**LOYOLA-ICAM COLLEGE**

**OF ENGINEERING AND TECHNOLOGY (LICET)**

Loyola Campus, Nungambakkam, Chennai -600034

**<u>INTERNSHIP PROJECT REPORT</u>**

# VLSI BASED SYNTHESIS OF IMAGE PROCESSING ALGORITHMS

SUBMITTED BY

- **ANNIE PERSIA S - 311121106007**
- **JANANI V - 311121106024**
- **LAETITIA MATHEW - 311121106038**
- **LINNET A S - 311121106041**
- **SAHAYA SHEENA A - 311121106051**

III Year - Electronics & Communication Engineering

**"Online Summer VLSI Internship Programme"**

AT TESSOLVE SEMICONDUCTOR PVT. LTD

In partial fulfillment of the requirements for the degree of BACHELOR OF ENGINEERING (ELECTRONICS AND COMMUNICATION)

Under the reputable guidance of
**MR. NEPOLEAN M.**

Training period: 24/01/2024 to 20/06/2024

## ACKNOWLEDGEMENT

We are grateful to our Almighty God for His blessings to complete our project successfully. We would also like to thank our parents for their continuous support and encouragement in this regard.

We express our sincere gratitude to **Rev. Dr. S. Sebastian S. J , Director, LICET**. We extend our gratitude to **Dr. L. Antony Michael Raj, Principal**, for his support of our project. We wish to convey our sincere thanks to **Rev. Dr. Justine S. J, Dean of Students,** for his support and inspiration towards this project.

We express our sincere appreciation to **Ms. L.J. Jennifer Suriya , Assistant Professor and Head**, Department of Electronics and Communication Engineering, for her constant care, motivation, and reassurance in being our source of inspiration throughout this project and during the course of this semester.

We are extremely obliged to our **Guide, Mr Nepolean M**, for his able guidance and whole-hearted cooperation which enabled us to finish this project successfully.

We are deeply indebted to our **Mentor , Dr. A. Anitha Juliette , Dean of Research** for her unwavering support and invaluable guidance throughout this project. Her insights and encouragement have been instrumental in achieving these results.

We convey our special thanks to our **Class Advisor, Ms. Jerlin A., Assistant Professor**, for providing us with the required resources and timely support for completing this project.

## ENDORSEMENT

We, hereby affirm that the report titled "**VLSI based synthesis of image processing algorithms**" is our original work. We confirm that we have written this report ourselves and that it has not been previously submitted for academic credit at any other institution.

## BRIEF EXECUTIVE SUMMARY OF INTERNSHIP

Tessolve Semiconductors, founded in 2004 and headquartered in Bangalore, India, is a leading provider of end-to-end semiconductor engineering solutions. With a strong presence in Chennai and other global locations,Tessolve has established itself as a significant player in the semiconductor industry. The company is renowned for its comprehensive range of services, which include IC design, test engineering, PCB design, failure analysis,product engineering, and embedded systems design. Tessolve's mission is to deliver innovative and cost-effective solutions that enhance the performance and reliability of semiconductor products.

During the internship, we gained a solid understanding of both Verilog and SystemVerilog, starting with the basics of Verilog, a hardware description language (HDL) used for modeling electronic systems. We learned about the history and purpose of Verilog, focusing on its syntax and structure, which includes modules, ports, and data types. We explored how to define a module, the fundamental building block in Verilog, and how to declare and connect input, output, and bidirectional ports.

We delved into various data types such as \`wire\`, \`reg\`, \`integer\`, and \`logic\`, and understood the application of different operators, including arithmetic, logical, relational, and bitwise operators, to manipulate these data types. The concepts of continuous assignments using the \`assign\` statement for combinational logic, and procedural blocks using \`initial\` and \`always\` blocks for sequential logic, were thoroughly covered.

Furthermore, we advanced to SystemVerilog, which extends Verilog with additional features for system-level design and verification. This included learning about enhanced data types, interfaces, and the powerful verification constructs provided by SystemVerilog, which significantly aid in the design and verification of complex digital systems. Overall, the internship provided a comprehensive introduction to both Verilog and SystemVerilog, equipping us with the necessary skills to model, design, and verify electronic systems effectively.

Basically our project is VLSI based synthesis of image processing algorithm ,So our project is based on image processing and VLSI design. The project flow is converting images with salt and pepper noise into hex files using MATLAB, processing these files with Verilog-based algorithms(Quartus), and converting them back into image for visualization with reduced salt and pepper noise.

Key findings include a deep understanding of image data structures, successful implementation of an arithmetic mean filter in Verilog to reduce noise, and the importance of using Quartus Prime software for simulation and verification. The Project highlighted the differences between theoretical knowledge and practical application, such as handling boundary conditions and optimizing for hardware constraints.By this project we improve and enhance our skills in MATLAB, Verilog, and Quartus Prime, and help in demonstrating the feasibility of VLSI-based image processing.

The report highlights the successful completion of our project and collaborative tasks that effectively prepared us for real-world engineering challenges. Overall, the training provided valuable insights into the practical applications of VLSI design in various fields, including medical imaging and digital photography.

## COMPANY OVERVIEW

## TESSOLVE SEMICONDUCTOR PVT. LTD.

Tessolve Semiconductor Pvt. Ltd., founded in 2004 by V. Raja Manickam, is a prominent player in the semiconductor and electronics sector. Headquartered in Bengaluru, Karnataka, India, Tessolve has grown significantly, boasting a workforce of around 3000 employees and generating an estimated annual turnover exceeding $100 million. The company operates as a private entity, focusing on providing comprehensive semiconductor solutions.

