DUAL-PORT RAM-BASED IMAGE STORAGE

A Report submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

By

B. DEDEEPYA
20JG1A0410

K. HARIKA
20JG1A0446

CH. BHARGAVI
20JG1A0419

M. LAVANYA
20JG1A0459

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

(Affiliated to Jawaharlal Nehru Technological University Kakinada)

MADHURAWADA, VISAKHAPATNAM-48

(2020-2024)

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the report titled **"Dual-ported RAM-based image storage"** is a bonafide work of the following 1V/1V B-Tech-I Semester students in the Department of Electronics and Communication Engineering during the academic year 2023-2024, in partial fulfillment of the requirement for the award of the degree of bachelor of Technology of Jawaharlal Nehru Technological University, Kakinada.

B. Dedeepya (20JG1A0410)

CH. Bhargavi (20JG1A0419)

K. Harika (20JG1A0446)

M. Lavanya(20JG1A0459)

Faculty Coordinator
Lakshmi

Asst. Professor ,

Department of E.C.E

Head of the Department B.
Dr. P.M.K PRASAD

Assoc. Professor,

Department of E.C.E

External Examiner

ACKNOWLEDGEMENT

We sincerely thank our Faculty Coordinator **B. LAKSHMI**, Asst. Professor, for her guidance and constant encouragement to us at every stage and aspect by including the spirit of understanding and support in carrying out our work at college.

We would like to express sincere thanks to our Head of the Department of Electronics and Communication Engineering **Dr. P.M.K PRASAD** for his valuable suggestions and constant motivation that greatly helped me in completing the r work successfully.

We express sincere thanks to our Vice Principal, Professor **Dr. G. Sudheer,** for his encouragement and co-operation in completion of our work.

We wish to express our deep sense of our gratitude to our Principal, Professor **Dr. R.K. Goswami**, for giving us the opportunity to carry out the project work successfully.

B. Dedeepya (20JG1A0410)

CH. Bhargavi (20JG1A0419)

K. Harika (20JG1A0446)

M. Lavanya (20JG1A0459)

1.VISION:

A dual-ported RAM (Random Access Memory) is a type of memory that allows two independent devices or systems to access it simultaneously. It contains two separate ports that can be accessed independently without interfering with each other's operations. This configuration allows for concurrent read and write operations from two different sources.

The vision of our project is to implement a dual-ported RAM system that takes image hexadecimal pixel values as input. This system is designed to read the pixel values and generate an output file containing the read pixel values. The goal is likely efficient image data storage and retrieval for further processing or analysis. The success of the project would be reflected in its ability to handle image data effectively and provide accurate output.

2.MISSION:

The mission of this project is to create an end-to-end system that seamlessly converts images to hexadecimal pixel values using Python, stores them in a text file, processes the file with a Verilog-based dual-ported RAM system to read the values, and finally, reconstructs the image using Python.

The project aims to demonstrate an integrated image processing pipeline, combining software and hardware components for efficient data storage, retrieval, and reconstruction. Success will be measured by the accuracy of data preservation and the fidelity of image reconstruction throughout the entire process.

3.DESIGNER TOOLS:

3.1) Xilinx ISE 14.7:

For our project using Xilinx ISE 14.7, we can leverage the following designer tools to implement the Verilog-based dual-ported RAM system:

3.1.1) ISE Simulator (ISim):

Description: A built-in simulator for testing and debugging your Verilog code.

Use: Simulate your Verilog code using ISim to verify its functionality, identify issues, and debug any errors.

3.2) Jupyter Notebooks:

Jupyter Notebooks are a community standard for communicating and performing interactive computing. They are a document that blends computations, outputs, explanatory text, mathematics, images, and rich media representations of objects.

JupyterLab is one interface used to create and interact with Jupyter Notebooks.

Although it is possible to use many different programming languages in Jupyter Notebooks, this article will focus on Python, as it is the most common use case. (Among R users, R studio tends to be a more popular choice).

4. ABSTRACT:

This project presents an integrated image processing pipeline, using software tools to efficiently handle image data. The workflow begins with a Python script converting an input image into hexadecimal pixel values, which are stored in a text file. The text file is then processed by a Verilog-based dual-ported RAM system designed using Xilinx ISE 14.7. This system reads the hexadecimal values and produces an output file containing the read pixel values.

