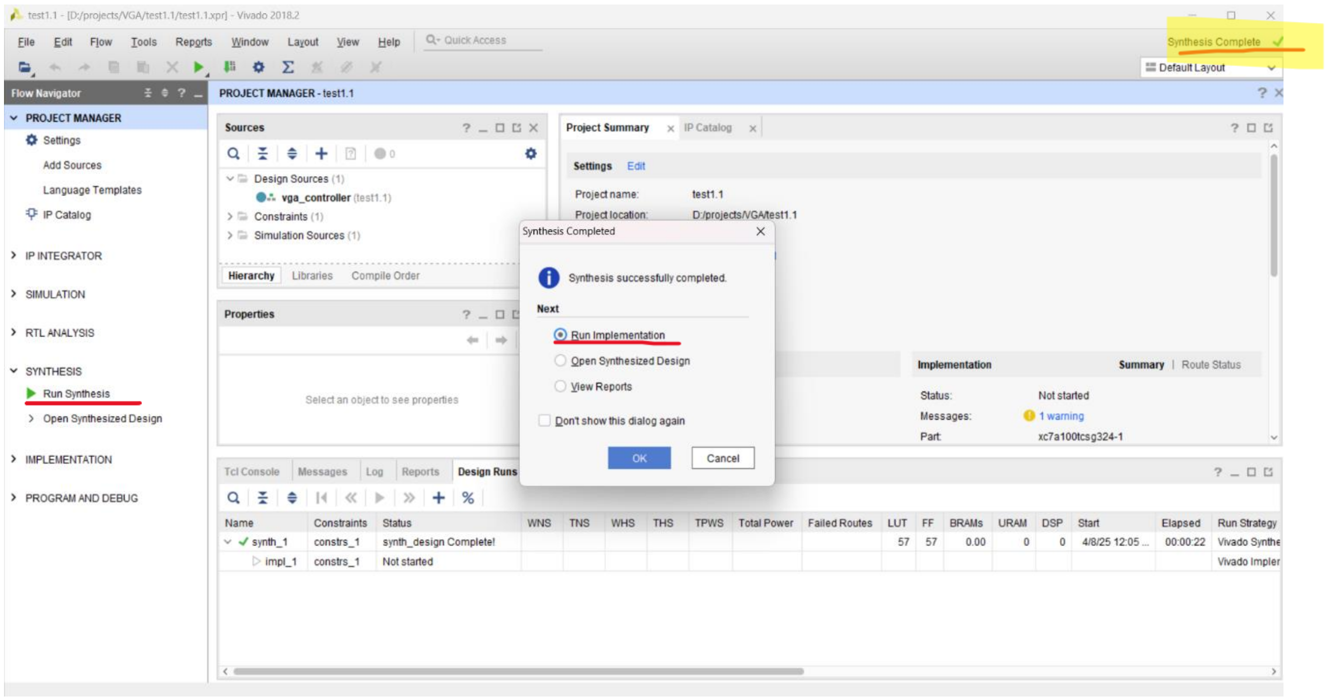


Figure 5.11. Flow Of Synthesis

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.



Settings Edit

Project name: test1.1

Project location: D:/projects/VGA/test1.1

Synthesis Completed

Synthesis successfully completed.

Next

[✓ Run Implementation]

Open Synthesized Design

View Reports

Don't show this dialog again

OK Cancel

Implementation Summary | Route Status

Status: Not started

Messages: 1 warning

Part: xc7a100tcsg324-1

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
✓ synth_1	constrs_1	synth_design Complete!								57	57	0.00	0	0	4/8/25 12:05 ...	00:00:22	Vivado Synthe
impl_1	constrs_1	Not started															Vivado Impler

Figure 5.12. Start Synthesis process and monitor the synthesis log

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.