The company's core strengths lie in its extensive expertise in semiconductor testing, packaging, and system design. Tessolve's services encompass end-to-end semiconductor solutions, including design, testing, failure analysis, and yield analysis. These capabilities enable Tessolve to support clients across various stages of semiconductor development, ensuring high-quality and reliable outcomes. Additionally, Tessolve's focus on innovation and continuous improvement has solidified its reputation as a leader in the industry.

Tessolve's product offerings are diverse and cater to a broad spectrum of needs within the semiconductor industry. Their solutions include advanced testing services, engineering solutions for integrated circuits (ICs), system-level design and analysis, and failure analysis services. By leveraging their strengths and broad expertise, Tessolve provides critical support to semiconductor manufacturers and ensures the delivery of high-performance, reliable products. The company's commitment to quality, customer satisfaction, and technological advancement positions it as a valuable partner in the semiconductor ecosystem.

## TABLE OF CONTENTS

# 6. INTERNSHIP PROJECT

## 6.1 Context of the Project

Our internship spanned six months, during which we gained comprehensive training and hands-on experience in concepts, Verilog, and SystemVerilog. The first three months were dedicated to in-depth learning of these topics, a solid foundation for practical application. The subsequent three months involved working on a collaborative project, providing an opportunity to apply the theoretical knowledge acquired. Our team, consisting of five members, selected a project titled "VLSI Based Synthesis of Image Processing Algorithm," focusing on the implementation of an arithmetic mean filter.

## 6.2 Tasks Assigned

The project aimed to address the challenge of removing salt and pepper noise, commonly found in medical imaging and remote sensing applications. Our tasks included selecting an appropriate image affected by salt and pepper noise, converting this image into a hexadecimal (hex) file format suitable for processing in Verilog, and implementing the arithmetic mean filter using Verilog, SystemVerilog, and testbench code. We were also responsible for synthesizing the processed output back into a hex file and visualizing the filtered image using MATLAB to verify the effectiveness of our algorithm.

## 6.3 Project Schedule

The project was structured over a three-month period, divided into distinct phases:

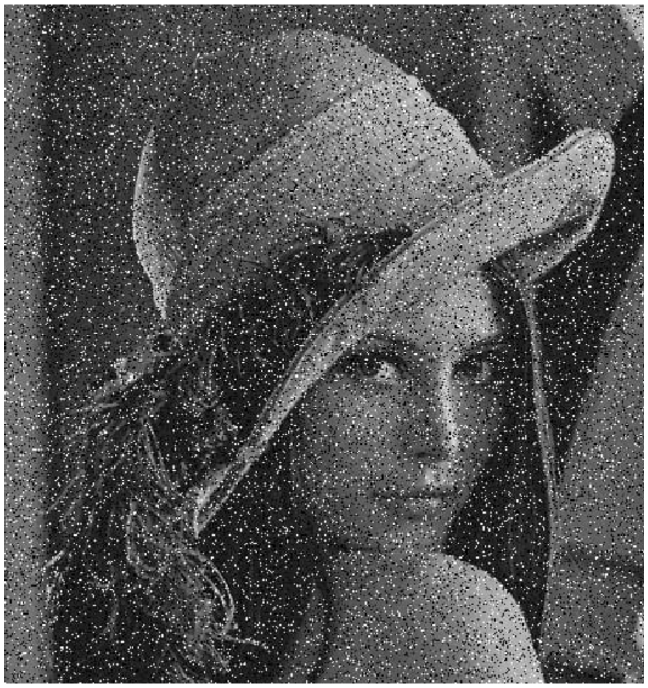
- Phase 1 (Weeks 1-4):Selection of the project topic, literature review, and image selection with salt and pepper noise.
- Phase 2 (Weeks 5-8): Conversion of the noisy image into a hex file and development of the arithmetic mean filter algorithm in Verilog.
- Phase 3 (Weeks 9-12): Implementation of the algorithm in SystemVerilog and development of the testbench code.
- Phase 4 (Weeks 13-14):Synthesis of the output into a hex file and visualization of the filtered image using MATLAB.
- Phase 5 (Weeks 15-16): Final testing, validation, and project documentation.

## 6.4 Proposed Solution

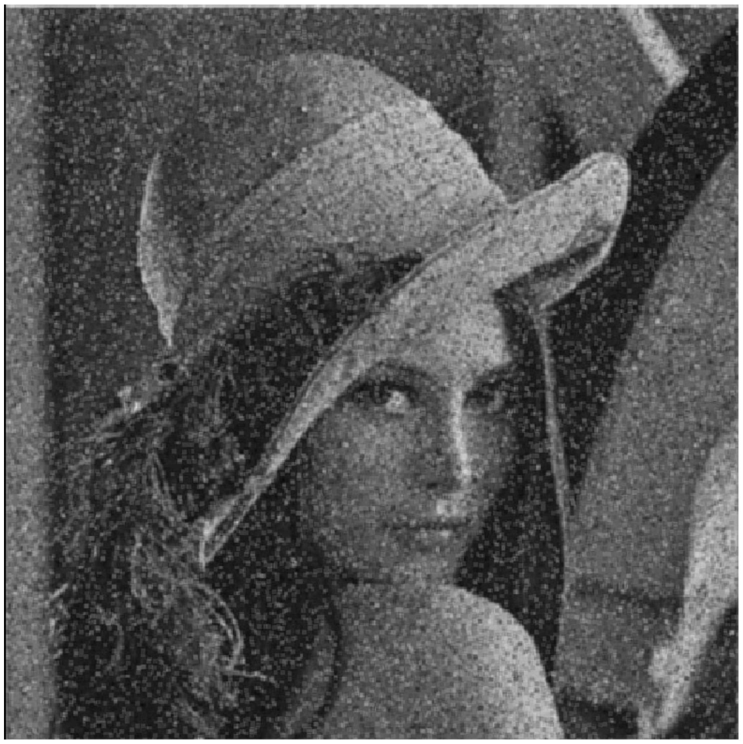
To address the project's objective, we proposed converting the noisy image into a hex file format that could be processed using VLSI techniques. The arithmetic mean filter algorithm was implemented in Verilog using Quartus Prime Lite, chosen for its suitability in FPGA design and synthesis. This Verilog implementation aimed effectively reduce salt and pepper noise. To ensure the accuracy and functionality of our design, we utilized SystemVerilog for enhanced verification capabilities and developed a comprehensive testbench code to simulate various test scenarios. The final output was synthesized back into a hex file format and visualized using MATLAB to confirm the reduction of salt and pepper noise. This multi-step approach ensured a systematic and efficient execution of the project, resulting in a significantly improved image quality.