The objective is to use dual-ported ram to read image, demonstrating effective data storage and retrieval. The final phase involves utilizing another Python script to interpret the output file and reconstruct the original image. The project aims for accuracy in data preservation and fidelity in image reconstruction.

The integration of Python and Verilog, facilitated by Xilinx ISE tools, showcases a comprehensive approach to image retrieval, encompassing high-level software abstraction. Success in this project is marked by the coherent interplay of these tools.

INDEX

5.INTRODUCTION:

This project delves into the integration of Python and Verilog, facilitated by Xilinx ISE 14.7, to create a comprehensive image processing pipeline. Culminating in an end-to-end solution that not only preserves image data but also provides a foundation for intricate data manipulation.

The workflow begins with a Python script converting an input image into hexadecimal pixel values, which are stored in a text file. The text file is then processed by a Verilogbased dual-ported RAM system designed using Xilinx ISE 14.7. This system reads the hexadecimal values and produces an output file containing the read pixel values. The objective is to use dual-ported ram to read image, demonstrating effective data storage and retrieval. The final phase involves utilizing another Python script to interpret the output file and reconstruct the original image. The project aims for accuracy in data preservation and fidelity in image reconstruction.

The integration of Python and Verilog, facilitated by Xilinx ISE tools, showcases a comprehensive approach to image retrieval, encompassing high-level software abstraction. Success in this project is marked by the coherent interplay of these tools.

5.1) Dual-port RAM

Dual-port RAM is a specialized form of memory that allows for simultaneous independent access to two different sets of data. It consists of a memory array with two separate ports, enabling two distinct operations (read and write) to occur concurrently and independently. Each port has its own set of address and data lines, allowing for simultaneous access without interfering with each other.

This memory architecture facilitates efficient data exchange between two different circuits or modules within a system, enabling one module to write data while another module reads data at the same time. Dual-port RAM is commonly used in scenarios requiring parallel access to memory, such as buffering data in real-time systems, implementing shared memory architectures, or facilitating communication between different components within a hardware design.

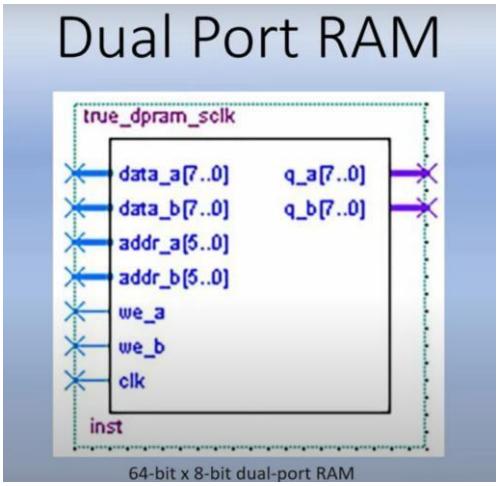

Figure 1: Dual- Port RAM

5.2) Introduction to Image Processing:

The digital era has ushered in an unprecedented surge in the generation and consumption of visual data, compelling us to develop sophisticated tools for its efficient manipulation. Image processing stands at the forefront of these endeavors, seeking not only to enhance visual content but also to devise robust methodologies for storage and retrieval.

This project embarks on a journey to seamlessly integrate software and hardware components, addressing the intricacies of image data processing.At the heart of our project lies the initial step of converting an input image into hexadecimal pixel values using a Python script.

This not only serves as a prelude to the subsequent stages but also underscores the flexibility and ease of use offered by high-level programming languages. The transition from the intricate details of pixel values in an image to a more abstract representation in hexadecimal form sets the stage for the subsequent hardware-centric processing.

5.3) The Verilog Paradigm and Xilinx ISE Integration:

Moving beyond the realms of software, the project introduces a Verilog-based dual-ported RAM system as the core hardware component. Leveraging the capabilities of Xilinx ISE 14.7, we delve into the intricacies of hardware description and synthesis. The dual-ported RAM system not only exemplifies efficient data storage but also introduces the concept of concurrent access, a vital aspect in scenarios where simultaneous reading and writing operations are paramount.

The Xilinx ISE environment emerges as a crucial facilitator in this integration. The Project Navigator serves as the control center, enabling the organization and management of project files. Platform Studio steps into the system-level design, providing an interface for seamlessly integrating hardware and software components. This integration, while challenging, is a testament to the versatility of Xilinx ISE in managing diverse design elements within a unified environment.

