FPGA-ACCELERATED RETINEX IMAGE ENHANCEMENT FOR LOW-LIGHT CONDITIONS

A PROJECT REPORT

Submitted by

ANUSRI G S	**111721104010**
ASWINI R	**111721104014**
EKTHA SUDHAKAR REDDY	**111721104036**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

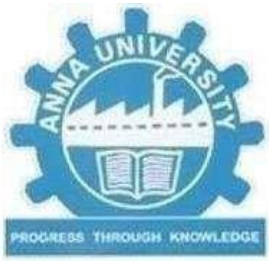
in

ELECTRONICS AND COMMUNICATION ENGINEERING

R.M.K. ENGINEERING COLLEGE

(An Autonomous Institution)

R.S.M. Nagar, Kavaraipettai-601 206



ANNA UNIVERSITY: CHENNAI 600025

MARCH 2025

R.M.K. ENGINEERING COLLEGE

(An Autonomous Institution)

R.S.M. Nagar, Kavaraipettai-601 206

BONAFIDE CERTIFICATE

Certified that this project report titled **"FPGA-ACCELERATED RETINEX IMAGE ENHANCEMENT FOR LOW-LIGHT CONDITIONS"** is the bonafide work of **ANUSRI G S (111721104010), ASWINI R (111721104014), EKTHA SUDHAKAR REDDY (111721104036)** who carried out the work under my supervision.

SIGNATURE
Dr. T.Suresh, M.E.,Ph.D.,
HEAD OF THE DEPARTMENT
Department of Electronics and Communication Engineering
R.M.K. Engineering College,
R.S.M. Nagar,
Kavaraipettai-601206.

SIGNATURE
Mr.Sivakumar A, M.E.,(Ph.D.,)
ASSOCIATE PROFESSOR
Department of Electronics and Communication Engineering
R.M.K. Engineering College,
R.S.M. Nagar,
Kavaraipettai-601206.

Submitted for the project Viva-Voce held on ……..…… at **R.M.K. Engineering College**, Kavaraipettai, Tiruvallur District – 601 206.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We would like to express our heartfelt thanks to the Almighty, our beloved parents for their blessings and wishes for successfully doing this project.

We convey our thanks to Chairman **Thiru R.S. Munirathinam and Vice Chairman Thiru R.M. Kishore** who took keen interest on us and encouraged throughout the course of study and for their kind attention and valuable suggestions offered to us.

We express our sincere gratitude to our Principal **Dr. K.A.Mohamed Junaid, M.E., Ph.D.,** for fostering an excellent climate to excel.

We are extremely thankful to **Dr. T.Suresh, M.E., Ph.D.,** Professor and Head, Department of Electronics and Communication Engineering, for having permitted us to carry out this project effectively.

We convey our sincere thanks to our mentor, **Mr.Sivakumar A, M.E., (Ph.D.,)** Supervisor Associate Professor for her extremely valuable guidance throughout the course of the project.

We are grateful to our project Co-Ordinator and all the department staff members for their intense support.

ABSTRACT

Real-time low-light image enhancement plays a crucial role in applications such as advanced driver assistance systems (ADAS), remote sensing, and object tracking. Among various techniques, Retinex-based algorithms have proven effective in improving visibility under low-light conditions by mimicking human visual perception of color and brightness.

To address this issue, we propose a novel Retinex-based algorithm that integrates a low-cost edge-preserving filter for efficient illumination estimation. Our approach reduces the computational burden by making strategic approximations, thereby minimizing hardware logic resource requirements while preserving image quality.

By optimizing the algorithm for hardware efficiency, our method facilitates real-time processing without requiring expensive or high-power computing resources. The proposed approach strikes a balance between enhancement quality and hardware feasibility, providing a practical solution for real-time low-light image enhancement in various domains.

Keywords : Real-time image enhancement, low-light enhancement, Retinex-based algorithm, illumination estimation, edge-preserving filter, hardware optimization, computational efficiency

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

KEYWORDS	ABBREVIATION
Clk	Clock
rst	Reset
valid_in	Valid Input
valid_out	Valid Output
fd_in[31:0]	FD input
fd_out[31:0]	FD output
i[31:0]	Index
num_pixels[31:0]	Number of Pixels
VGA	Video Graphics Array

CHAPTER 1

INTRODUCTION

Low-light images often suffer from poor visibility and color degradation, which can significantly impact the performance of real-time computer vision systems expected to function effectively under varying illumination conditions. The loss of information during image acquisition in low-light scenarios necessitates the integration of an efficient low-light image enhancement (LLIE) subsystem. However, implementing a reliable and cost-effective LLIE system in real-time remains challenging. Traditional methods like histogram equalization (HE) were used to adjust the dynamic range of low-light images but achieved limited enhancement as they failed to account for the overall image features. Retinex-based methods, which decompose an image into reflectance and illumination channels for enhancement and recombination, have gained more popularity. For instance, techniques use Gaussian filters for illumination estimation, while the LIME technique in enhances low-light images by imposing structural information on the illumination channel, offering an efficient approach for LLIE.

A low-light image enhancement algorithm is essential in applications where images captured under poor lighting conditions need to be improved for better visualization and analysis. These applications span various fields, including surveillance, medical imaging, remote sensing, and consumer photography. Among the many algorithms available, Retinex-based methods stand out due to their ability to mimic the human visual system, providing enhanced images with natural contrast and color balance. The Retinex theory, introduced by Edwin Land, explains how the human eye perceives color and brightness by considering the ratio of light reflected from an object to the light reflected from its surroundings.

The core principle of the Retinex algorithm is to separate the illumination and reflectance components of an image. Illumination represents the light source's effect, while reflectance corresponds to the inherent properties of the objects in the scene. By enhancing the reflectance and suppressing the influence of uneven illumination, Retinex-based methods achieve a more visually pleasing output. The algorithm also reduces noise amplification, a common challenge in low-light enhancement techniques. This makes it highly suitable for scenarios where preserving image details and color fidelity is critical.

FPGA-based implementations of Retinex algorithms provide an efficient solution for real-time applications. Field-Programmable Gate Arrays (FPGAs) are known for their parallel processing capabilities, low latency, and high flexibility. These features make FPGAs an ideal platform for computationally intensive image processing tasks. Unlike general-purpose processors, FPGAs can be tailored to execute specific algorithms with optimized hardware resources, resulting in faster processing times and lower power consumption.

The computational complexity of Retinex algorithms poses challenges for hardware implementation. Conventional methods often rely on iterative or multiscale processing, which can be resource-intensive. Therefore, a low-cost FPGA implementation requires a carefully designed architecture to balance computational efficiency, hardware resource utilization, and image quality. Optimizing the algorithm for FPGA deployment involves simplifications, pipelining, and parallelism, all of which contribute to a faster and more efficient system.

One key advantage of using FPGAs for low-light image enhancement is the ability to process high-resolution images in real-time. This capability is crucial in applications like autonomous vehicles and surveillance, where decisions must be made quickly

based on enhanced visual data. The design of a low-cost FPGA implementation involves several considerations, including resource constraints, precision, and memory requirements. For instance, fixed-point arithmetic is often preferred over floating-point to save hardware resources while maintaining acceptable accuracy. Additionally, memory usage must be carefully managed to store intermediate data and ensure smooth data flow throughout the pipeline. These design decisions significantly influence the performance and cost of the final implementation.

Retinex-based methods are not without their challenges. Over-enhancement, halo artifacts, and color distortion are common issues that must be addressed during implementation. FPGA-based designs can incorporate pre-processing and post-processing modules to mitigate these effects. For example, filtering techniques can smoothen edges and reduce noise, while color correction modules ensure that the enhanced images retain natural color tones. These additional features further enhance the overall image quality.

In practical terms, an FPGA implementation of a Retinex-based algorithm involves several stages, including illumination estimation, reflectance computation, and contrast adjustment. Each stage is mapped to dedicated hardware modules to exploit parallelism. For instance, illumination estimation can be achieved using a Gaussian filter, which is implemented as a series of parallel convolution operations. Similarly, reflectance computation and contrast adjustment are designed to maximize throughput and minimize latency.

Testing and validating the FPGA implementation is a critical step in the development process. The design must be verified using standard image datasets to ensure its effectiveness in enhancing low-light images. Metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and computational latency are

commonly used to evaluate performance. The results are compared against software-based implementations to demonstrate the benefits of the FPGA approach.

One of the challenges in low-cost FPGA designs is balancing image quality with resource usage. To address this, hardware-efficient techniques such as approximation and down sampling are often employed. These techniques reduce the complexity of computations while maintaining acceptable levels of image quality. Additionally, hardware reuse and shared resources can further minimize the cost of implementation without compromising performance.

The integration of Retinex-based image enhancement into FPGA platforms also opens doors for hybrid systems, where the FPGA works alongside other processors. For example, the FPGA can handle the most computationally demanding tasks, while a CPU or microcontroller manages high-level control and decision-making. Such hybrid systems leverage the strengths of each platform, achieving a balance between performance, flexibility, and cost.

FPGA-based implementations also offer the potential for further innovation. For instance, adaptive algorithms that adjust enhancement parameters based on the input image's characteristics can be realized on FPGAs. This dynamic approach ensures optimal enhancement for diverse lighting conditions, improving the system's overall robustness and applicability. Moreover, advancements in FPGA technology, such as the integration of machine learning capabilities, can further enhance the performance of Retinex-based methods.

The adoption of FPGAs for Retinex-based low-light image enhancement is not limited to specific industries. In healthcare, enhanced images can aid in diagnostics and analysis. In security, clearer images improve monitoring and threat detection. Even in consumer electronics, better low-light photography enhances user experience. The

versatility of FPGA-based solutions makes them a valuable tool across various domains.