Below,the input image with salt and pepper noise and the output image after applying the arithmetic mean filter:

- Input Image:



- Output Image:



## 6.5 Code Implementation

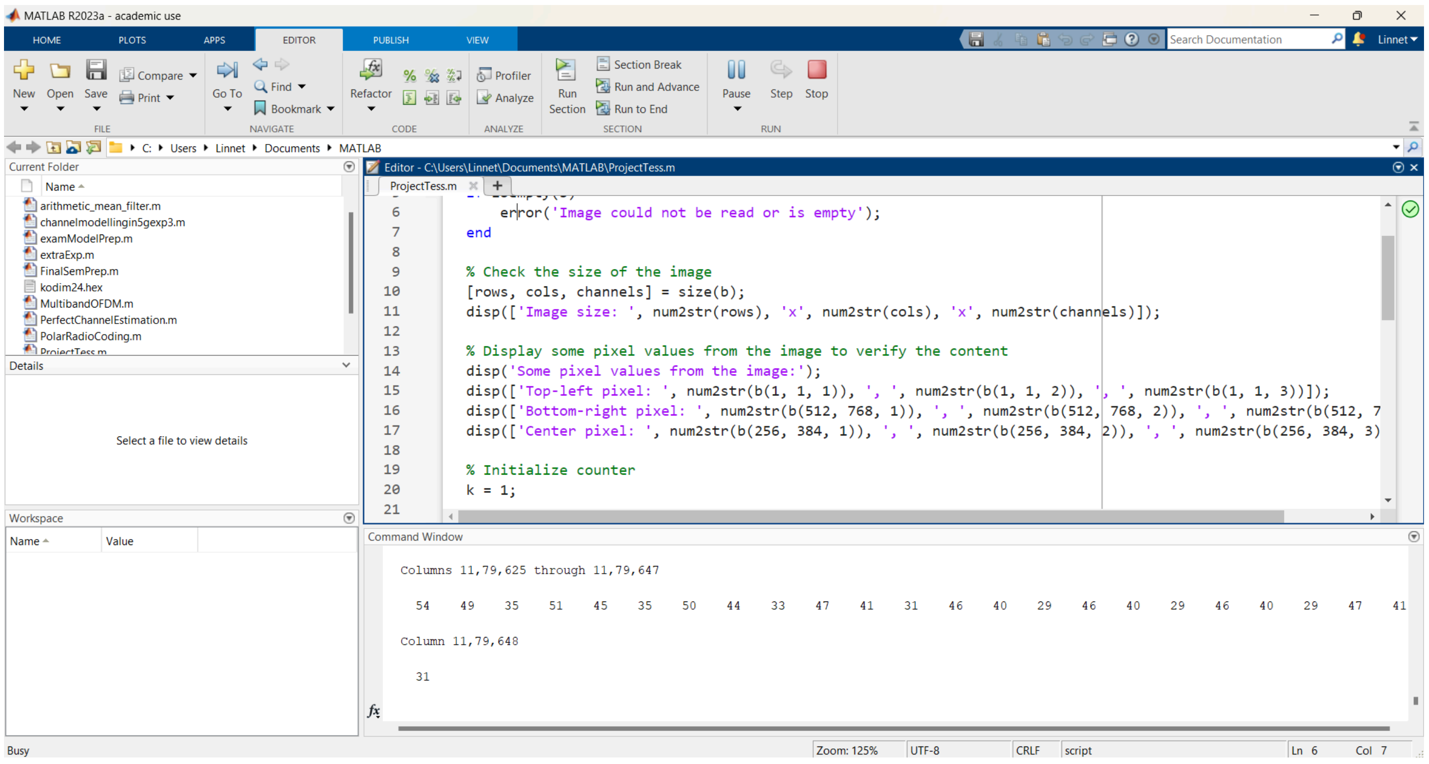
### 6.5.1 Matlab Code (Conversion of image to hex file) and its Output

- **Code**

```
% Read the 24-bit BMP image in RGB888 format
imagePath = "D:\Image Processing using Verilog\SampleImage.bmp";
b = imread(imagePath);

% Verify that the image has been read correctly
if isempty(b)
   error('Image could not be read or is empty');
end

% Check the size of the image
[rows, cols, channels] = size(b);
disp(['Image size: ', num2str(rows), 'x', num2str(cols), 'x', num2str(channels)]);

% Display some pixel values from the image to verify the content
disp('Some pixel values from the image:');
disp(['Top-left pixel: ', num2str(b(1, 1, 1)), ', ', num2str(b(1, 1, 2)), ', ', num2str(b(1, 1, 3))]);
disp(['Bottom-right pixel: ', num2str(b(rows, cols, 1)), ', ', num2str(b(rows, cols, 2)), ', ',
num2str(b(rows, cols, 3))]);
disp(['Center pixel: ', num2str(b(floor(rows/2), floor(cols/2), 1)), ', ', num2str(b(floor(rows/2),
floor(cols/2), 2)), ', ', num2str(b(floor(rows/2), floor(cols/2), 3))]);

% Initialize counter
k = 1;

% Preallocate array 'a' for performance
a = zeros(1, rows * cols * channels, 'uint8');

% Process the entire image
for i = rows:-1:1  % Reverse row order
   for j = 1:cols  % Normal column order
      a(k) = b(i, j, 1);   % Red channel
      a(k+1) = b(i, j, 2); % Green channel
      a(k+2) = b(i, j, 3); % Blue channel
      k = k + 3;
   end
end

% Verify some elements of the array 'a' before writing to the file
disp('First few elements of array a after full image processing:');
disp(a(1:10));  % Display only the first few elements for brevity
```