5.4) Vision and Objectives:

The vision of this project is rooted in the amalgamation of software and hardware paradigms to create an end-to-end image processing pipeline. Success is not merely defined by the accuracy of data preservation or the fidelity of image reconstruction but also by the harmony achieved in transitioning between Python and Verilog, between high-level abstraction and low-level implementation.

The objectives set forth encompass the seamless conversion of image data to hexadecimal pixel values, the efficient storage and retrieval through a Verilog-based dual-ported RAM system, and the ultimate reconstruction of the original image. The cross-disciplinary collaboration presented here exemplifies the synergy between Python and Verilog, guided by the comprehensive capabilities of Xilinx ISE 14.7. The project aspires to offer not just a solution but a methodology—a methodology that harmoniously integrates diverse elements to address the evolving challenges of digital image processing.

5.5) Simulation, Implementation, and Real-World Impact:

The project progression includes a pivotal phase of simulation using ISim, where the Verilog-based system is rigorously tested for functionality and robustness. Simulation becomes a critical step in validating the design before it advances to the physical implementation on an FPGA. Xilinx ISE's suite of implementation tools, including Translate, Map, Place & Route, and Generate Programming File, orchestrate the conversion of abstract designs into tangible configurations on the FPGA.

The real-world impact of this project extends beyond the confines of the digital environment. The seamless integration of Python and Verilog provides a powerful toolset for researchers, engineers, and developers, offering a blueprint for creating adaptable and efficient image processing systems. As industries increasingly seek solutions that bridge the gap between high-level abstraction and low-level implementation, the methodologies presented here become instrumental in shaping future endeavors.

6.LITERATURE SURVEY:

Haitong Li, Tony F. Wu, Subhasish Mitra, and H.-S. Philip Wong [1]:

this project focuses on addressing the challenges associated with high-speed image acquisition systems, specifically emphasizing the importance of acquisition speed in the overall system performance. The proposed solution involves the design and implementation of a high-speed, low-cost dual-port buffer memory using complex programmable logic (FPGA).

The image acquisition module is described, detailing its responsibility for driving the image sensor (OV7670) and storing the collected image information in an external SDRAM chip. The image acquisition process is explained, highlighting the use of control signals such as HREF and VSYNC, and the chosen RGB565 format for data storage.

The design of the dual-port RAM control module is then introduced, showcasing its role in efficiently managing data storage and retrieval. The implementation involves independent write and read clocks, enabling the system to optimize the use of kernel space and improve overall processing capabilities. The flow chart provides a visual representation of the image acquisition module based on the dual-port RAM.

Simulation and implementation of the dual-port RAM control module are discussed, emphasizing the verification of basic functions through Quartus II simulation. The schematic diagram illustrates the connections and operations within the control module. The project successfully demonstrates the feasibility of the proposed dual-port RAM solution, showcasing its effectiveness in enhancing system performance.

While the project presents a viable solution for high-speed image acquisition systems, there are potential drawbacks to consider. One notable limitation is the focus on a specific image sensor (OV7670) and its associated parameters, which may not be universally applicable to all image acquisition scenarios. Additionally, the project may not address potential challenges related to power consumption or scalability for larger datasets.

In conclusion, this project offers a practical solution for improving the acquisition speed and overall processing capability of image acquisition systems using a dual-port RAM approach. However, careful consideration should be given to the specific requirements of different image sensors and potential scalability issues for broader applications.

Yuan Qinghui, male, Jining Shandong, Binzhou Polytechnic[2]:

In conclusion, this project offers a comprehensive exploration of a memory-centric computing system based on resistive random access memory (RRAM). The design and modeling methodology focus on 3D RRAM technology, providing unique opportunities for energy-efficient native and local information processing. The project encompasses various aspects, including the development of a hierarchical RRAM SPICE model, inmemory operation schemes, and a case study involving hyperdimensional (HD) computing for language recognition.

The RRAM SPICE model is a key contribution, featuring three hierarchical levels of physics realism. This model enhances the accuracy of RRAM behavior representation, incorporating stochasticity for a more precise depiction of RRAM operations. The project successfully demonstrates three in-memory operation schemes for reconfigurable inmemory logic, showcasing the flexibility and efficiency of RRAM in 3D vertical structures.