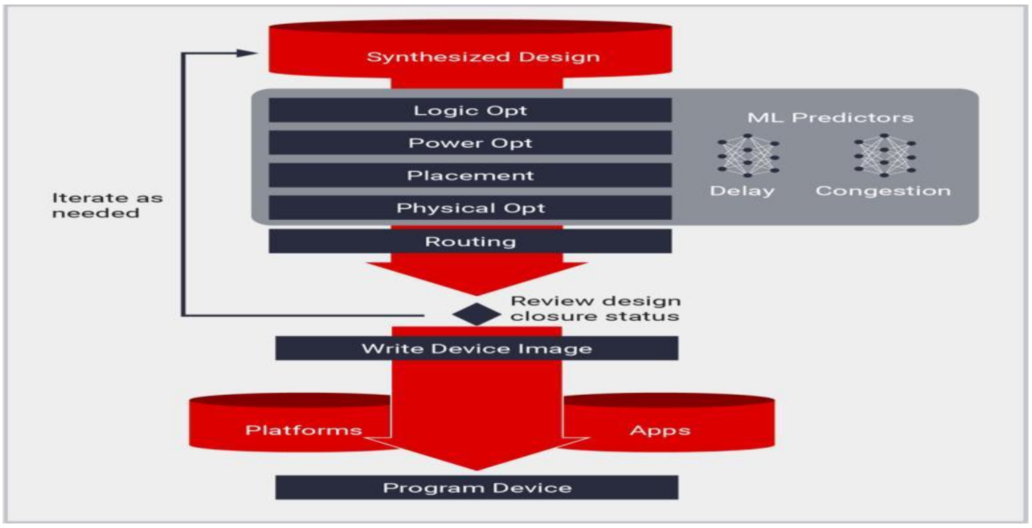
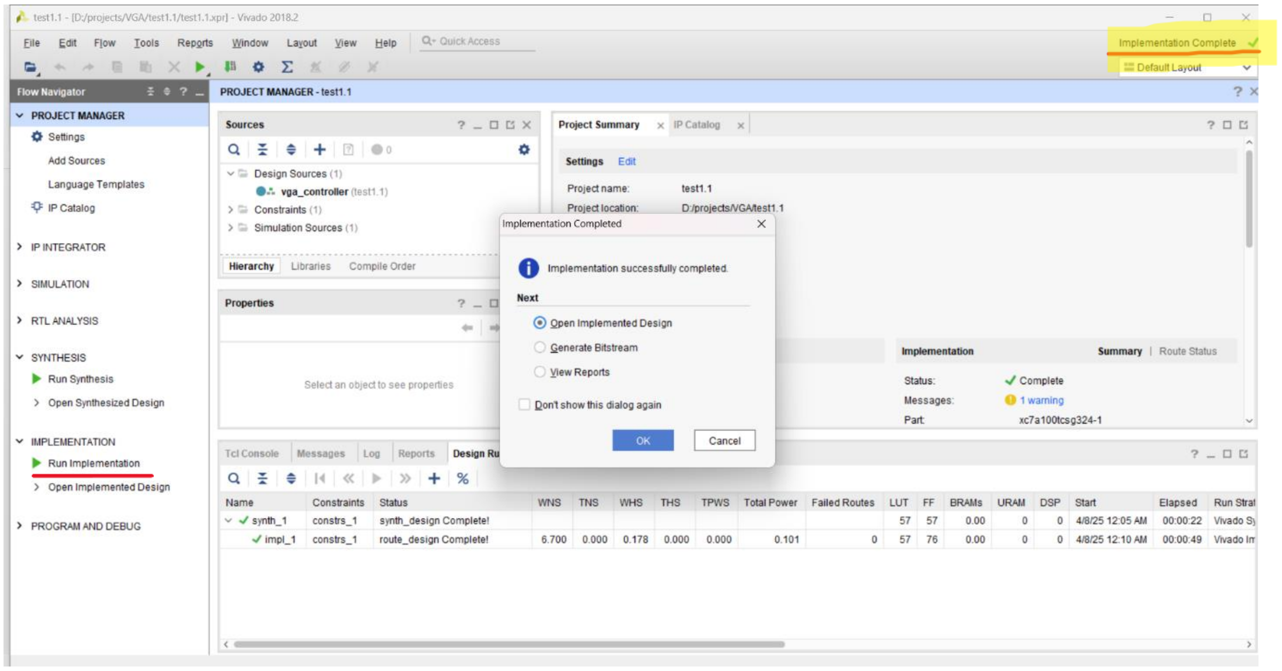


Figure 5.13. Flow of Implementation

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



Implementation successfully completed.