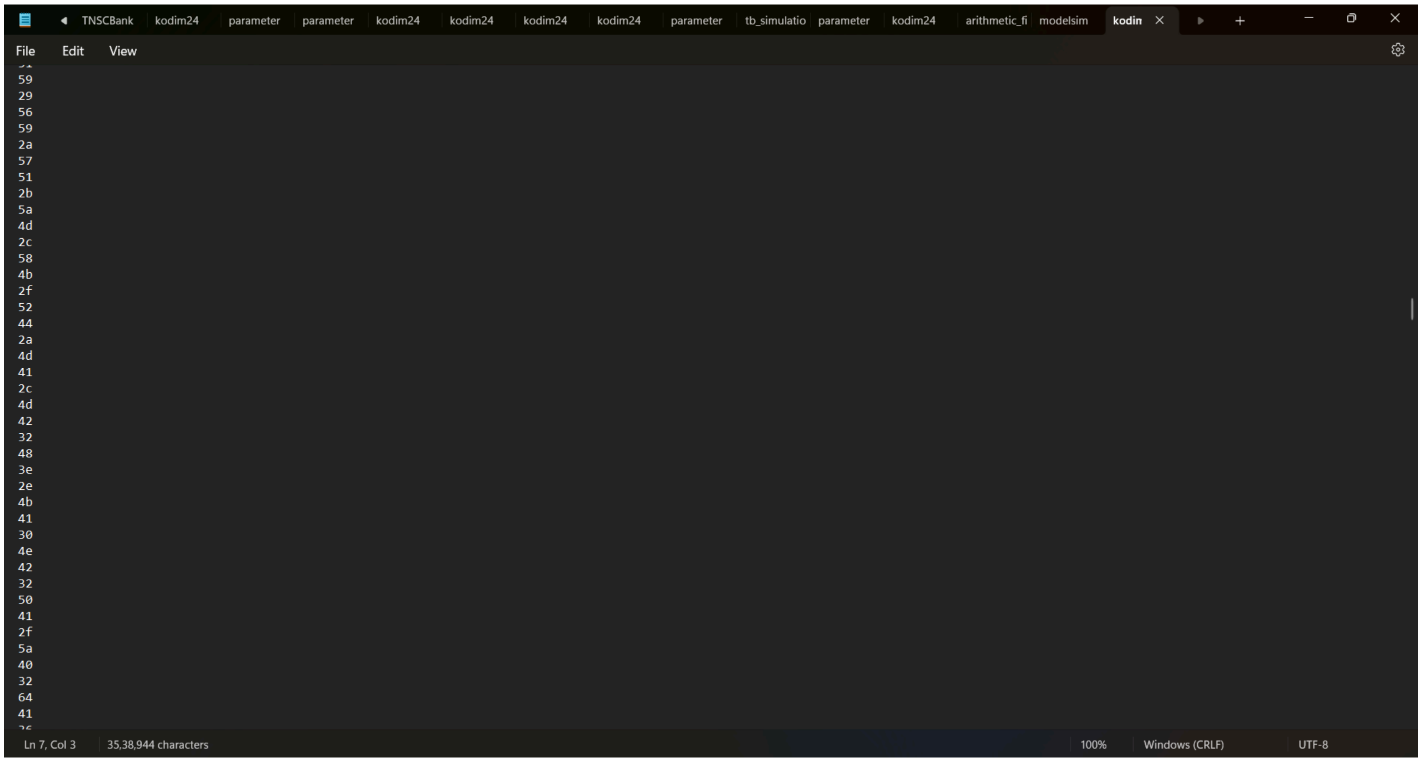
```
% Open a text file for writing in text mode
hexFilePath = 'C:\Users\Linnet\Documents\MATLAB\kodim24.hex';
fid = fopen(hexFilePath, 'wt');

% Check if the file opened successfully
if fid == -1
    error('Could not open file for writing: %s', hexFilePath);
end

% Write the data in 'a' to the file in hexadecimal format
for i = 1:length(a)
    fprintf(fid, '%02x\n', a(i));
end

% Display a message to confirm that the file write is done
disp('Text file write done');
disp(' ');

% Close the file
fclose(fid);
```

- **Output**



```
    error('Image could not be read or is empty');
end

% Check the size of the image
[rows, cols, channels] = size(b);
disp(['Image size: ', num2str(rows), 'x', num2str(cols), 'x', num2str(channels)]);

% Display some pixel values from the image to verify the content
disp('Some pixel values from the image:');
disp(['Top-left pixel: ', num2str(b(1, 1, 1)), ', ', num2str(b(1, 1, 2)), ', ', num2str(b(1, 1, 3))]);
disp(['Bottom-right pixel: ', num2str(b(512, 768, 1)), ', ', num2str(b(512, 768, 2)), ', ', num2str(b(512, 7
disp(['Center pixel: ', num2str(b(256, 384, 1)), ', ', num2str(b(256, 384, 2)), ', ', num2str(b(256, 384, 3)

% Initialize counter
k = 1;
```

```
  Columns 11,79,625 through 11,79,647

    54    49    35    51    45    35    50    44    33    47    41    31    46    40    29    46    40    29    46    40    29    47    41

  Column 11,79,648

    31
```



```
59
29
56
59
2a
57
51
2b
5a
4d
2c
58
4b
2f
52
44
2a
4d
41
2c
4d
42
32
48
3e
2e
4b
41
30
4e
42
32
50
41
2f
5a
40
32
64
41
```