The cost-effectiveness of FPGA solutions stems from their ability to integrate multiple functions on a single chip. Unlike traditional hardware implementations, which may require separate components for different tasks, FPGAs can consolidate these functions. This reduces the overall system cost, making high-performance image enhancement accessible to a broader range of applications and industries.

Another factor contributing to the low cost of FPGA implementations is the availability of open-source tools and libraries. These resources enable designers to accelerate development, reducing time-to-market and associated costs. Additionally, the growing ecosystem of FPGA platforms, ranging from entry-level to high-end devices, allows developers to choose solutions tailored to their budget and performance needs.

As technology evolves, FPGA-based Retinex implementations are likely to benefit from advancements in both hardware and algorithms. The increasing density of logic cells, improved memory architectures, and faster interconnects enhance the capabilities of FPGAs. Simultaneously, refinements in Retinex algorithms, such as better illumination estimation and noise reduction techniques, contribute to superior image quality.

The growing demand for real-time, high-quality image enhancement solutions underscores the importance of FPGA-based Retinex implementations. By combining the strengths of Retinex algorithms with the efficiency of FPGA platforms, these systems offer a compelling solution to the challenges of low-light imaging. As more applications recognize the value of enhanced visual data, the role of FPGA-based solutions is set to expand further.

CHAPTER 2

OBJECTIVE

The objective of this work is to design and implement a low-cost FPGA-based Retinex algorithm for real-time low-light image enhancement. The system aims to enhance image visibility, contrast, and color balance while minimizing common artifacts such as over-enhancement, halo effects, and color distortion. By leveraging the parallel processing capabilities of FPGAs, the proposed implementation seeks to achieve real-time performance with minimal latency, making it suitable for time-sensitive applications such as surveillance, autonomous vehicles, remote sensing, and medical imaging.

Additionally, the design focuses on optimizing hardware utilization by balancing computational complexity, memory requirements, and power consumption to ensure cost-effectiveness. Fixed-point arithmetic and efficient memory management techniques will be employed to reduce hardware overhead while maintaining high image quality. The FPGA implementation will be validated using standard image datasets, with performance evaluated based on metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), Mean Squared Error (MSE), and processing speed.

To improve adaptability, the system will incorporate dynamic parameter adjustment, allowing it to optimize enhancement based on different lighting conditions and image characteristics. The architecture will also be scalable to support various image resolutions and frame rates, ensuring compatibility with a wide range of applications. Moreover, pre-processing and post-processing techniques will be integrated to reduce noise amplification, smooth edges, and maintain natural color tones.

CHAPTER 3

EXISTING SYSTEM

3.1. EXISTING SYSTEM

Current low-light image enhancement methods rely on various computational techniques, including histogram equalization, gamma correction, deep learning models, and Retinex-based approaches. These methods are typically implemented on CPUs, GPUs, or embedded processors for applications in image processing, surveillance, autonomous vehicles, and medical imaging. The major categories of existing systems include:

1. **Histogram Equalization (HE)**
    - Enhances contrast by redistributing pixel intensity values across the entire dynamic range.
    - Common techniques include Adaptive Histogram Equalization (AHE) and Contrast-Limited Adaptive Histogram Equalization (CLAHE).
2. **Gamma Correction**
    - Adjusts image brightness by applying a non-linear power function to intensity values.
    - Suitable for quick enhancement but may result in unnatural-looking images.
3. **Deep Learning-Based Approaches**
    - Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) are used to enhance low-light images.
    - Techniques such as Zero-DCE and LLNet learn end-to-end mappings to improve brightness and contrast.

4. **Traditional Retinex-Based Methods**
    - Multi-Scale Retinex (MSR) and Single-Scale Retinex (SSR) models separate illumination and reflectance to enhance low-light images.
    - Variants like Multi-Scale Retinex with Color Restoration (MSRCR) improve color consistency.
5. **Wavelet and Frequency-Based Enhancement**
    - Image decomposition in the wavelet or frequency domain is used to enhance details while suppressing noise.

3.2. DISADVANTAGE

High Computational Complexity

- Deep learning models and advanced Retinex-based methods require high computational power, making real-time implementation difficult on resource-limited devices.

High Latency

- CPU and GPU-based implementations introduce processing delays, making them unsuitable for real-time applications such as surveillance, robotics.

Power Consumption

- GPU-based image enhancement methods consume significant power, making them inefficient for energy-constrained applications like drones and battery-operated devices.

Blocky Artifacts and Over-Enhancement

- Histogram equalization and traditional Retinex methods often introduce blocky artifacts and unnatural color saturation, leading to visually unappealing results.

CHAPTER 4

LITERATURE REVIEW

4.1. AN IMPROVED SFNet ALGORITHM FOR SEMANTIC SEGMENTATION OF LOW-LIGHT AUTONOMOUS DRIVING ROAD SCENES[1]

Abstract: "H. Wang et al" presented *an improved SFNet algorithm for semantic segmentation of low-light autonomous driving road scenes*. The outcome of the SFNet-N algorithm is a significant improvement in semantic segmentation accuracy for low-light autonomous driving road scenes. By enhancing the original SFNet algorithm with tailored adjustments for low-light conditions, such as advanced feature extraction, noise reduction, and improved illumination modeling, SFNet-N achieves better scene understanding and object recognition in challenging lighting environments. This leads to more reliable segmentation results, ensuring safer and more efficient decision-making in autonomous driving systems, even in adverse visibility scenarios.

4.2. VULNERABLE OBJECTS DETECTION FOR AUTONOMOUS DRIVING[2]

Abstract: "E. Khatab, A. Onsy, M. Varley, and A. Abouelfarag" presented the *vulnerable objects detection for autonomous driving* . The outcome of the vulnerable objects detection for autonomous driving is the development of a robust and efficient framework that enhances the detection and recognition of small, fast-moving, and potentially hazardous objects in dynamic driving environments. By leveraging advanced neural network architectures, SFNet-N improves the accuracy and reliability of identifying pedestrians, cyclists, and other vulnerable road users, even under challenging conditions such as low visibility, occlusion, and complex urban scenarios. This framework significantly contributes to the safety and decision-making capabilities

of autonomous vehicles, reducing the risk of accidents and ensuring a more reliable autonomous driving experience.

4.3. AN EXPERIMENT-BASED REVIEW OF LOW-LIGHT IMAGE ENHANCEMENT METHODS[3]

Abstract: "W. Wang, X. Wu, X. Yuan, and Z. Gao" presented *An Experiment-Based Review of Low-Light Image Enhancement Methods.* The outcome of an Experiment-Based Review of Low-Light Image Enhancement Methods *is a* comprehensive evaluation of existing enhancement techniques, with a focus on comparing their performance across diverse low-light conditions. By leveraging the SFNet-N framework, the study provides a standardized and systematic analysis of various algorithms, highlighting their strengths, weaknesses, and suitability for real-world applications. The results emphasize the importance of balancing enhancement quality, computational efficiency, and artifact suppression, offering valuable insights for selecting or designing methods tailored to specific use cases, such as surveillance, medical imaging, and photography.

4.4. PROPERTIES AND PERFORMANCE OF A CENTER / SURROUND RETINEX[4]

Abstract: "D. J. Jobson, Z. Rahman, and G. A. Woodell" presented the *properties and performance of a center/ surround retinex*. The outcome of SFNet-N, a center/surround Retinex-based method, demonstrates its ability to effectively enhance low-light images by balancing contrast, preserving natural color tones, and suppressing noise. Its unique architecture leverages a center/surround framework to separate illumination and reflectance, enabling dynamic enhancement across varying lighting conditions. SFNet-

N achieves superior performance in metrics such as PSNR, SSIM, and visual quality compared to traditional Retinex algorithms, making it well-suited for applications like surveillance, medical imaging, and autonomous systems where clarity and color fidelity are critical. Additionally, its computational efficiency ensures adaptability for real-time image processing tasks.

4.5. A MULTISCALE RETINEX FOR BRIDGING THE GAP BETWEEN COLOR IMAGES AND THE HUMAN OBSERVATION OF SCENES[5]

Abstract: "D. J. Jobson, Z. Rahman, and G. A. Woodell" presented *a multiscale retinex for bridging the gap between color images and the human observation of scenes*. The outcome of implementing a multiscale Retinex algorithm is an effective enhancement of low-light color images to closely align with human visual perception of scenes. By leveraging the Retinex theory, which separates illumination and reflectance, the algorithm bridges the gap between raw image data and natural scene observation. This results in images with improved contrast, balanced brightness, and accurate color reproduction, providing a more realistic and visually appealing representation. Such enhancement not only enhances visual quality but also improves the utility of images in applications like surveillance, medical imaging, and photography, where detail and color fidelity are critical.