As a case study, the project evaluates the use of 3D RRAMs for a language recognition system using the HD computing model. The integration of essential kernels for HD operations, namely multiplication, addition, and permutation (MAP), is achieved using fabricated 3D RRAM integrated with FinFET. The study reveals that RRAM-centric HD systems exhibit resilience to hard errors induced by RRAM endurance failures, making a compelling case for the utilization of various types of RRAM in memory-centric HD systems.

However, it's important to acknowledge potential drawbacks and limitations. Firstly, the project's focus on 3D RRAM technology may limit its applicability to other non-volatile

memory technologies. Additionally, the evaluation is specific to the language recognition case study, and the generalizability of the findings to other applications needs further exploration. Furthermore, the project does not extensively address potential challenges related to power consumption, scalability, or integration with existing semiconductor technologies.

In summary, this project lays the foundation for RRAM-centric computing systems, providing valuable insights into the design, modeling, and application of 3D RRAM technology. While the study shows promise for memory-centric HD systems, further research and development are required to address potential drawbacks and extend the applicability of the proposed approach to a broader range of computing applications.

7.IMPLEMENTATION OF PRELIMINARY APPROACH:

7.1) Introduction:

In the design and development of advanced instruments, particularly in real-time image acquisition and radar data acquisition, we will face a common problem: dealing with large amounts of data and high-speed transmission. As the sample data increases and information processing tasks become more demanding, there is a growing need for high-speed data transmission interfaces to prevent bottlenecks and delays between system modules.

To address this, the author proposes the use of dual-port RAM designed with static RAM. This solution aims to maintain high-speed data acquisition and processing while avoiding congestion and waiting issues. The dual-port RAM is described as offering high speed, large capacity, and low cost. The paper suggests that implementing this solution improves the overall data acquisition and processing capabilities of the system.

The proposed approach is validated through simulation, demonstrating its feasibility.

7.2) Image Conversion and Memory Storage:

The image acquisition module is responsible for driving the image and storing the collected image information in a specific format. The workflow begins with a Python script converting an input image into hexadecimal pixel values, which are stored in a text file. The text file is then processed by a Verilog-based dual-ported RAM system designed using Xilinx ISE 14.7. This system reads the hexadecimal values and produces an output file containing the read pixel values.

In the initial phase of our project, we employ Python as the tool of choice for image preprocessing. A dedicated Python script takes an input image, systematically traverses its pixels, and converts their values into a hexadecimal format. This conversion not only condenses the representation of pixel data but also prepares the information for seamless integration with subsequent hardware-based operations.

The resulting hexadecimal pixel values are meticulously stored in a text file, establishing a bridge between the high-level abstraction of Python and the low-level processing to be conducted in Verilog.

Python code utilizing Pillow to read an image and convert its pixel data into hexadecimal values. These values are stored in a text file `hex_pixel_values.txt`.

7.2.1. Dual -Port RAM Image Storage

Verilog code implementing a dual-port RAM to store the hexadecimal pixel values read from the file. The data is written into memory locations and can be accessed through read operations based on the specified address.

Image memory storage involves the conversion and organization of image data within a storage medium, often represented in digital systems using various data structures. In the context of the implemented approach, the image memory storage process begins with the Python script utilizing the Pillow library to convert an image into a sequence of hexadecimal values, representing the RGB (Red, Green, Blue) components of each pixel. These values are then stored in a text file `hex_pixel_values.txt`.

In the Verilog code, a dual-port RAM structure is employed to accommodate these hexadecimal pixel values. The Verilog code initializes a memory array and loads the pixel data into this memory by reading the content of `hex_pixel_values.txt` using the `$readmemh` task.

This operation effectively organizes the image data into memory locations, enabling subsequent access and retrieval. Each memory location corresponds to a pixel in the image, allowing for the efficient storage and manipulation of pixel information. The dual-port RAM's capability to perform simultaneous read and write operations enables seamless handling of the image data, facilitating both the storage and retrieval processes efficiently.

This consolidated approach ensures a systematic and organized storage mechanism for image data within the digital system's memory, enabling accessibility and manipulation for various image processing applications or further computational tasks.