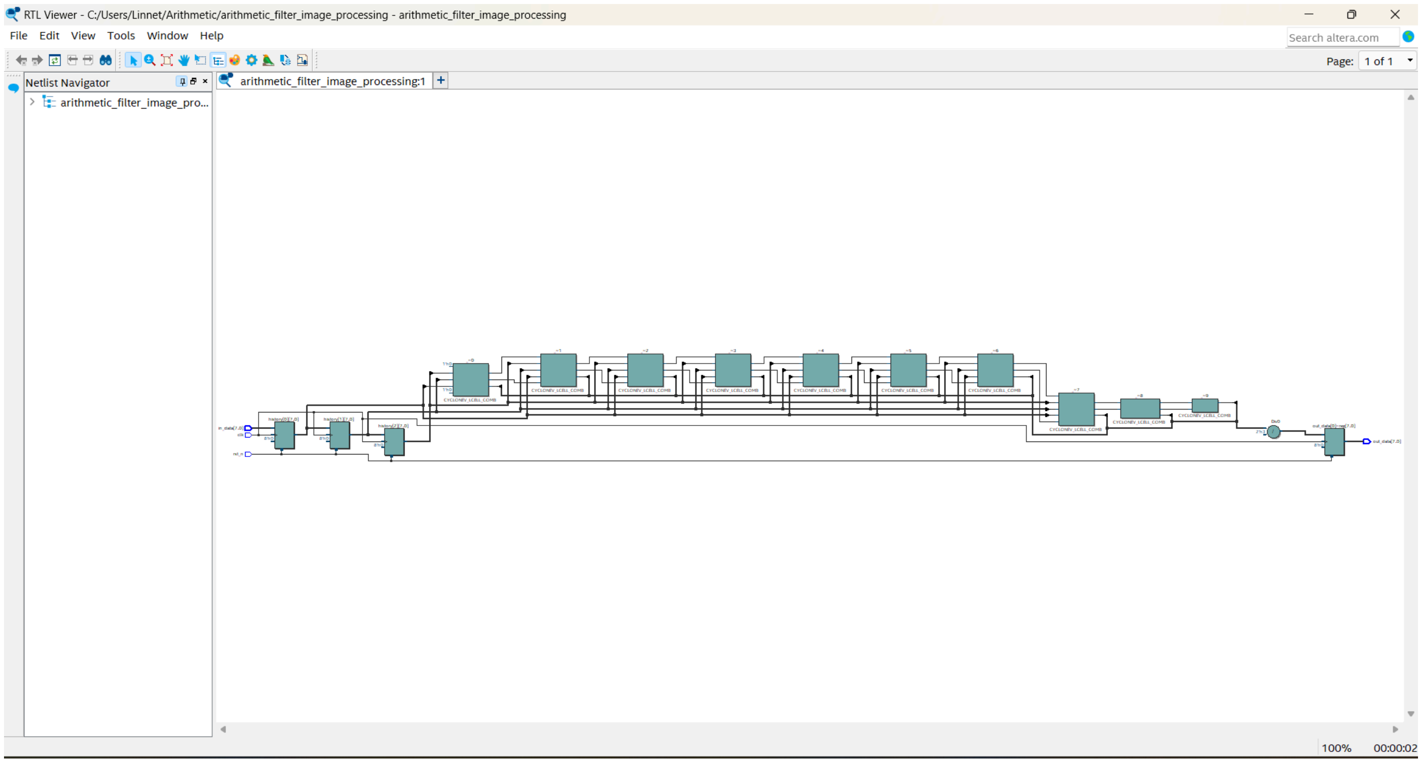
## 6.5.2 Verilog Code and RTL Schematic Visualization