CHAPTER 5

PROPOSED METHOD

5.1. PROPOSED SYSTEM

- According to the Retinex model, an image P is given as the product of illumination E and reflectance R as
- $P(x, y) = E(x, y) \times R(x, y)$
- where x and y are the pixel coordinates. The illumination channel E represents the light intensity. The Retinex model assumes that E should be devoid of textures and vary smoothly. On the contrary, the reflectance channel R comprises textures and details. Estimation of E using the Retinex model should retain edge information and smooth out textural details in it.
- For an input image P, we obtain the coarse illumination Pci by finding the maximum intensity in a local patch centred at coordinate (x,y) as shown below Pci(x, y) = max (x,y)∈ { max c∈(R,G,B) Pc (x, y)} (2) where c is the R, G, B color channel of P. However, the coarse illumination estimated using (2) cannot be used for reflectance estimation as it contains block effects. An edge-preserving filter is proposed in [8] for efficient illumination estimation. It preserves sharp edges in the illumination channel and maintains spatial smoothness. This filter performs edge preservation based on guidance image Gi, which is also the maximum channel of P.
- where is a local window centered around a pixel in Pci. Here, β and γ are defined as β(m, n) = 1, Pci(m, n) − Gi(x, y) ≥ 0 0, otherwise , (5) γ (m, n) = exp − Pci(m, n) − Gi(x, y) 2σ2 (6) where σ controls the range similarity. The filter obtains the refined illumination corresponding to a pixel in the guidance image Gi by computing the weighted average of pixels in the coarse illumination Pci over a region. The region is an odd-size square patch or window. The averaging operation

over the entire window would require many multipliers and adders. This increases the total hardware cost. However, to reduce the hardware complexity, we replaced each column of the 2-D window with the minimum value of coarse illumination pixels occupying the respective column of . This transforms the 2-D filter into a 1-D filter of size 1×r, where r is the number of columns in the given window. The output of the proposed edge-preserving filter is used as the refined illumination Fm(x, y), which is given as Fm(x, y) = r 1 βm(k)γm(k)Pci min(k) r 1 βm(k)γm(k) (7) where Pci min(k) represents the minimum value of the kth column of the 2-D window centered at location (x,y) of coarse illumination Pci(x, y). Further, βm and γm are given as βm(k) = 1, Pci min(k) − Gi(x, y) ≥ 0 0, otherwise (8) γm(k) = exp − Pci min(k) − Gi(x, y) 2σ2 .

- where c ∈ (R,G,B). The main advantages of this method are • The proposed filter preserves edges and eliminates blocky effects in Fm(x, y) by assigning lower weights to pixels that differ more from the guided image pixels, as can be observed from (9). • The proposed filter limits the lower bound of Fm(x, y) to the maximum channel using (8). This further limits the reflectance in the range [0, 1] and prevents an overflow of reflectance values, thereby preserving fine details. Fm(x, y) is further modified using the contrast stretch function given by (11) to obtain modified illumination Emod(x, y). The contrast stretch function employs the normalized pixel value of Fm(x, y). Finally, Emod(x, y) and R(x, y) are remapped using (12) to obtain the visually enhanced image PE. These operations are mathematically represented as

  - Emod(x, y) = Fm(x, y) 2 − Fm(x, y),
  - PE(x, y) = Emod(x, y) × R(x, y)

5.2. BLOCK DIAGRAM

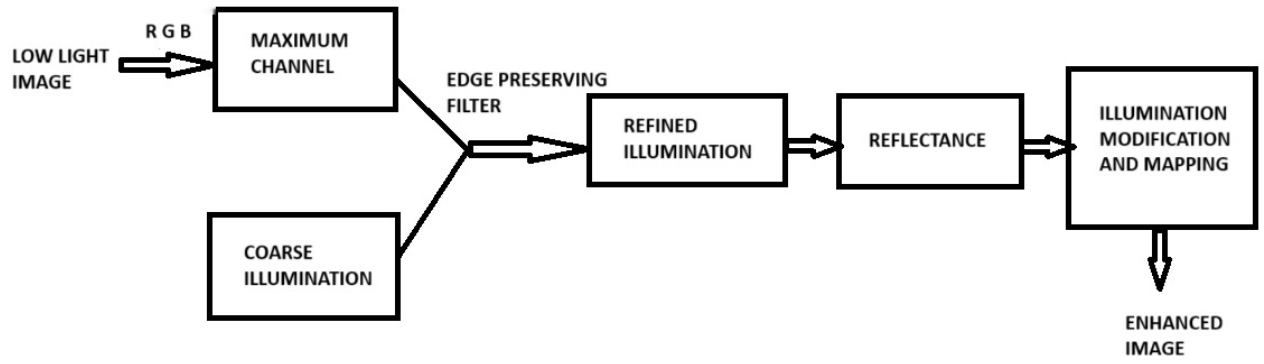


Figure No 5.1 Block Diagram

Illumination and Reflectance Characteristics

In this model, the illumination channel E(x,y)E(x, y)E(x,y) is expected to be smooth and devoid of textural details, while the reflectance channel R(x,y)R(x, y)R(x,y) retains fine textures and sharp details. The challenge lies in accurately estimating E(x,y)E(x, y)E(x,y) to ensure that edge information is preserved while textural details are smoothed out.

Coarse Illumination Estimation

For an input image P(x,y)P(x, y)P(x,y), the coarse illumination Pci(x,y)Pci(x, y)Pci(x,y) is computed by identifying the maximum intensity value in a local patch centered around the pixel coordinate (x,y)(x, y)(x,y):

Here, ccc represents the red, green, and blue color channels of P(x,y)P(x, y)P(x,y). While this method provides a starting point, the coarse illumination suffers from blocky effects that make it unsuitable for direct reflectance estimation.

Edge-Preserving Illumination Estimation

To overcome these artifacts, an edge-preserving filter is applied. This filter refines the illumination estimation by preserving edges and ensuring spatial smoothness. The filter uses a guidance image Gi(x,y)Gi(x, y)Gi(x,y), which is derived from the maximum channel of P(x,y)P(x, y)P(x,y).

The filter computes a weighted average of the coarse illumination Pci(x,y)Pci(x, y)Pci(x,y) within a local region ω\omegaω. The weights β\betaβ and γ\gammaγ are defined as: Here, σ\sigmaσ controls the range similarity, ensuring that pixels similar to the guidance image are given higher weights.

Hardware Optimization

To reduce hardware complexity, the 2D filtering operation is simplified. Instead of processing a full 2D window, the filter reduces the computation to a 1D filter by replacing each column of the window with the minimum value of the coarse illumination pixels in that column.

- **Reduction in Computational Load:** By simplifying the **2D filtering operation** to a **1D filter**, the number of required computations is significantly reduced. This optimization minimizes processing time and lowers the hardware resource utilization, making it more efficient for real-time applications.
- **Efficient Memory Utilization:** Since only a **single column of values** is considered instead of a full **2D window**, the memory required to store

intermediate filter results is reduced. This leads to **lower power consumption** and **better performance** in hardware implementations.

5.2. ADVANTAGES:

Real-Time Processing: FPGA implementation enables real-time enhancement of low-light images, making it suitable for applications like surveillance, automotive vision, and medical imaging.

Low Latency: Due to parallel processing capabilities, FPGAs significantly reduce the latency compared to CPU or GPU implementations, ensuring faster image enhancement.

Cost-Effectiveness: The hardware architecture is optimized to minimize resource usage, enabling the implementation on low-cost FPGA boards without compromising performance.

Energy Efficiency: FPGAs consume less power compared to traditional processors, making them ideal for energy-constrained applications like drones and portable imaging devices.

Edge Preservation: The proposed edge-preserving filter maintains sharpness in the enhanced image by selectively weighting pixels, leading to natural-looking results.

Elimination of Blocky Effects: The algorithm eliminates block artifacts in the illumination channel, resulting in smooth transitions and improved visual quality.

Dynamic Range Adjustment: The Retinex model's ability to separate illumination and reflectance prevents over-enhancement or loss of detail, ensuring that the enhanced image retains natural contrast and brightness.

Hardware Optimization: The use of a 1D filter in place of a 2D filter reduces the number of multipliers and adders, lowering hardware complexity and making the implementation more resource-efficient.

Scalability: The FPGA design can be easily scaled to handle different image resolutions or applied to systems with varying resource constraints.

Flexibility: FPGAs allow for reconfiguration, enabling customization of the algorithm to meet specific application requirements, such as enhancing different lighting conditions.

5.3. APPLICATIONS:

Surveillance Systems: In low-light environments, such as night-time or dimly lit areas, this algorithm enhances the visibility of objects, aiding in surveillance for security cameras, public safety, and crime detection.

Autonomous Vehicles: Low-light image enhancement is crucial for autonomous vehicles that need to operate efficiently in challenging lighting conditions, such as night driving or tunnels, to improve object detection and navigation.

Medical Imaging: The Retinex-based enhancement can be applied in medical imaging systems, such as X-rays, MRIs, or endoscopes, where visibility of fine details is critical, even in low-light conditions or images with poor contrast.

Satellite and Aerial Imaging: In satellite or drone-based imaging, low-light image enhancement can help improve the clarity and detail of images captured in dim conditions, like during dawn or dusk, or in areas with poor lighting.

Underwater Imaging: In underwater applications where light penetration is limited, enhancing images to improve contrast and visibility is essential for tasks

like marine biology studies, underwater exploration, and remote inspection of submerged structures.

Night Vision Systems: The algorithm is ideal for enhancing images from night vision cameras, which are used in military operations, wildlife observation, or search and rescue missions.

Robotics: For robots operating in low-light environments (e.g., warehouse robots, mobile robots in unlit areas), the enhanced image can improve object recognition and navigation.

Agricultural Drones: Drones used in precision agriculture often operate in low-light or overcast conditions. Enhanced images help in identifying crop health, detecting pests, and monitoring growth patterns.

Smartphones and Consumer Cameras: Smartphone cameras and consumer-level cameras can incorporate the Retinex-based enhancement algorithm for better image quality in low-light scenarios, improving user experience for photography at night or indoors.

Forensic Imaging: In forensics, images captured from scenes in low-light conditions need enhancement to clarify critical details, such as fingerprints, facial features, or object identification in crime investigations.

Industrial Inspection: Low-light image enhancement can improve the quality of images used in industrial applications like quality control, defect detection in manufacturing processes, or the inspection of mechanical parts in poor lighting conditions.

CHAPTER 6

XILINX AND VERILOG HDL

6.1. HISTORY OF VERILOG

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. It is rumored that the original language was designed by taking features from the most popular HDL language of the time, called HiLo, as well as from traditional computer languages such as C. At that time, Verilog was not standardized and the language modified itself in almost all the revisions that came out within 1984 to 1990.

Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog simulator sold by Gateway. The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm" which was a very efficient method for doing gate-level simulation.