7.3. Operation of Dual-Port RAM based Image Storage

7.3.1. Dual-Port RAM Write Process

The write process begins with the initialization of memory space and an output file in the Verilog code. This process involves the utilization of the $readmemh system task to read the pre-converted hexadecimal pixel values from the specified file, such as hex_pixel_values.txt.

These values are then sequentially written into the dual-port RAM memory locations. During this operation, the Verilog code writes the read data into the memory array, allocating memory space for storing the image's pixel information. Additionally, it simultaneously records the data into an output file, creating a record of the data written into the memory for verification or future reference purposes.

7.3.2. Dual-Port RAM Read Process

Upon initialization and set-up, the read operation in the Verilog code is triggered by the clock signal or reset signal's positive edge. The code listens for a read enable signal and an address input to perform a read operation from the dual-port RAM. When the read enable signal is asserted and an address is provided, the corresponding pixel data stored at that address in the memory is retrieved.

Subsequently, this read data is displayed in hexadecimal format using the $display function, providing visibility into the retrieved pixel value. Moreover, the Verilog code also writes this read data into an output file (out_pixel_values.txt), creating a log of the read data for further analysis or verification purposes.

In essence, the write operation initializes and populates the dual-port RAM with image data in hexadecimal format, while the read operation facilitates the retrieval of this stored data based on specified memory addresses, enabling subsequent analysis or utilization of the image pixel information. This coherent write and read process demonstrates the efficient storage and retrieval mechanisms inherent in the Dual-Port RAM-based image storage approach.

A image processing with Python script and a Verilog design to store and retrieve image data using a dual-port RAM.

The Python script uses the Pillow library to load an image file and convert its pixel data into hexadecimal values. The image is opened and processed to extract the RGB values of each pixel, which are then converted into a hexadecimal format. This conversion allows for a compact representation of the pixel data. The resulting hexadecimal values are stored in a list and subsequently written into a text file named `hex_pixel_values.txt`. This file serves as a repository for the converted image data.

The Verilog code is designed to manage the storage and retrieval of the image data using a dual-port RAM structure. Initially, the code initializes a memory array, which represents the storage space for the image pixel values. Additionally, it opens an output file (`out_pixel_values.txt`) for recording read operations.

The code utilizes the `$readmemh` system task to read the previously generated `hex_pixel_values.txt` file. This task populates the dual-port RAM with the hexadecimal pixel values obtained from the Python script. The read operation reads data from the dualport RAM based on the provided address and the read enable signal.

To ensure the functionality of the Verilog design, a test bench is employed. This test bench initializes necessary signals, generates clock signals, and orchestrates read operations on the implemented dual-port RAM. It simulates the reading of pixel values from memory locations.

During simulation, the test bench iterates through multiple memory addresses, effectively emulating the process of reading all pixel values stored in the dual-port RAM. This simulated process checks the functionality of the design by verifying the retrieval of pixel data at specific memory locations.

The overall project workflow entails the Python script initially converting the image data into hexadecimal values and storing them in a text file. Subsequently, the Verilog design reads these values from the file and populates a dual-port RAM with the image data. The

test bench then verifies the read functionality of the implemented design by simulating the retrieval of pixel values stored in the memory locations.

This comprehensive integration of Python image processing with Verilog-based dual-port RAM storage provides a systematic approach for converting, storing, and verifying the manipulation of image data through a hardware-based memory system.

8.SIMULATION RESULTS AND ANALYSIS:

The Verilog code involves a data path that manages reading and writing operations for a dual-port RAM module `ReadHexValues`. The input signals `clk`, `reset`, `read_en`, `write_en`, and `address' control the operations of the dual-port RAM. Upon receiving a clock signal (`clk`), the module reads data from the dual-port RAM at the specified address when the `read_en` signal is active. The `reset` signal initializes the module's state. When reading is enabled (`read_en`), the module accesses the `mem` array based on the provided address, displaying the read data on the console and writing it to an output file. This Verilog module operates based on clock edges and control signals, handling data retrieval from the dual-port RAM and showcasing the read data, simulating the read process within the specified addresses.

8.1) RTL:

The RTL Schematic of Data path shows that address(23:0), clk, read_en, reset, write_en are the inputs and the outputs are data_out (23:0).