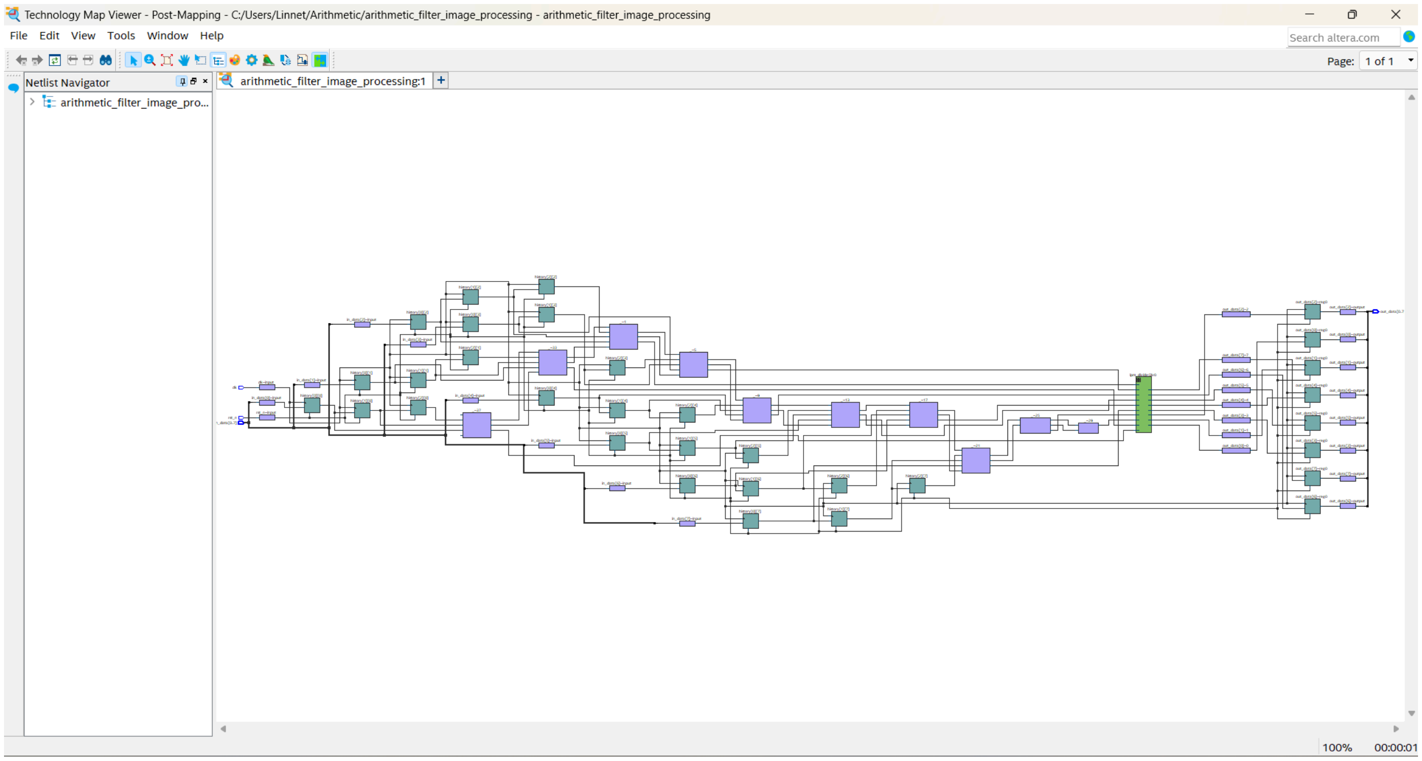
- **Code**

```
module arithmetic_filter_image_processing (
    input clk,           // Clock
    input rst_n,         // Reset (active low)
    input [7:0] in_data, // Input pixel data
    output reg [7:0] out_data // Output filtered pixel data
);

reg [7:0] history [2:0]; // History buffer to store previous pixel values

always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
  history[0] <= 8'h00;
  history[1] <= 8'h00;
  history[2] <= 8'h00;
  out_data <= 8'h00;
end else begin
  // Shift data through history buffer
  history[0] <= in_data;
  history[1] <= history[0];
  history[2] <= history[1];
```

```
        // Apply arithmetic filter (e.g., averaging filter)
        out_data <= (history[0] + history[1] + history[2]) / 3; // Simple averaging filter
      end
    end
    endmodule
```

- **Output**


RTL Viewer - C:/Users/Linnet/Arithmetic/arithmetic_filter_image_processing - arithmetic_filter_image_processing
File Edit View Tools Window Help
Search altera.com
Page: 1 of 1
Netlist Navigator
arithmetic_filter_image_pro...
arithmetic_filter_image_processing:1
100%
00:00:02


Technology Map Viewer - Post-Mapping - C:/Users/Linnet/arithmetic_filter_image_processing - arithmetic_filter_image_processing
File Edit View Tools Window Help
Search altera.com
Page: 1 of 1
Netlist Navigator
arithmetic_filter_image_pro...
arithmetic_filter_image_processing:1
100%
00:00:01

### 6.5.3 System Verilog Code

- **Code**

```
module arithmetic_filter_image_processing (
    input logic clk,            // Clock
    input logic rst_n,          // Reset (active low)
    input logic [7:0] in_data,  // Input pixel data
    output logic [7:0] out_data // Output filtered pixel data
);

logic [7:0] history [2:0]; // History buffer to store previous pixel values

always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        history[0] <= 8'h00;
        history[1] <= 8'h00;
        history[2] <= 8'h00;
        out_data <= 8'h00;
    end
    else begin
        // Shift data through history buffer
        history[0] <= in_data;
        history[1] <= history[0];
        history[2] <= history[1];

        // Apply arithmetic filter (e.g., averaging filter)
        out_data <= (history[0] + history[1] + history[2]) / 3; // Simple averaging filter
    end
end

endmodule
```