The time was late 1990. Cadence Design System, whose primary product at that time included thin film process simulator, decided to acquire Gateway Automation System. Along with other Gateway products, Cadence now became the owner of the Verilog language, and continued to market Verilog as both a language and a simulator.

At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination. In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized the Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event which "opened" the language.

6.2. INTRODUCTION OF VERILOG

- HDL is an abbreviation of Hardware Description Language. Any digital system can be represented in a REGISTER TRANSFER LEVEL (RTL) and HDLs are used to describe this RTL.
- Verilog is one such HDL and it is a general-purpose language –easy to learn and use. Its syntax is similar to C.
- The idea is to specify how the data flows between registers and how the design processes the data.
- To define RTL, hierarchical design concepts play a very significant role. Hierarchical design methodology facilitates the digital design flow with several levels of abstraction.
- Verilog HDL can utilize these levels of abstraction to produce a simplified and efficient representation of the RTL description of any digital design.
- For example, an HDL might describe the layout of the wires, resistors and transistors on an Integrated Circuit (IC) chip, i.e., the switch level or, it may describe the design at a more micro level in terms of logical gates and flip flops in a digital system, i.e., the gate level. Verilog supports all of these levels.

6.3. DESIGN STYLES

Any hardware description language like Verilog can be designed in two ways: one is bottom-up design and other one is top-down design.

Bottom-Up Design:

The traditional method of electronic design is bottom-up (designing from transistors and moving to a higher level of gates and, finally, the system). But with the increase in design complexity traditional bottom-up designs have to give way to new structural, hierarchical design methods.

Top-Down Design:

For HDL representation it is convenient and efficient to adapt this design-style. A real top-down design allows early testing, fabrication technology independence, a structured system design and offers many other advantages. But it is very difficult to follow a pure top-down design. Due to this fact most designs are mix of both the methods, implementing some key elements of both design styles.

6.4. Features of Verilog HDL

- Verilog is case sensitive.
- Ability to mix different levels of abstract freely.
- One language for all aspects of design, testing, and verification.
- In Verilog, Keywords are defined in lower case.
- In Verilog, Most of the syntax is adopted from "C" language.
- Verilog can be used to model a digital circuit at Algorithm, RTL, Gate and Switch level.
- There is no concept of package in Verilog.
- It also supports advanced simulation features like TEXTIO, PLI, and UDPs.

6.5. VLSI DESIGN FLOW

The VLSI design cycle starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip. An essential step in the VLSI design cycle is verification and validation, which includes techniques like simulation, formal verification, and hardware emulation to ensure that the design meets functional and performance requirements before fabrication, reducing the risk of costly errors in later stages.

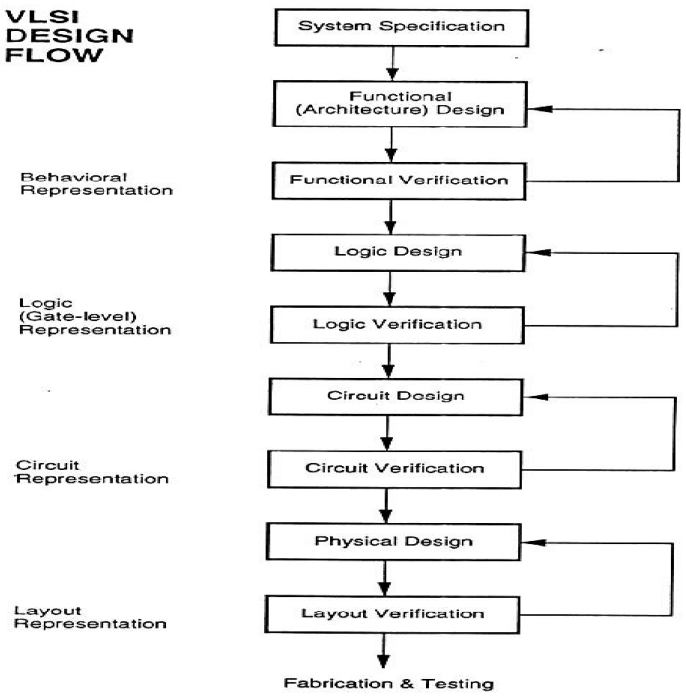


Figure No. 6.1 VLSI Design Flow

System Specification:

The first step of any design process is to lay down the specifications of the system. System specification is a high level representation of the system. The factors to be considered in this process include: performance, functionality, and physical dimensions like size of the chip.

The specification of a system is a compromise between market requirements, technology and economical viability. The end results are specifications for the size, speed, power, and functionality of the VLSI system.

Architectural Design

The basic architecture of the system is designed in this step. This includes, such decisions as RISC (Reduced Instruction Set Computer) versus CISC (Complex Instruction Set Computer), number of ALUs, Floating Point units, number and structure of pipelines, and size of caches among others. The outcome of architectural design is a Micro-Architectural Specification (MAS).

Behavioral or Functional Design:

In this step, main functional units of the system are identified. This also identifies the interconnect requirements between the units. The area, power, and other parameters of each unit are estimated. The key idea is to specify behavior, in terms of input, output and timing of each unit, without specifying its internal structure. The outcome of functional design is usually a timing diagram or other relationships between units.

Logic Design:

In this step the control flow, word widths, register allocation, arithmetic operations, and logic operations of the design that represent the functional design are derived and tested.

This description is called Register Transfer Level (RTL) description. RTL is expressed in a Hardware Description Language (HDL), such as VHDL or Verilog. This description can be used in simulation and verification

Circuit Design:

The purpose of circuit design is to develop a circuit representation based on the logic design. The Boolean expressions are converted into a circuit representation by taking into consideration the speed and power requirements of the original design. Circuit Simulation is used to verify the correctness and timing of each component

The circuit design is usually expressed in a detailed circuit diagram. This diagram shows the circuit elements (cells, macros, gates, transistors) and interconnection between these elements. This representation is also called a netlist. And each stage verification of logic is done.

Physical design:

In this step the circuit representation (or netlist) is converted into a geometric representation. As stated earlier, this geometric representation of a circuit is called a layout.

Layout is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers), which perform the intended logic function of the corresponding component. Connections between different components are also expressed as geometric patterns typically lines in multiple layers.

Layout verification:

Physical design can be completely or partially automated and layout can be generated directly from netlist by Layout Synthesis tools. Layout synthesis tools, while fast, do have an area and performance penalty, which limit their use to some designs. These are verified. Another important aspect of layout verification is Design Rule Checking **(**DRC**)**, which ensures that the layout adheres to the manufacturing constraints set by the foundry. DRC checks for violations such as minimum spacing, width, and enclosure rules, ensuring the design can be fabricated reliably without defects.

Fabrication and Testing:

Silicon crystals are grown and sliced to produce wafers. The wafer is fabricated and diced into individual chips in a fabrication facility. Each chip is then packaged and tested to ensure that it meets all the design specifications and that it functions properly.

After packaging, the chips undergo parametric and functional testing, including wafer-level testing (wafer probing) and final test to detect defects and ensure compliance with electrical and performance specifications. Burn-in testing and reliability tests may also be conducted to assess long-term stability and durability under various operating conditions.

6.6. MODULE

A module is the basic building block in Verilog. It can be an element or a collection of low level design blocks. Typically, elements are grouped into modules to provide

common functionality used in places of the design through its port interfaces, but hides the internal implementation.

Syntax:

module<module name> (<module_port_list>);

…..

<module internals> //contents of the module

….

Endmodule

Instances

A module provides a template from where one can create objects. When a module is invoked Verilog creates a unique object from the template, each having its own name, variables, parameters and I/O interfaces. These are known as *instances*.

Ports:

- Ports allow communication between a module and its environment.
- All but the top-level modules in a hierarchy have ports.
- Ports can be associated by order or by name.

You declare ports to be input, output or inout. The port declaration syntax is:

Input [range_val:range_var] list_of_identifiers;

output[range_val:range_var] list_of_identifiers;

inout[range_val:range_var] list_of_identifiers;

Identifiers

- Identifiers are user-defined words for variables, function names, module names, and instance names. Identifiers can be composed of letters, digits, and the underscore character.

- The first character of an identifier cannot be a number. Identifiers can be any length.
- Identifiers are case-sensitive, and all characters are significant.

An identifier that contains special characters, begins with numbers, or has the same name as a keyword can be specified as an escaped identifier. An escaped identifier starts with the backslash character(\) followed by a sequence of characters, followed by white space.

Keywords:

- Verilog uses keywords to interpret an input file.
- You cannot use these words as user variable names unless you use an escaped identifier.
- Keywords are reserved identifiers, which are used to define language constructs.
- Some of the keywords are always, case, assign, begin, case, end and end case etc.

Data Types:

Verilog Language has two primary data types:

- ***Nets*** - represents structural connections between components.
- ***Registers*** - represent variables used to store data.

Every signal has a data type associated with it. Data types are:

- ***Explicitly declared*** with a declaration in the Verilog code.
- ***Implicitly declared*** with no declaration but used to connect structural building blocks in the code. Implicit declarations are always net type "wire" and only one bit wide.

Register Data Types

- Registers store the last value assigned to them until another assignment statement changes their value.
- Registers represent data storage constructs.
- Register arrays are called memories.
- Register data types are used as variables in procedural blocks.
- A register data type is required if a signal is assigned a value within a procedural block
- Procedural blocks begin with keyword initial and always.

The data types that are used in register are register, integer, time and real.

6.7. MODELING CONCEPTS

Abstraction Levels:

- Behavioral level
- Register-Transfer Level
- Gate Level
- Switch level

Behavioral or algorithmic Level

- This level describes a system by concurrent algorithms (Behavioral).
- Each algorithm itself is sequential, meaning that it consists of a set of instructions that are executed one after the other.
- The blocks used in this level are ‘initial’, ‘always’ ,‘functions’ and ‘tasks’ blocks
- The intricacies of the system are not elaborated at this stage and only the functional description of the individual blocks is prescribed.
- In this way the whole logic synthesis gets highly simplified and at the same time more efficient.