Next
- [✓ Open Implemented Design]
- Generate Bitstream
- View Reports
- Don't show this dialog again

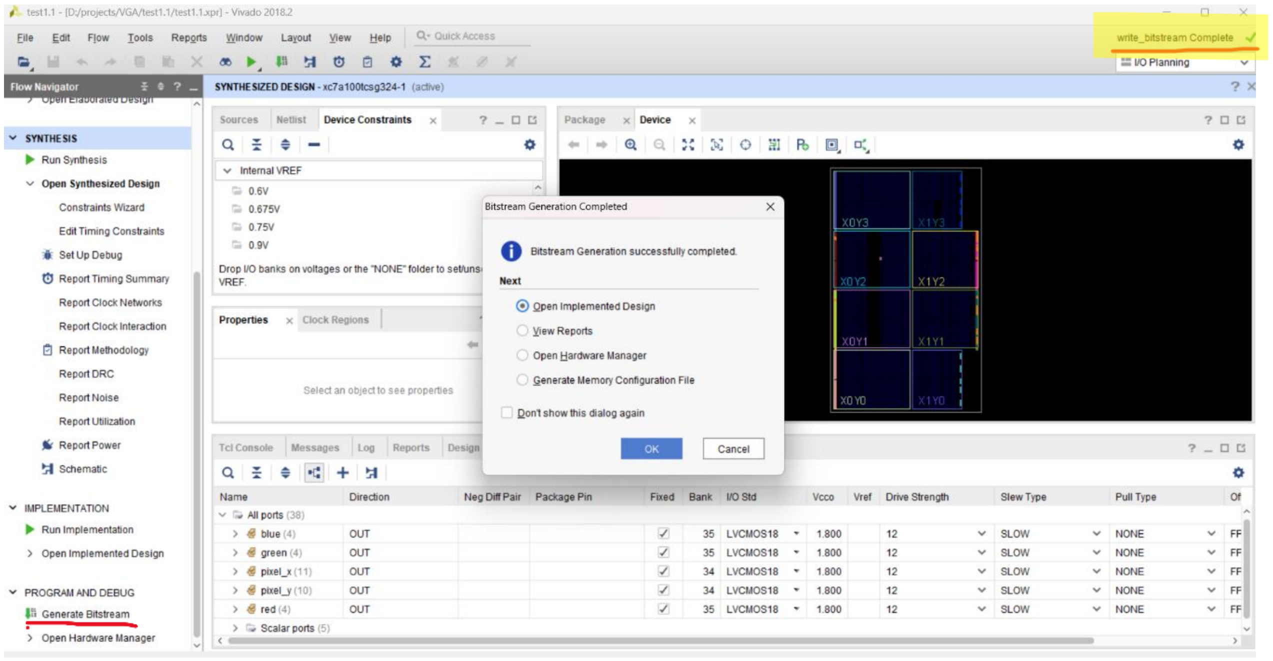
Project name: test1.1
Project location: D:/projects/VGA/test1.1

Status: Complete
Messages: 1 warning
Part: xc7a100tcsg324-1

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strat
synth_1	constrs_1	synth_design Complete!								57	57	0.00	0	0	4/8/25 12:05 AM	00:00:22	Vivado Sy
impl_1	constrs_1	route_design Complete!	6.700	0.000	0.178	0.000	0.000	0.101	0	57	76	0.00	0	0	4/8/25 12:10 AM	00:00:49	Vivado Im

Figure 5.14. Start Implementation process and monitor the implementation log

3. Generate Bitstream.


test1.1 - [D:/projects/VGA/test1.1/test1.1.xpr] - Vivado 2018.2
write_bitstream Complete
SYNTHESIZED DESIGN - xc7a100tcsg324-1 (active)
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Bitstream Generation Completed
X0Y3
X1Y3
X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0
OK
Cancel

Bitstream Generation successfully completed.