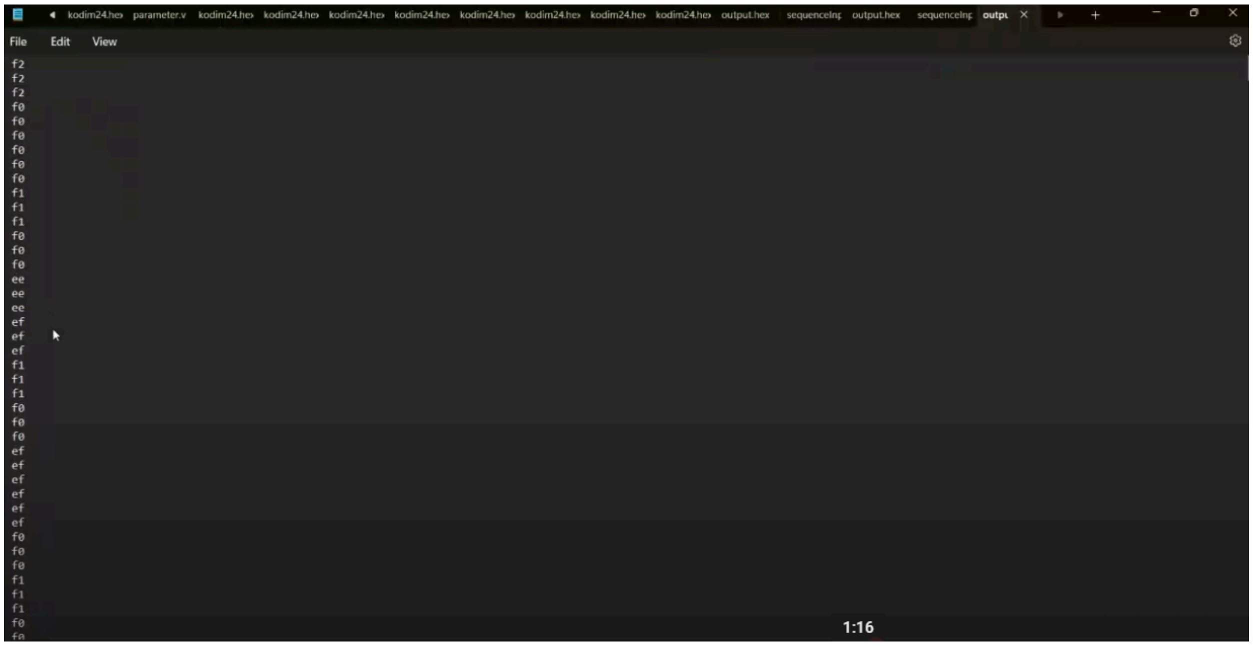
### 6.5.4 Testbench Code and Output of Verilog

- **Code**

```
module arithmetic_filter_image_processing_tb;

// Parameters
 parameter CLK_PERIOD = 10; // Clock period in ns

 // Signals
reg clk = 0;          // Clock signal
reg rst_n = 0;        // Reset signal (active low)
 reg [7:0] in_data;    // Input pixel data
 wire [7:0] out_data;  // Output filtered pixel data
```

```
     reg [7:0] input_memory [0:99]; // Adjust size as needed
     integer i;

     // Instantiate the module under test
     arithmetic_filter_image_processing uut (
       .clk(clk),
       .rst_n(rst_n),
       .in_data(in_data),
       .out_data(out_data)
 );

    // Clock generation
     always #((CLK_PERIOD/2)) clk = ~clk;

    initial begin
   // Open output file
   integer output_file;
   output_file = $fopen("output.hex", "w");
   if (output_file == 0) begin
       $display("Error opening output file");
       $finish;
   end

   // Reset
   rst_n = 0;
   #100;
   rst_n = 1;

   // Read hex file
   $readmemh("C:\\Users\\Linnet\\kodim24.hex", input_memory);

   // Apply input data and capture output
   for (i = 0; i < 100; i = i + 1) begin
       in_data = input_memory[i];
       #CLK_PERIOD;
       $fwrite("C:\Users\Linnet\Documents\MATLAB\kodim24.hex", "%h\n", out_data);
   end

   // Close output file
   $fclose("C:\Users\Linnet\Documents\MATLAB\kodim24.hex");

   // End simulation
   $finish;
end

endmodule
```

- **Output**



```
f2
f2
f2
f0
f0
f0
f0
f0
f0
f1
f1
f1
f0
f0
f0
ee
ee
ee
ef
ef
ef
f1
f1
f1
f0
f0
f0
ef
ef
ef
ef
ef
ef
f0
f0
f0
f1
f1
f1
f0
```

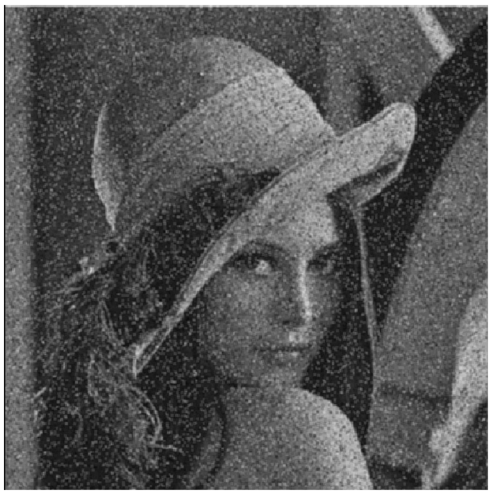
## 6.5.5 Matlab Code (Conversion of hex file to image) and its Output

- **Code**

```
% Define the image dimensions
image_width = 648;
image_height = 646;
% Read the hex file
hex_file_path = "D:\Image Processing using Verilog\output.hex";
fid = fopen(hex_file_path, 'r');
data = textscan(fid, '%s');
fclose(fid);
% Convert hexadecimal to decimal
decimal_data = hex2dec(data{1});
% Ensure the total number of elements matches the expected size
expected_elements = image_width * image_height * 3; % 3 channels (RGB)
if numel(decimal_data) < expected_elements
     error('The hex data does not contain enough elements to form the specified image’);
elseif numel(decimal_data) > expected_elements
    decimal_data = decimal_data(1:expected_elements);
end
% Reshape the data into an image matrix
 image_matrix = reshape(decimal_data, [3, image_width, image_height]);
 image_matrix = permute(image_matrix, [3, 2, 1]); % Permute to [height, width, channels]
```

```
% Flip the image vertically to correct the orientation
 image_matrix = flipud(image_matrix);

 % Convert to uint8
 image_matrix = uint8(image_matrix);

 % Display the image
 imshow(image_matrix);
```

- **Output**



# 7. ANALYSIS OF EXPERIENCE

## 7.1 Observations

### 7.1.1 Conversion Process:

- **From Image to Hex File**: The detailed process of converting an image with salt and pepper noise into a hex file using MATLAB was learnt. This involved understanding the image data structure and how to represent it in a format that Verilog can process.
- **From Hex File to Image**: The reverse process of converting the processed hex file back into an image, allowed us to visualize the effects of our Verilog-based image processing algorithms.