Register-Transfer Level:

- Designs using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers.
- An explicit clock is used. RTL design contains exact timing possibility, operations are scheduled to occur at certain times.
- Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".

Gate Level:

- Within the logic level the characteristics of a system are described by logical links and their timing properties.
- All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates).
- It must be indicated here that using the gate level modeling may not be a good idea in logic design.
- Gate level code is generated by tools like synthesis tools in the form of netlists which are used for gate level simulation and for backend.

Switch Level:

This is the lowest level of abstraction. A module can be implemented in terms of switches, storage nodes and interconnection between them. However, as has been mentioned earlier, one can mix and match all the levels of abstraction in a design. RTL is frequently used for Verilog description that is a combination of behavioral and dataflow while being acceptable for synthesis.

CHAPTER 7

SOFTWARE IMPLEMENTATION

7.1. XILINX VERILOG HDL USER GUIDE

Getting started

First we need to download and install Xilinx and ModelSim. These tools both have free student versions. Please accomplish Appendix B, C, and D in that order before continuing with this tutorial. Additionally if you wish to purchase your own Spartan3 board, you can do so at Digilent's Website. Digilent offers academic pricing. Please note that you must download and install Digilent Adept software. The software contains the drivers for the board that you need and also provides the interface to program the board.

Introduction

Xilinx Tools is a suite of software tools used for the design of digital circuits implemented using Xilinx Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). The design procedure consists of (a) design entry, (b) synthesis and implementation of the design, (c) functional simulation and (d) testing and verification. Digital designs can be entered in various ways using the above CAD tools: using a schematic entry tool, using a hardware description language (HDL) – Verilog or VHDL or a combination of both. In this lab we will only use the design flow that involves the use of Verilog HDL.

The CAD tools enable you to design combinational and sequential circuits starting with Verilog HDL design specifications. The steps of this design procedure are listed below:

1. Create Verilog design input file(s) using template driven editor.

2. Compile and implement the Verilog design file(s).

3. Create the test-vectors and simulate the design (functional simulation) without using a PLD (FPGA or CPLD).

4. Assign input/output pins to implement the design on a target device.

5. Download bitstream to an FPGA or CPLD device.

6. Test design on FPGA/CPLD device

A Verilog input file in the Xilinx software environment consists of the following segments:

Header: module name, list of input and output ports.

Declarations: input and output ports, registers and wires.

Logic Descriptions: equations, state machines and logic functions.

End: endmodule

All your designs for this lab must be specified in the above Verilog input format. Note that the *state diagram* segment does not exist for combinational logic designs.

Programmable Logic Device: FPGA

In this lab digital designs will be implemented in the Basys2 board which has a Xilinx Spartan 3E –XC3S250E FPGA with CP132 package. This FPGA part belongs to the Spartan family of FPGAs. These devices come in a variety of packages. We will be using devices that are packaged in 132 pin package with the following part number: XC3S250E-CP132.

Creating a New Project

Creating Projects You can use the New Project wizard to easily create different types of projects in the Vivado IDE. To open the New Project wizard, select File > New Project. This wizard enables you to specify a project location and name and create the types of projects shown in below figure

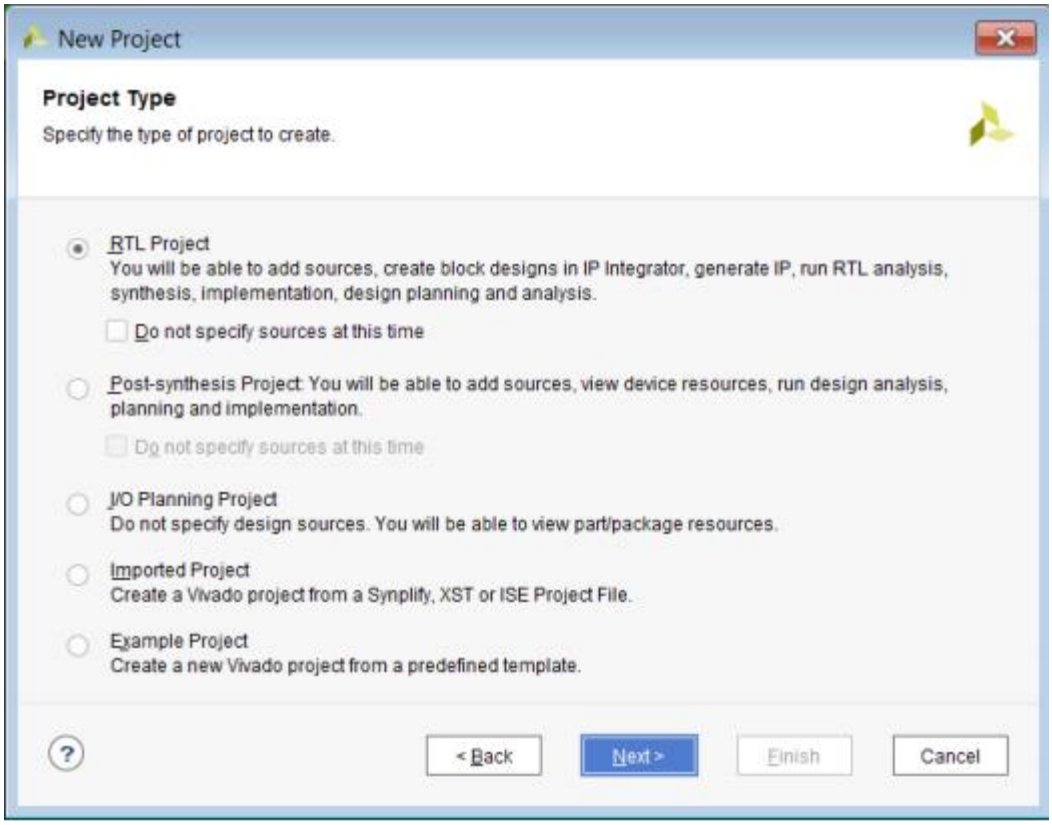


Figure No 7.1 New Project Wizard—Project Type Page

Project Name: Write the name of your new project which is user defined.

Project Location: The directory where you want to store the new project in the specified location in one of your drive. In above window they are stored in location c drive which is not correct, the location of software and code should not be same location and Clicking on NEXT.

For each of the properties given below, click on the '**value**' area and select from the list of values that appear.

- **Device Family**: Family of the FPGA/CPLD used. In this laboratory we will be using the Spartan 3E FPGA.
- **Device**: The number of the actual device. For this lab you may enter **XC3S250E** (this can be found on the attached prototyping board)

- **Package**: The type of package with the number of pins. The Spartan FPGA used in this lab is packaged in CP132 package.
- **Speed Grade**: The Speed grade is "-4".
- **Synthesis Tool**: **XST** [VHDL/Verilog]
- **Simulator:** The tool used to simulate and verify the functionality of the design. Then click on **NEXT** to save the entries.

Opening Designs:

Use the Flow Navigator or Flow menu to select the following commands:

- Open Elaborated Design
- Open Synthesized Design
- Open Implemented Design

The Flow > Open Implemented Design command populates the Vivado IDE as shown in below figure.

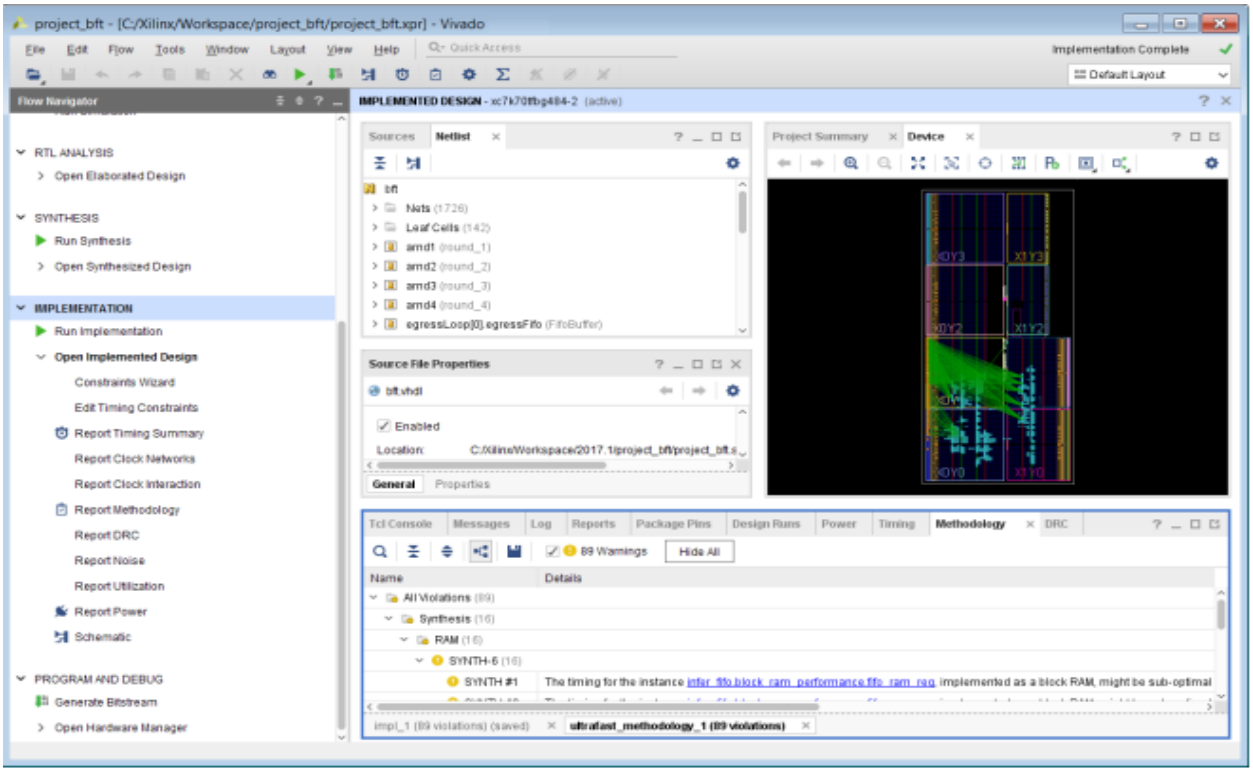


Figure No.7.2 Implemented design

All project files such as schematics, netlists, Verilog files, VHDL files, etc., will be stored in a subdirectory with the project name.