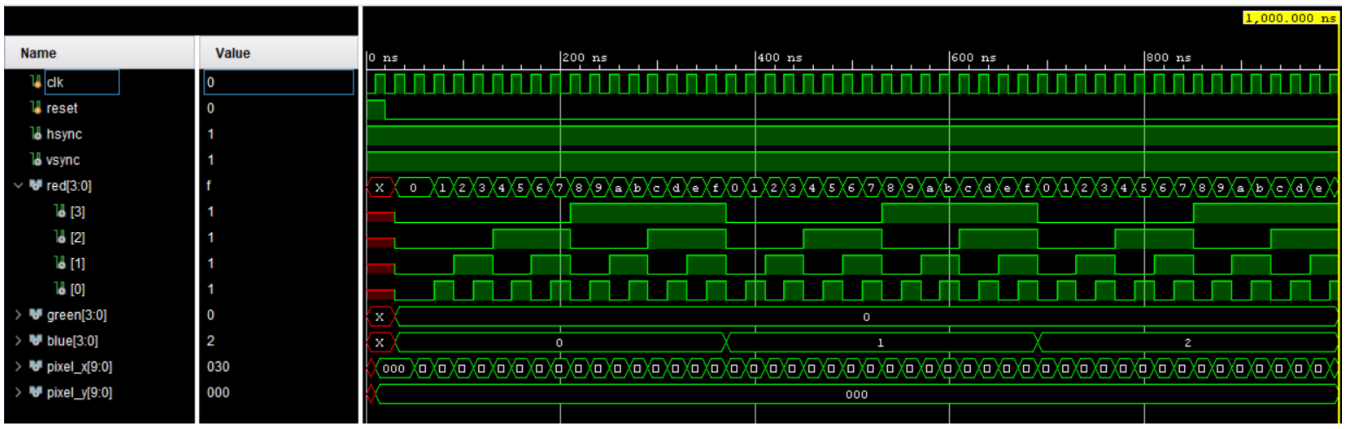
Next
- [✓ Open Implemented Design]
- View Reports
- Open Hardware Manager
- Generate Memory Configuration File
- Don't show this dialog again

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Of
All ports (38)												
blue (4)	OUT			✓	35	LVCMOS18	1.800		12	SLOW	NONE	FF
green (4)	OUT			✓	35	LVCMOS18	1.800		12	SLOW	NONE	FF
pixel_x (11)	OUT			✓	34	LVCMOS18	1.800		12	SLOW	NONE	FF
pixel_y (10)	OUT			✓	34	LVCMOS18	1.800		12	SLOW	NONE	FF
red (4)	OUT			✓	35	LVCMOS18	1.800		12	SLOW	NONE	FF
Scalar ports (5)												

Figure 5.15. Generate Bitstream

CHAPTER 6
RESULTS

6.1 SIMULATION RESULTS



Name	Value
clk	0
reset	0
hsync	1
vsync	1
red[3:0]	f
[3]	1
[2]	1
[1]	1
[0]	1
green[3:0]	0
blue[3:0]	2
pixel_x[9:0]	030
pixel_y[9:0]	000