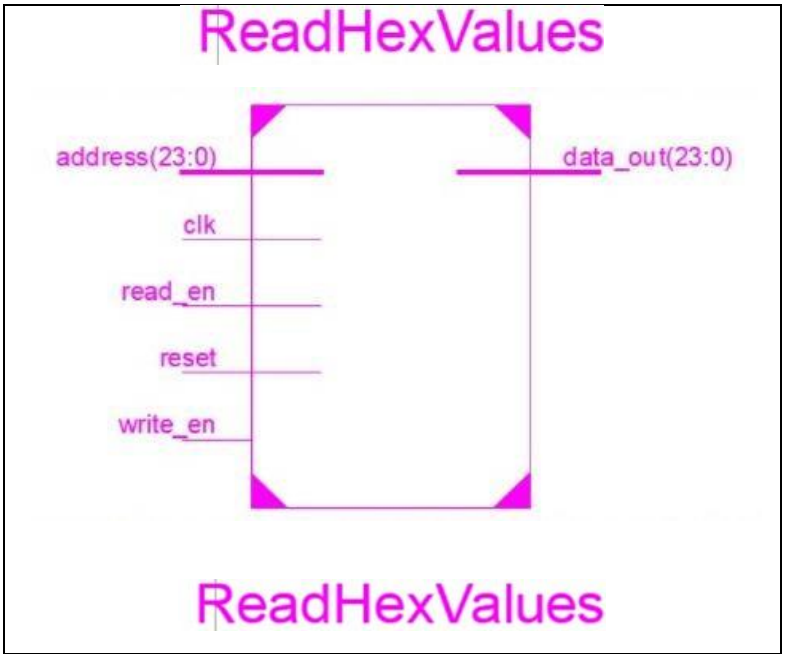

Figure 8.1: RTL Schematic

8.2) Simulation results:

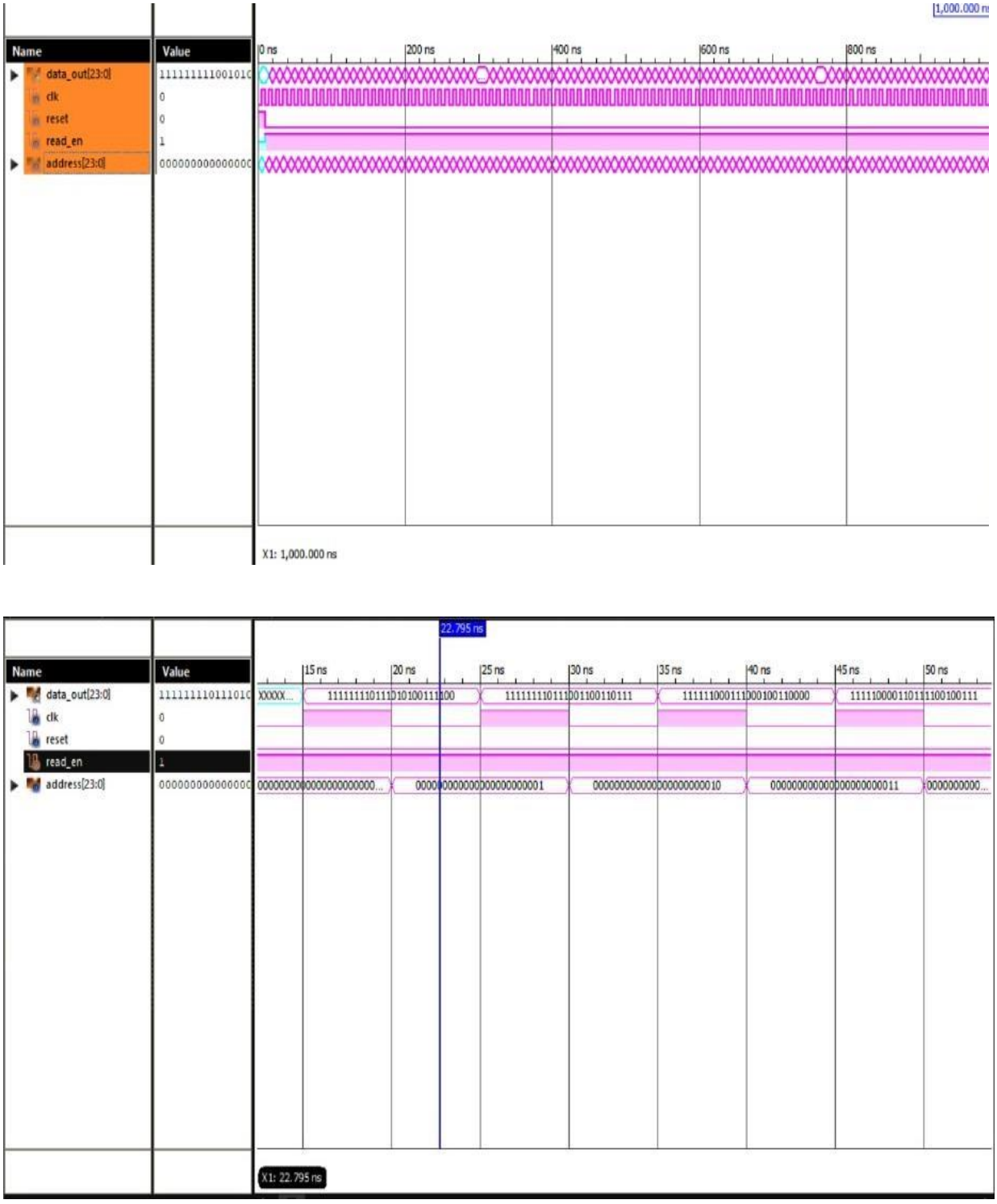

Figure 2: Waveforms

8.3) Analysis:

For image data processing the integration of Python and Verilog within the Xilinx ISE framework has yielded a multifaceted analysis. The project's success hinges on the seamless collaboration between software and hardware components, underscoring the efficiency of data storage, retrieval, and image reconstruction.

The project establishes a harmonious interplay between Python and Verilog, showcasing the adaptability of high-level and hardware description languages. The initial Python script efficiently converts image data into hexadecimal pixel values, providing a succinct interface between the original image and subsequent hardware-centric processing. The dual-ported RAM system, implemented in Verilog, navigates the intricacies of concurrent read and write operations, demonstrating the flexibility and power of hardware description languages.

Xilinx ISE 14.7 emerges as a pivotal enabler throughout the project lifecycle. The Project Navigator serves as a centralized hub for project management, facilitating the organization of files and design elements. Platform Studio extends this capability, enabling seamless integration of hardware and software components

The project places a strong emphasis on simulation using ISim, an integral aspect of digital design. Through rigorous simulation, potential issues in the Verilog-based dual-ported RAM system are identified and resolved before physical implementation. This iterative process of simulation and validation ensures the reliability and correctness of the design, reflecting industry best practices in digital design workflows.

9.CONCLUSION:

The project's culmination brings forth a comprehensive image processing pipeline, seamlessly transitioning from a high-level software environment to a low-level hardware implementation. The objectives, rooted in the conversion of image data to hexadecimal pixel values, efficient storage and retrieval through a Verilog-based dual-ported RAM system, and the ultimate reconstruction of the original image, have been met.

The integration of Python and Verilog exemplifies the evolving landscape of mixedlanguage design, providing a versatile methodology for tackling complex challenges in digital image processing.

This project not only serves as a demonstrative solution but lays the groundwork for future endeavors in cross-disciplinary applications. The seamless integration of software and hardware, facilitated by Xilinx ISE tools, opens avenues for researchers, engineers, and developers to explore adaptable and efficient solutions for diverse image processing needs.

In conclusion, the project stands as a testament to the synergies achievable through the thoughtful integration of Python, Verilog, and the powerful Xilinx ISE environment. The methodologies presented here contribute to the evolving landscape of digital design, offering insights and solutions that bridge the gap between high-level abstraction and lowlevel hardware implementation in the context of image processing.

10.REFERENCES:

1. Haitong Li, Student Member, IEEE, Tony F. Wu, Subhasish Mitra, Fellow, IEEE, and H.-S. Philip Wong, Fellow, IEEE **"**Resistive RAM-Centric Computing: Design and Modeling Methodology**"**.