In order to open an existing project in Xilinx Tools, select **File->Open Project** to show the list of projects on the machine. Choose the project you want and click **OK**.

If creating a new source file, click on the NEW SOURCE.

Creating a Verilog HDL input file for a combinational logic design:

In this lab we will enter a design using a structural or RTL description using the Verilog HDL. You can create a Verilog HDL input file (**.v**file) using the HDL Editor available in the Xilinx Vivado Tools (or any text editor).

In the previous window, click on the NEW SOURCE

(Note: "**Add to project**" option is selected by default. If you do not select it then you will have to add the new source file to the project manually.)

Select **Verilog Module** and in the "File Name:" area, enter the name of the Verilog source file you are going to create. Also make sure that the option **Add to project** is selected so that the source need not be added to the project again. Then click on **Next** to accept the entries.

In the **Port Name** column, enter the names of all input and output pins and specify the **Direction** accordingly. A Vector/Bus can be defined by entering appropriate bit numbers in the **MSB/LSB** columns. Then click on **Next>**to get a window showing all the new source information above the window. If any changes are to be made, just click on **<Back** to go back and make changes. If everything is acceptable, click on **Finish > Next > Next > Finish** to continue.

Once you click on **Finish**, the source file will be displayed in the sources window in the **Project Navigator.** If a source has to be removed, just right click on the source file in the **Sources in Project** window in the **Project Navigator** and select **remove** in that.

Then select **Project -> Delete Implementation Data** from the Project Navigator menu bar to remove any related files.

Editing the Verilog source file

The source file will now be displayed in the **Project Navigator** window (Figure 8). The source file window can be used as a text editor to make any necessary changes to the source file. All the input/output pins will be displayed. Save your Verilog program periodically by selecting the **File->Save** from the menu. You can also edit Verilog programs in any text editor and add them to the project directory using "Add Copy Source".

Here in the above window we will write the Verilog programming code for specified design and algorithm in the window.

After writing the programming code we will go for the synthesis report.

1. **Syntax and Error Checking:** Before proceeding to synthesis, it is important to check for syntax errors and logical mistakes in the Verilog code. Most Verilog development environments provide a syntax checker and simulator to identify and correct errors before compilation.
2. **Simulation and Functional Verification:** Once the Verilog code is written, it should be tested using a testbench to verify its functionality. This involves running simulation tools to check whether the design behaves as expected under different input conditions before moving to the synthesis stage.

Configuring Project Settings

You can configure the Project Settings in the Settings dialog box to meet your design needs. These settings include general settings, related to the top module definition and language options, as well as simulation, elaboration, synthesis, implementation, bitstream, and IP settings.

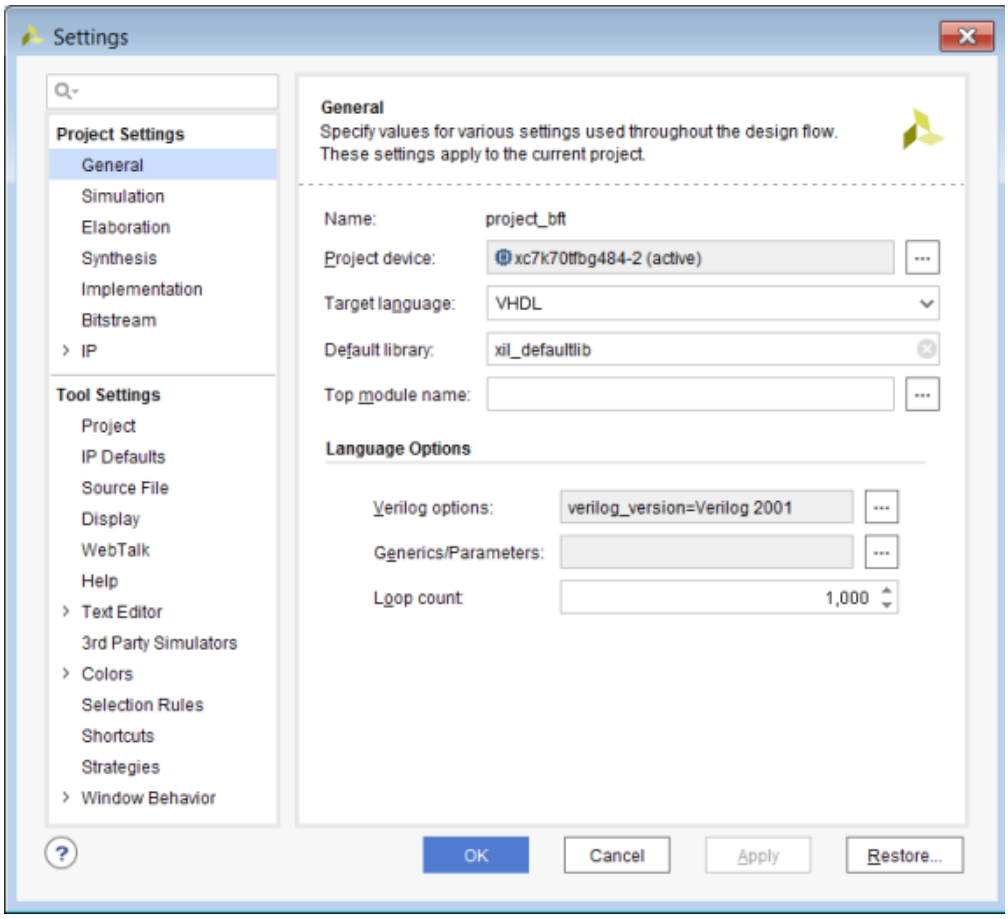


Figure No.7.3 Settings Dialog Box—Project Settings General Category

To open the Settings dialog box, use any of the following methods:

• In the Flow Navigator Project Manager section, click Settings.

• Select Tools > Settings.

• In the main toolbar, click the Settings toolbar button .

• In the Project Summary, click the Edit link next to Settings.

Synthesis and Implementation of the Design:

The design has to be synthesized and implemented before it can be checked for correctness, by running functional simulation or downloaded onto the prototyping board. With the top-level Verilog file opened (can be done by double-clicking that file) in the HDL editor window in the right half of the Project Navigator, and the view of

the project being in the **Module view** , the **implement design** option can be seen in the **process view**. **Design entry utilities** and **Generate Programming File** options can also be seen in the process view.

To synthesize the design, double click on the **Synthesize Design** option in the **Processes window**.

To implement the design, double click the **Implement design** option in the **Processes window**. It will go through steps like **Translate, Map and Place & Route**. If any of these steps could not be done or done with errors, it will place a **X** mark in front of that, otherwise a tick mark will be placed after each of them to indicate the successful completion

After synthesis right click on synthesis and click view text report in order to generate the report of our design.

7.2 XILINX VIVADO Simulation Procedure:

After completion of synthesis we will go simulation in order to verify the functionality of the implemented design.

Click on **Run Simulation** and set the module that is need to Run

Next **double click on Run Behavioral Simulation** to check the errors. If no errors are found then double click on simulate behavioral model to get the output waveforms.

After clicking on **the simulated behavioral model**, the simulation window will appear to pass the input values by making force constant and if it is clock by making force clock. Mention the simulation period and run for a certain time and results will appear as shown in the following window. Verify the results to the given input values.

1. **Adding Testbench for Verification:** Before running the simulation, a testbench must be created to apply test inputs and observe the output waveforms. The testbench provides stimulus signals to the design and helps in verifying the functionality of the implemented logic.

2. **Adjusting Simulation Settings:** The simulation settings such as time resolution, waveform display, and signal probing can be configured in Vivado to analyze critical signals and debug any potential issues. Users can zoom in on waveforms, add markers, and measure delays for better analysis of circuit behavior.

Using the Schematic Window:

You can generate a Schematic window for any level of the logical or physical hierarchy. You can select a logic element in an open window, such as a primitive or net in the Netlist window, and use the Schematic command in the popup menu to create a Schematic window for the selected object.

An elaborated design always opens with a Schematic window of the top-level of the design, as shown in below figure.

- **Navigating the Design Hierarchy:** The Schematic Window allows users to explore different levels of the design hierarchy, from the top-level module down to lower-level components and primitives. This helps in understanding the logical connections between different modules and verifying the design structure.