Fig6.1 Simulation output

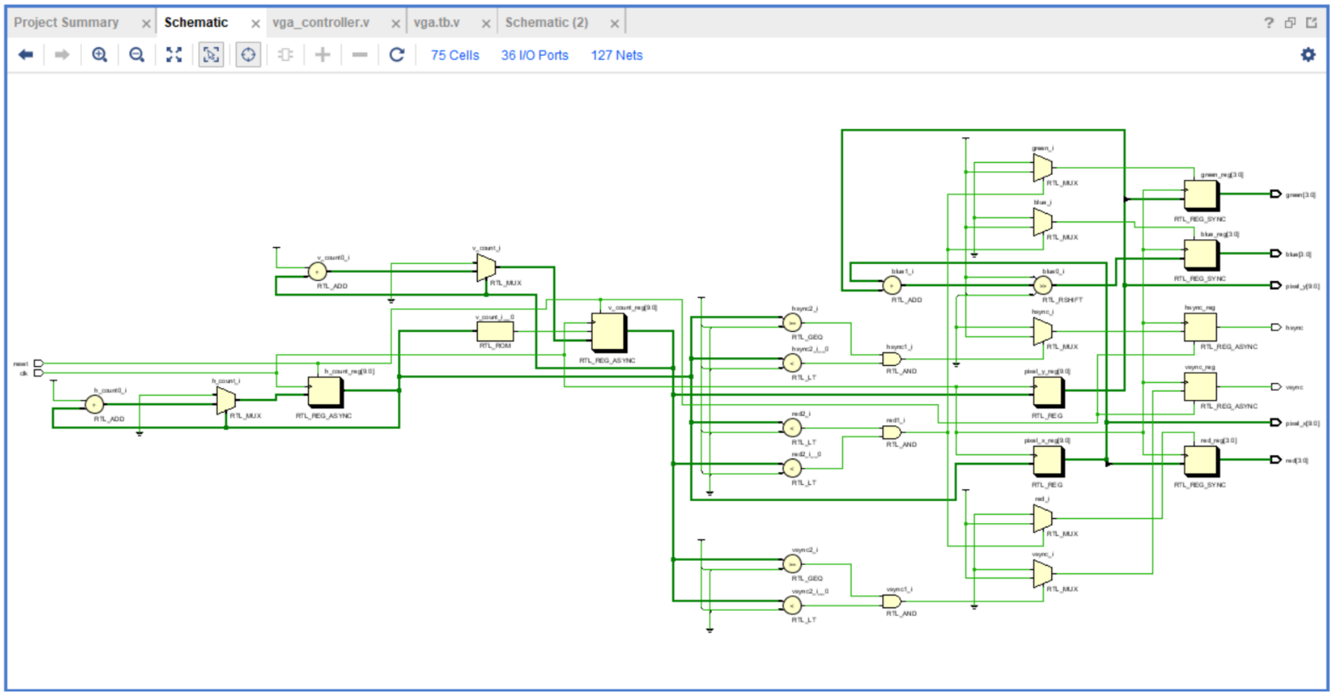
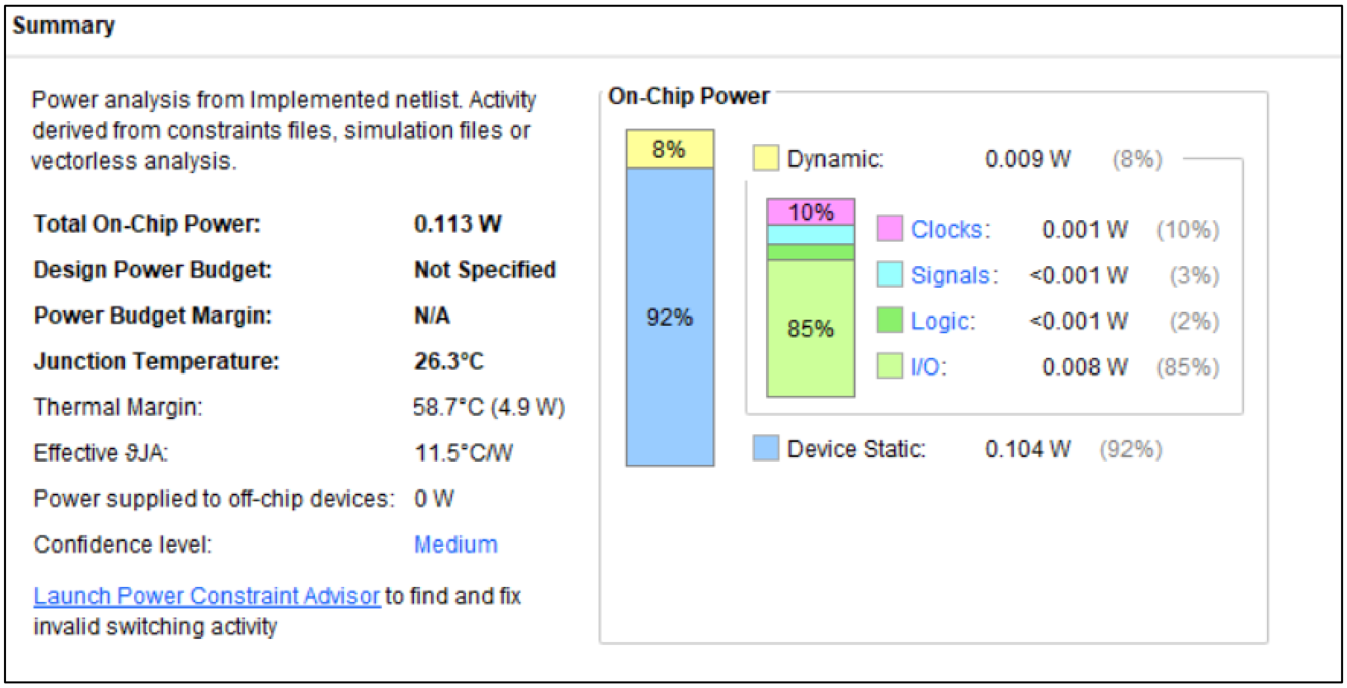