### 7.1.2 Image Processing with Verilog:

- Implementing the arithmetic mean filter in Verilog provided a practical understanding of how noise reduction algorithms can be synthesized at the hardware level. We observed how the filter effectively reduced salt and pepper noise by averaging the values of neighboring pixels.

### 7.1.3 Simulation and Verification:

- Simulation and verification of the Verilog design were done using Quartus Prime software. The testbench and system Verilog code were crucial in ensuring our design worked correctly before implementation.

## 7.2 Presentation of the Differences Noted Between Books and Practice

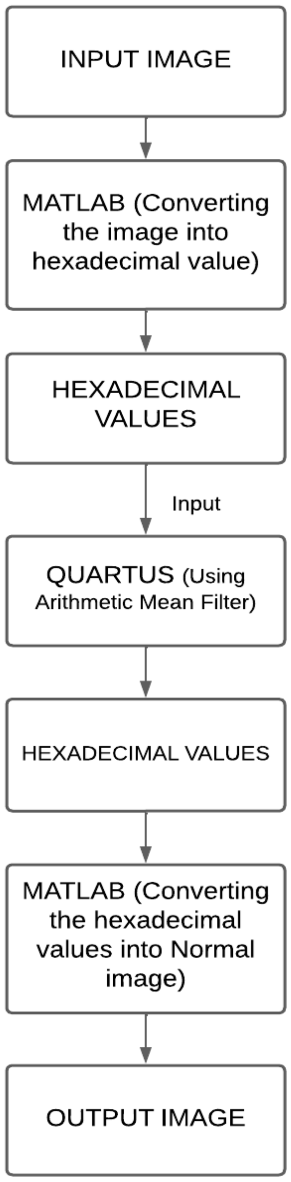
### 7.2.1 Theoretical Understanding vs. Practical Application:

- The theoretical aspects of image processing algorithms and VLSI design are often presented directly, with ideal conditions and simplified models.
- In practice, the implementation involves many additional considerations, such as handling boundary conditions, optimizing for hardware constraints, and debugging unexpected issues that arise during simulation.

### 7.2.2 Tools and Software:

- The books typically describe algorithms and processes without delving deeply into the specifics of tools like MATLAB and Quartus Prime.
- Hands-on experience with these tools revealed their complexity and the steep learning curve involved.

## 7.3 Presentation of the process


INPUT IMAGE
MATLAB (Converting the image into hexadecimal value)
HEXADECIMAL VALUES
Input
QUARTUS (Using Arithmetic Mean Filter)
HEXADECIMAL VALUES
MATLAB (Converting the hexadecimal values into Normal image)
OUTPUT IMAGE

The input image is taken in BMP format. Since an image cannot be directly processed by Verilog code, it is converted to pixels in hexadecimal format using MATLAB. This allows Verilog to process it. The hexadecimal file is then fed into Verilog code, which applies an arithmetic mean filter to suppress noise peaks, such as salt and pepper noise. It rewrites an output hexadecimal file with filtered pixel data. To visualize the filtered image, the hex file is fed into MATLAB to convert it back to BMP format.

## 7.4 Difficulties

### 7.4.1 Learning Curve:

- The transition from theoretical knowledge to practical implementation was challenging. Understanding the intricacies of MATLAB and Verilog required significant effort and time.

### 7.4.2 Debugging:

- Debugging Verilog code and ensuring correct synthesis in Quartus Prime was a complex task. Issues such as timing errors and incorrect logic were common and required meticulous troubleshooting.

### 7.4.3 File Conversion:

- Converting images to hex files and vice versa involved understanding data representation and handling large data sets efficiently. Any mistake in conversion could lead to significant errors in the output.

## 7.5 Satisfaction

### 7.5.1 Skill Development:

- This project provided a comprehensive learning experience, enhancing skills in MATLAB, Verilog, and Quartus Prime. It bridged the gap between theoretical knowledge and practical application.

### 7.5.2 Successful Implementation:

- Successfully implementing the arithmetic mean filter and seeing the tangible results in noise reduction was highly satisfying. It demonstrated the effectiveness of our design and the feasibility of VLSI-based image processing.

### 7.5.3 Real-World Application:

- Understanding how image processing algorithms can be synthesized at the hardware level broadened the perspective on the practical applications of VLSI design in various fields, including medical imaging and digital photography.

## 8.CONCLUSION

The internship at Tessolve Semiconductors was a highly enriching experience that bridged the gap between theoretical knowledge and practical application. We gained a comprehensive understanding of Verilog and SystemVerilog, delving into their syntax, structure, and practical uses. Our VLSI-based project on the synthesis of image processing algorithms provided hands-on experience with MATLAB and Quartus Prime, enhancing our skills in these tools and demonstrating the feasibility of hardware-level image processing. Successfully implementing an arithmetic mean filter to reduce noise in images allowed us to see the practical benefits of our work. This experience highlighted the importance of simulation and verification, as well as the complexities involved in transitioning from theoretical concepts to real-world applications. Overall, the training equipped us with valuable skills and insights, preparing us effectively for future engineering challenges and broadening our perspective on the applications of VLSI design in fields such as medical imaging and digital photography.

## 9. REFERENCES

1. https://ieeexplore.ieee.org/document/8784190
2. https://ieeexplore.ieee.org/document/6707301
3. https://www.sciencedirect.com/science/article/abs/pii/0141933183905367
4. https://www.semanticscholar.org/paper/A-study-on-VLSI-implementation-of-image-enhancement-Aklak-Pugazhenthi/8bb22e7a61711704fb6270b46d5ff120033c0e0e