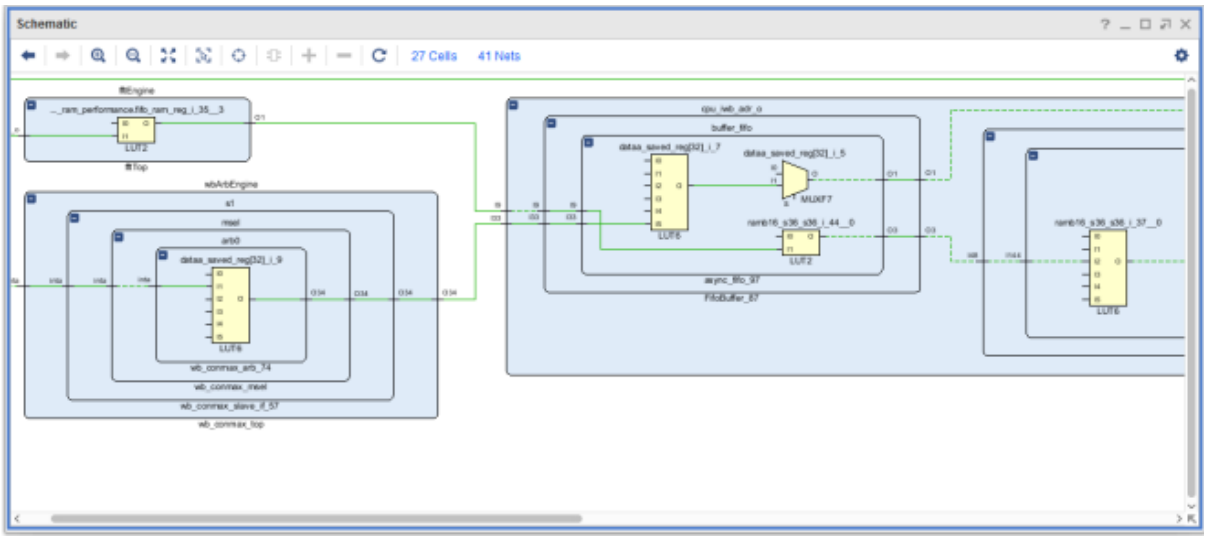


Figure No.7.4 Schematic window

Using the Project Summary

The Vivado IDE includes an interactive Project Summary (Figure 3-11) that updates dynamically as design commands are run and the design progresses through the design flow. It provides project and design information, such as the project part, board, and state of synthesis and implementation.

It also provides links to detailed information, such as links to the Messages and Reports windows as well as the Settings dialog box.

As synthesis and implementation complete, DRC violations, timing values, utilization percentages, and power estimates are also populated. To open the Project Summary, do either of the following:

- Select Window > Project Summary.
- Click the Project Summary toolbar button.

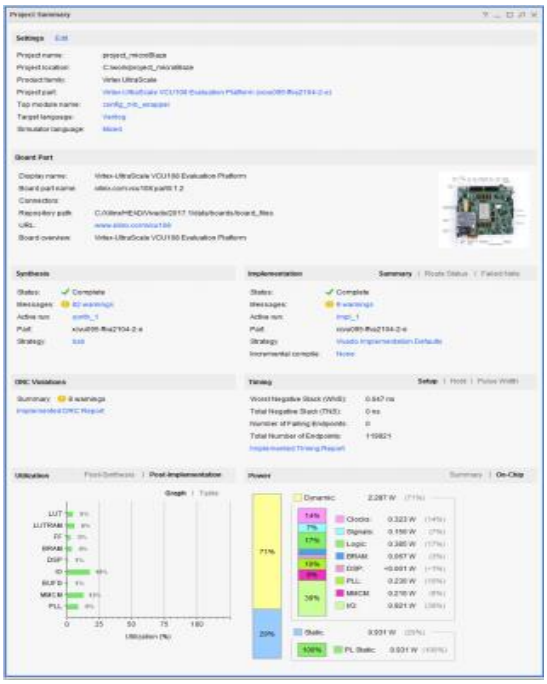


Figure No.7.5 Project Summary

CHAPTER 8

HARDWARE DESCRIPTION

8.1. NEXYS 4 Xilinx Artix-7

The NEXYS 4 FPGA development board features a diverse set of pins and connectors that allow seamless interfacing with external devices and peripherals. It includes multiple Pmod connectors, which provide standardized 12-pin interfaces for expansion modules such as sensors, displays, and communication devices. The board also has 16 slide switches and five push buttons that serve as direct user inputs for FPGA-based designs. Additionally, it features 16 LEDs for debugging and status indication, along with a four-digit 7-segment LED display for numerical or alphanumeric output. For display applications, the board includes a VGA connector with dedicated RGB and synchronization pins. A 3.5mm audio jack is present for sound output, which can be driven by the FPGA using PWM or DAC-based techniques. The USB interface enables programming and serial communication, while the Ethernet port allows for connectivity.

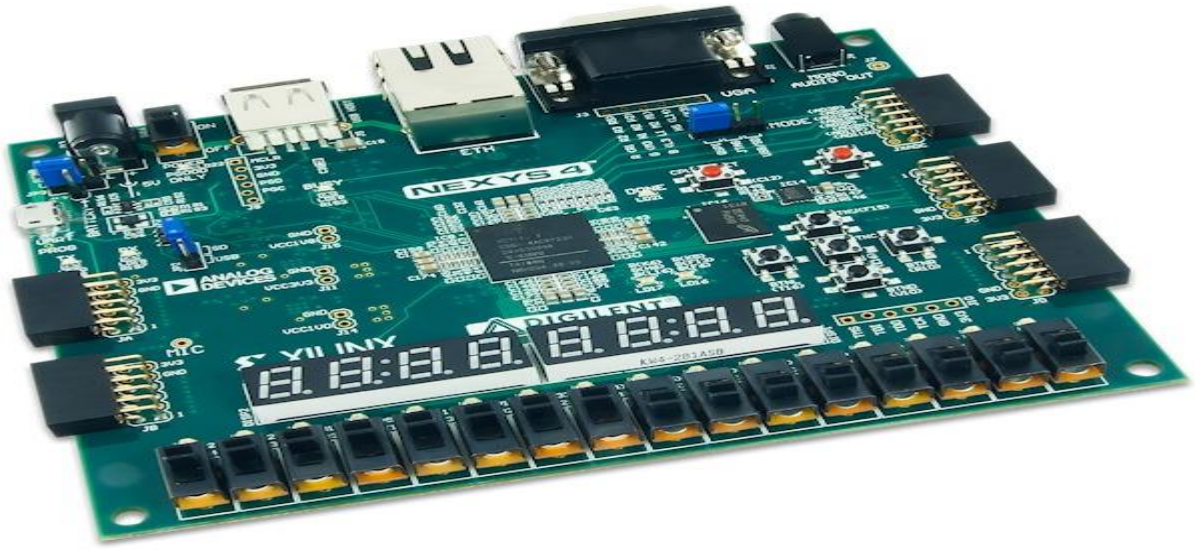


Figure No.8.1 Digilent NEXYS4 Xilinx Artix-7 FPGA Board

8.2. VGA Controller

A VGA (Video Graphics Array) controller cable is used to transmit analog video signals from a computer or other video source to a display device such as a monitor, projector, or television. It consists of a 15-pin connector that carries red, green, and blue (RGB) color signals along with horizontal and vertical synchronization signals, allowing the display to properly render the image. The VGA cable is commonly used with older display systems but is still found in some modern setups for compatibility with legacy devices. Although digital interfaces like HDMI and DisplayPort have largely replaced VGA, it remains useful in situations where analog video output is required.

Figure No.8.2 VGA Controller

CHAPTER 9

RESULTS

9.1. Simulation waveform

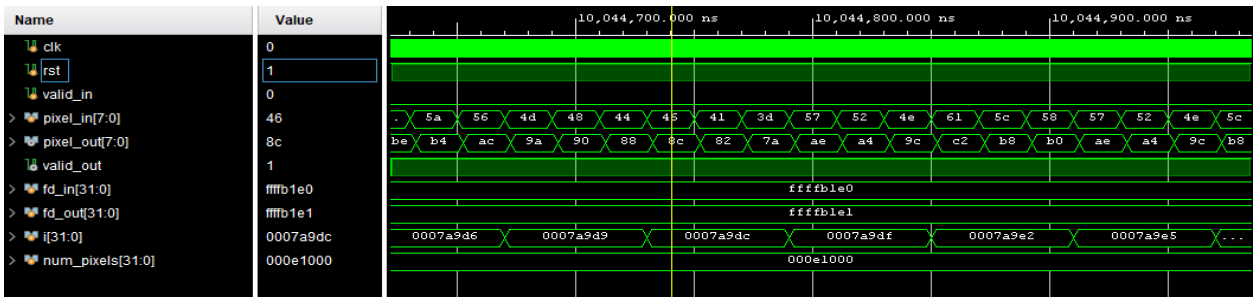


Figure No.9.1 Simulation waveform

In Figure No.9.1 simulation waveform shown is an FPGA-accelerated Retinex-based image enhancement process, specifically targeting low-light conditions.

1. Clock (clk): The main clock signal driving the circuit.
2. Reset (rst): When high (value 1), the system is reset, which initializes the registers and variables.
3. Valid Input (valid_in): Indicates when an input pixel is available for processing.
4. Pixel Input (pixel_in[7:0]): Represents the 8-bit pixel values being fed into the enhancement pipeline.
5. Pixel Output (pixel_out[7:0]): Represents the processed 8-bit pixel values after enhancement.
6. Valid Output (valid_out): Signals when the output pixel is valid and ready to be read.
7. FD Input (fd_in[31:0]) & FD Output (fd_out[31:0]): Representing floating-point data or intermediate filter values used in the enhancement process.

8. Index (i[31:0]): Indicate the current pixel index or processing step.
9. Number of Pixels (num_pixels[31:0]): Stores the total number of pixels being processed.

Observing the Simulation

- The reset (rst) is high, indicating the system is being initialized.
- Once the reset is de-asserted (rst goes low), valid_in starts toggling, meaning pixel data is flowing into the enhancement module.
- Pixel input (pixel_in) values are changing, representing different pixel intensities from the low-light image.
- Pixel output (pixel_out) values also change, showing the transformation applied to enhance the brightness and contrast.
- The valid_out signal goes high, ensuring the processed pixel data is valid.

Retinex-based image enhancement works by:

1. Decomposing the image into illumination and reflectance components.
2. Applying contrast enhancement to improve visibility in dark regions.
3. Reconstructing the enhanced image while preserving details.