Fig 6.2 schematic



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

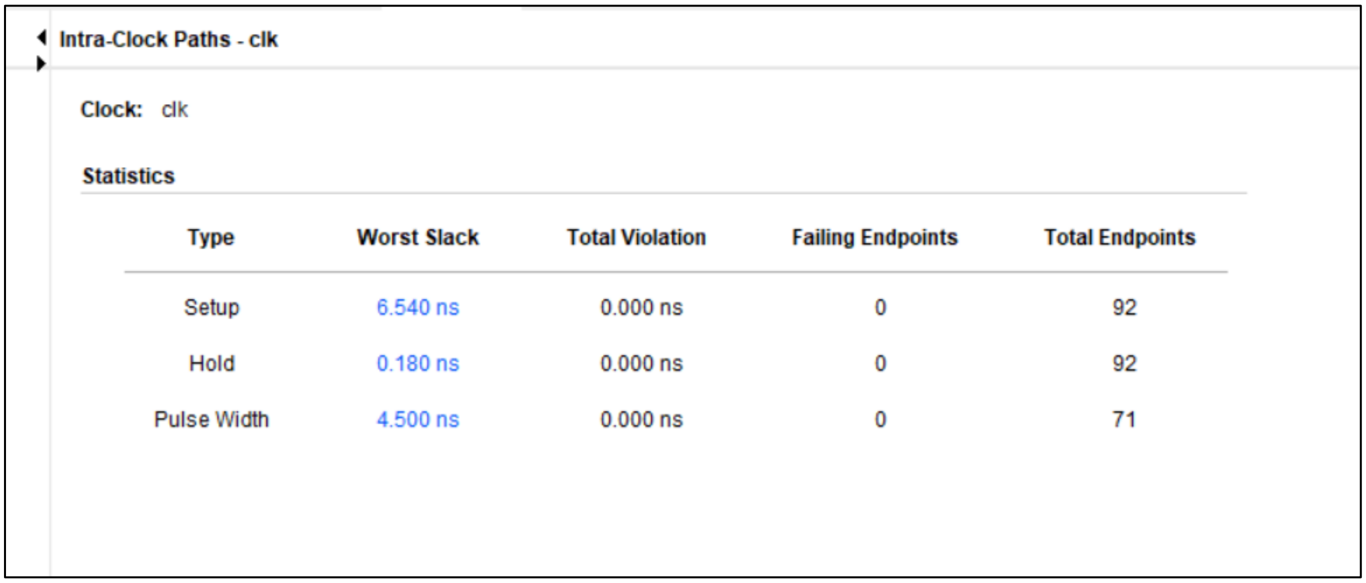
Total On-Chip Power:	**0.113 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.3°C**
Thermal Margin:	58.7°C (4.9 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	0.009 W	(8%)
🟪	Clocks:	0.001 W	(10%)
🟦	Signals:	<0.001 W	(3%)
🟩	Logic:	<0.001 W	(2%)
🟨	I/O:	0.008 W	(85%)
🟦	Device Static:	0.104 W	(92%)