2. Yuan Qinghui, male, Jining Shandong, Binzhou Polytechnic **"**Study on Dual-port RAMbased Image Capture and Storage**"**, mainly engaged in electrical automation engineering research and management. Conference Paper · January 2016
DOI: 10.2991/iccse-16.2016.30

11. APPENDIX:

11.1) Code:

11.1.1) Converting image into hexadecimal pixel values using python:

```
pip install Pillow from PIL import Image # Open the image file input_image_path =
'C:\\Users\\USER\\OneDrive\\pyim123.jpg'  # Replace with your image path image =
Image.open(input_image_path)
# Convert image data to a list of hexadecimal pixel values pixel_data =
list(image.getdata()) hex_pixel_values =
[f'{pixel[0]:02X}{pixel[1]:02X}{pixel[2]:02X}' for pixel in pixel_data]
# Save the hexadecimal pixel values to a file output_file_path =
'hex_pixel_values.txt'  # File to save the hexadecimal pixel values with
open(output_file_path, 'w') as file:     file.write('\n'.join(hex_pixel_values))
print(f"Hexadecimal pixel values saved to '{output_file_path}'")
```

Figure 3: Input Image

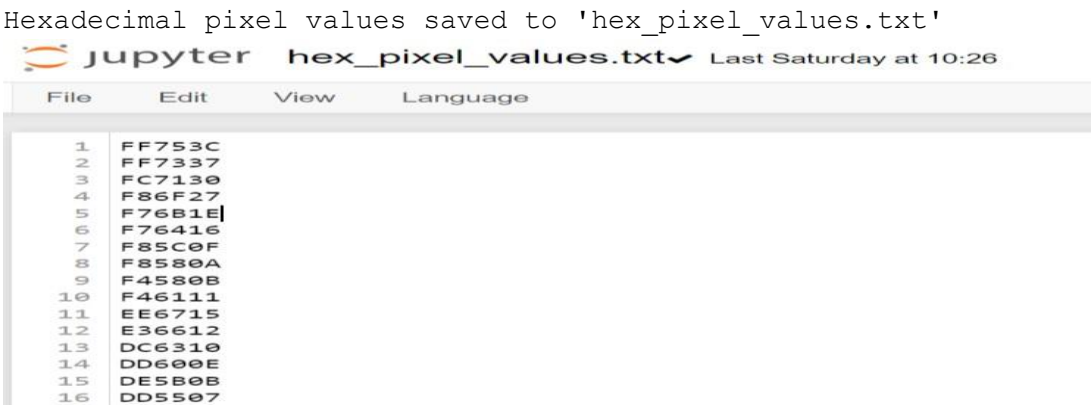

Figure 4: Hexadecimal Pixel Values

11.1.2) Verilog code: module ReadHexValues (

input wire clk, input wire reset, input wire read_en, input wire write_en, input wire [23:0] address, output reg [23:0] data_out);

reg [23:0] mem [0:4096- 1]; // Assuming Total_Pixels memory locations reg [23:0] read_data; integer file_handler; initial begin

$readmemh("hex_pixel_values.txt", mem); // Read data from the hexadecimal file // Open the output file for writing file_handler = $fopen("out_pixel_values.txt", "w"); end always @(posedge clk or posedge reset) begin if (reset) begin

// Reset values if needed

// ...

end else begin if (read_en) begin

// Read data from the dual-port RAM based on the address read_data = mem[address]; // Output the read data to the console

```
            $display("Read data: %h", read_data);
            // Write the read data to the output file
            $display(file_handler, "%h", read_data);
// Assign the read data to the output
data_out <= read_data;          end       end
end    initial begin
      $fclose(file_handler);
end
endmodule
```

11.1.3) Test bench:

```
`timescale 1ns / 1ps module
ReadHexValues_tb;    reg
clk;    reg reset;    reg
read_en;
   reg [23:0] address;    wire [23:0] data_out; // Changed from
reg to wire for output
   // Instantiate the module under test
   ReadHexValues dut (
      .clk(clk),
      .reset(reset),
      .read_en(read_en),
      .address(address),
      .data_out(data_out)
   );
   // Clock generation
always #5 clk = ~clk;     //
Testbench initial block
initial begin
      // Provide a clock signal (adjust frequency if needed)
clk = 0;
```

```
        // Reset signal initialization
reset = 1;          #10 reset = 0;
        // Read operation initialization
read_en = 1;          address = 0;

        // Loop through reading all pixel values (adjust this based on the number of pixels)
repeat (4095) begin
            // Simulating data output from the module
            #10; // Simulating delay for the module to respond
            // Displaying the pixel value read at the given address
            $display("Reading pixel value at address %d: %h", address, data_out);
            // Incrementing the address for the next read
address = address + 1;
        end         // Finish
simulation          $finish;
end
endmodule
```