From the simulation:

- The input pixels are darker values (mostly in the lower range).
- The output pixels have higher values, suggesting the system is effectively enhancing brightness and contrast.
- The fd_in and fd_out signals likely represent intermediate filter results used for illumination correction.

9.2. RTL schematic

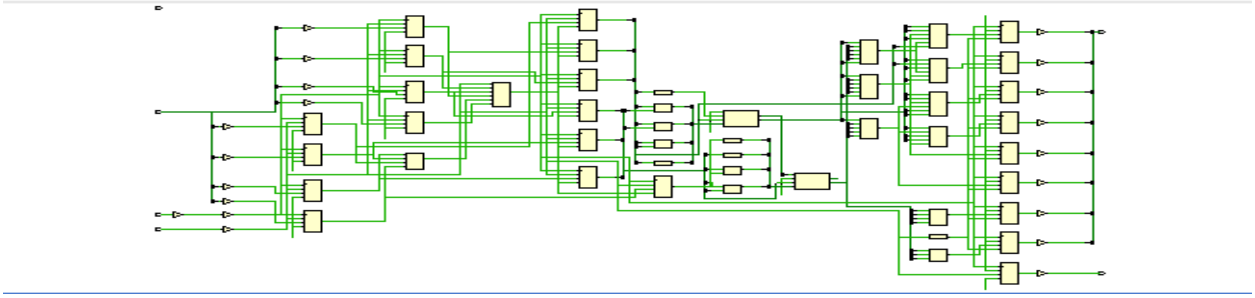


Figure No.9.2 RTL schematic

From Figure No.9.2 this RTL schematic is an FPGA-synthesized hardware representation of your Retinex-based image enhancement module. It includes a mix of combinational and sequential logic blocks designed to process image pixels in a structured pipeline, optimizing performance for low-light enhancement applications. If you want a deeper analysis, you can check the **technology schematic** (post-synthesis) to see how the logic maps to FPGA resources such as LUTs, FFs, and BRAMs.

9.3. Power

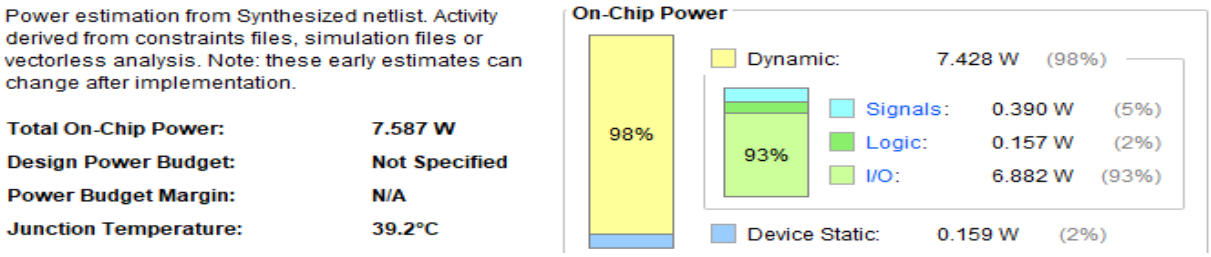


Figure No.9.3 Power

From the above Figure No.9.3 this image provides **power estimation details** for an FPGA design based on a synthesized netlist.

1. **Total On-Chip Power: 7.587 W**
    - This is the estimated power consumption of the entire FPGA design.
    - It includes **both dynamic and static power**.

2. **Dynamic Power: 7.428 W (98%)**
    - Dynamic power is the major contributor, consuming **98% of total power**.
    - It arises from **signal transitions, logic switching, and I/O activity**.
    - The breakdown:
        - **Signals: 0.390 W (5%)** – Power used in signal routing.
        - **Logic: 0.157 W (2%)** – Power consumed by internal combinational and sequential logic.
        - **I/O: 6.882 W (93%)** – The highest contributor, meaning the design heavily relies on external communication.
3. **Device Static Power: 0.159 W (2%)**
    - This is the leakage power consumed when the FPGA is powered but idle.
4. **Junction Temperature: 39.2°C**
    - This is the estimated FPGA temperature based on power consumption.
    - It is within a safe operating range, but higher power could increase thermal concerns.

9.4. Area

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (400)	BUFGCTRL (32)
N image_processing	20	24	19	1

Figure No.9.4 Area

Analysis and observation from the above Figure No.9.4,

- **Very low FPGA resource usage**: The image processing module is lightweight and does not stress FPGA resources.
- **Higher I/O usage relative to logic**: Suggests that the module primarily deals with data transfer rather than intensive computation.
- **Single clock domain**: Only one BUFGCTRL is used, implying a simple clocking scheme.

Potential Optimizations:

- If the design expands (e.g., more image processing functions), LUT and register usage might increase.
- Power consumption is likely low due to minimal resource utilization.
- Consider using **block RAM (BRAM) or DSP slices** for more complex image processing tasks.

9.5. Delay

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay
Path 1	∞	2	2	1	pixel_out_reg[0]/C	pixel_out[0]	3.521
Path 2	∞	2	2	1	pixel_out_reg[1]/C	pixel_out[1]	3.521
Path 3	∞	2	2	1	pixel_out_reg[2]/C	pixel_out[2]	3.521
Path 4	∞	2	2	1	pixel_out_reg[3]/C	pixel_out[3]	3.521

Figure No.9.5 Delay

From above Figure No.9.5 the observation will be,

No timing violations: The slack being ∞ confirms that the timing constraints are met with ease.

Uniform delay: All four paths have an identical delay of **3.521 ns**, indicating **consistent timing across multiple pixel outputs**.

Minimal logic depth (2 levels): The design is optimized and does not introduce excessive combinational logic between registers and outputs.

CHAPTER 10

CONCLUSION

In conclusion, real-time low-light image enhancement is critical for a wide range of applications, particularly in fields like advanced driver assistance systems, remote sensing, and object tracking. Retinex-based algorithms have proven to be effective in simulating human visual perception to enhance visibility in challenging lighting conditions. However, their complex mathematical computations often present significant challenges in hardware implementation, making them resource-intensive. To address this, our proposed approach introduces a novel Retinex-based algorithm that incorporates a low-cost edge-preserving filter, optimizing the illumination estimation process. By making approximations to reduce hardware complexity, our method ensures an efficient, cost-effective solution while preserving image quality, making it suitable for real-time implementation in practical scenarios.

REFERENCES:

[1] H. Wang et al., “SFNet-N: An improved SFNet algorithm for semantic segmentation of low-light autonomous driving road scenes,” IEEE Trans. Intell. Transp. Syst., vol. 23, no. 11, pp. 21405–21417, Nov. 2022.

[2] E. Khatab, A. Onsy, M. Varley, and A. Abouelfarag, “Vulnerable objects detection for autonomous driving: A review,” Integration, vol. 78, pp. 36–48, May 2021.

[3] W. Wang, X. Wu, X. Yuan, and Z. Gao, “An experiment-based review of low-light image enhancement methods,” IEEE Access, vol. 8, pp. 87884–87917, 2020.

[4] D. J. Jobson, Z. Rahman, and G. A. Woodell, “Properties and performance of a center/surround retinex,” IEEE Trans. Image Process., vol. 6, pp. 451–462, 1997.

[5] D. J. Jobson, Z. Rahman, and G. A. Woodell, “A multiscale retinex for bridging the gap between color images and the human observation of scenes,” IEEE Trans. Image Process., vol. 6, pp. 965–976, 1997.

[6] X. Guo, Y. Li, and H. Ling, “LIME: Low-light image enhancement via illumination map estimation,” IEEE Trans. Image Process., vol. 26, pp. 982–993, 2017.

[7] Y. Shin, S. Jeong, and S. Lee, “Efficient naturalness restoration for non-uniform illumination images,” IET Image Process., vol. 9, no. 8, pp. 662–671, 2015.

[8] Y. Gao, H.-M. Hu, B. Li, and Q. Guo, “Naturalness preserved nonuniform illumination estimation for image enhancement based on retinex,” IEEE Trans. Multimedia, vol. 20, no. 2, pp. 335–344, Feb. 2018. [9] C. Wei, W. Wang, W. Yang, and J. Liu, “Deep retinex decomposition for low-light enhancement,” in Proc. Brit. Mach. Vis. Conf., 2018, pp. 1–12.

[10] Y. Jiang et al., “EnlightenGAN: Deep light enhancement without paired supervision,” IEEE Trans. Image Process., vol. 30, pp. 2340–2349, 2021.

[11] R. Liu, L. Ma, J. Zhang, X. Fan, and Z. Luo, “Retinex-inspired unrolling with cooperative prior architecture search for low-light image enhancement,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2021, pp. 10561–10570.

[12] G.-D. Fan, B. Fan, M. Gan, G.-Y. Chen, and C. L. P. Chen, “Multiscale low-light image enhancement network with illumination constraint,” IEEE Trans. Circuits Syst. Video Technol., vol. 32, no. 11, pp. 7403–7417, Nov. 2022.

[13] D. I. Ustukov, Y. R. Muratov, and V. N. Lantsov, “Modification of retinex algorithm and its stream implementation on FPGA,” in Proc. 6th Mediterr. Conf. Embed. Comput. (MECO), 2017, pp. 1–4.

[14] J. W. Park et al., “A low-cost and high-throughput FPGA implementation of the retinex algorithm for real-time video enhancement,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 28, no. 1, pp. 101–114, Jan. 2020.

[15] P. Ambalathankandy, A. Horé, and O. Yadid-Pecht, “An FPGA implementation of a tone mapping algorithm with a halo-reducing filter,” J. Real-Time Image Process., vol. 16, no. 4, pp. 1317–1333, Aug 2019 (PRIME'20

.