Fig 6.3 : Final Power



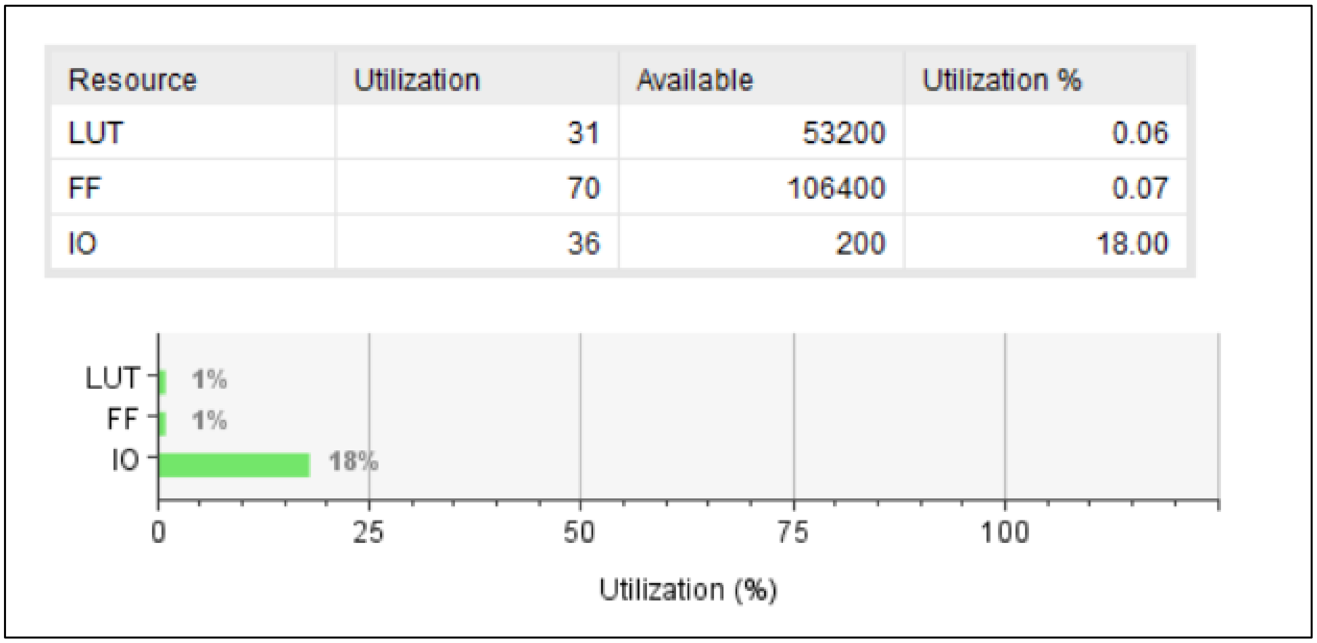
Intra-Clock Paths - clk

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



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

Fig 6.5 : Utilization Report

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, 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.

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. 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

II. L. Y. Akshitha V. Ramesh, Apeksha Ravi Kumar, Amulya S. Iyengar, “Implementation and Design of FIR Filters using Verilog HDL and FPGA,” 2020.

III. 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.

IV. Swati Chulet, Himanshu Joshi “FIR Filter Designing Using Wallace Multiplier”,IJETR ISSN: 2321-0869, Volume-3, Issue-6, June 2015.

V. O. Venkata Krishna “Design and Implementation of FIR Filter using Low Power and High- Speed Multiplier and Adders” CVR Journal ISSN:2277-3916.

VI. G. Appala Naidu, “Implementation of Wallace tree multiplier using Xilinx with Verilog” – IERJ, E-ISSN No: 2454-991, Vol. 9, Issue 8, August 2023.

VII. 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.