International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Power – Efficient FPGA Implementation of Retinex – Based Low-Light Image Enhancement Using Flip-Flop Clock Gating

M. Mohan Sai, L. Sivani, S. Keerthi, B. Muralikrishna, P. Roopchand

B-Tech in Electronics and Communication Engineering, Department of Electronics and Communication Engineering, Sanketika Vidya Parishad Engineering College

Mrs. N. Ananda Kumari

Assistant Professor, Department of Electronics and Communication Engineering, Sanketika Vidya Parishad Engineering College

Abstract -
Low-light image enhancement is an essential task in modern image processing applications such as surveillance, autonomous vehicles, and medical imaging. Retinex-based algorithms are widely used due to their ability to simulate human visual perception by separating illumination and reflectance components. However, their hardware implementation is complex and power-consuming. This paper proposes a power-efficient FPGA implementation of a Retinex-based low-light image enhancement system using flip-flop-based clock gating. The proposed approach incorporates an edge-preserving filter for efficient illumination estimation and reduces computational complexity through hardware optimization. Clock gating minimizes unnecessary switching activity, significantly reducing power consumption. Experimental results demonstrate that the proposed system achieves substantial power reduction while maintaining image quality and real-time performance.

Key Words: FPGA, Retinex, Low-Light Enhancement, Clock Gating, Verilog, Power Optimization

1. INTRODUCTION

The Low-light images often suffer from poor visibility, noise, and reduced contrast, which affect the performance of vision-based systems. Enhancing such images is essential in applications like surveillance, autonomous driving, and medical diagnostics. Traditional methods such as histogram equalization fail to preserve image details and often produce over-enhanced outputs. Retinex-based algorithms provide a better solution by separating an image into illumination and reflectance components, thereby improving brightness and contrast while preserving natural appearance. Field Programmable Gate Arrays (FPGAs) offer advantages such as parallel processing, low latency, and reconfigurability, making them suitable for real-time image processing. However, implementing Retinex algorithms on FPGA involves high computational complexity and power consumption. To overcome these challenges, this paper proposes a power-efficient FPGA implementation using flip-flop-based clock gating, which reduces dynamic power consumption while maintaining performance.

2. LITERATURE SURVEY

Low-light image enhancement has gained significant attention due to its importance in applications such as surveillance, medical imaging, and autonomous systems. Traditional techniques like histogram equalization and gamma correction were initially used to improve image brightness and contrast. However, these methods often produce over-enhanced images, loss of details, and unnatural colour distortion, limiting their effectiveness. To address these limitations, Retinex-based methods were introduced, which model human visual perception by separating an image into illumination and reflectance components. Techniques such as Single-Scale Retinex (SSR) and Multi-Scale Retinex (MSR) improve contrast and dynamic range but suffer from high computational complexity and halo artifacts. Later methods like LIME improved illumination estimation using structural information, resulting in better visual quality. Recent advancements include deep learning approaches such as Deep Neural Networks (DNNs) and Generative Adversarial Networks (GANs), which provide superior enhancement results. However, these methods require large datasets and high computational resources, making them unsuitable for real-time embedded systems. FPGA-based implementations offer real-time processing capabilities but often face challenges such as high power consumption and hardware complexity. Therefore, there is a need for a power-efficient solution, which motivates the proposed FPGA-based Retinex enhancement using clock gating techniques.

3. METHODOLOGY

3.1 Proposed Methodology:

The proposed methodology presents a power-efficient FPGA-based implementation of a Retinex-based low-light image enhancement system. The input image is processed by estimating illumination using the maximum RGB channel, followed by refinement through an edge-preserving filter to maintain important details. The Retinex model separates illumination and reflectance components to enhance brightness and contrast effectively. The design is implemented using Verilog on FPGA to achieve real-time performance. To reduce dynamic power consumption, a flip-flop based clock gating technique is incorporated, which minimizes unnecessary switching activity. This integrated approach ensures improved image quality along with efficient power utilization.

4. SYSTEM ARCHITECTURE

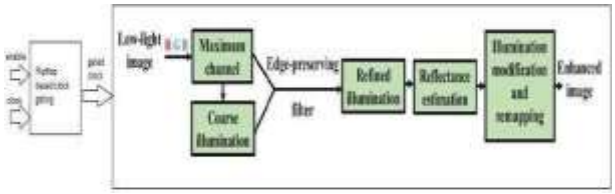

Fig1: Proposed Block Diagram

The block diagram represents a typical low-light image enhancement pipeline designed to improve visibility while preserving details. The process begins with the acquisition of a low-light image, usually captured through a camera system. This image contains RGB colour channels that are often dark and noisy due to insufficient illumination. The first major step is the extraction of the maximum channel, where, for each pixel, the maximum value among the red, green, and blue channels is selected. This helps in estimating the brightest intensity at each location and serves as a basis for computing the coarse illumination map. The coarse illumination represents an initial approximation of how light is distributed across the image, but it may contain noise and artifacts. To address this, an edge-preserving filter is applied. This filter smooths the illumination map while maintaining important edges and structural details in the image. As a result, a refined illumination map is generated, which more

accurately represents the lighting conditions without blurring object boundaries. Next, reflectance estimation is performed. In image processing, an image is often modelled as the product of illumination and reflectance. By separating these components, the algorithm isolates the true scene details (reflectance) from lighting effects. Finally, illumination modification and remapping are applied. This step adjusts the refined illumination to enhance brightness and contrast, often using techniques like gamma correction or nonlinear mapping. The modified illumination is then recombined with the reflectance to produce the enhanced image, which appears brighter, clearer, and more visually informative while preserving natural details.

5. Algorithm Description

The proposed system implements a power-efficient Retinex-based low-light image enhancement algorithm on FPGA. The methodology combines image decomposition with hardware optimization techniques.

5.1 Retinex Model:

The Retinex theory models an image as the product of illumination and reflectance:

$$[P(x,y) = E(x,y) \times R(x,y)]$$

Where:

- (P(x,y)) → Input image
- (E(x,y)) → Illumination component
- (R(x,y)) → Reflectance component

5.2 Illumination Estimation:

The coarse illumination is estimated using the maximum value among RGB channels:

$$[E(x,y) = \max \{ R(x,y), G(x,y), B(x,y) \}]$$

To refine illumination and preserve edges, an edge-preserving filter is applied:

$$[E_{ref}(x,y) = \text{Filter}(E(x,y))]$$

5.3 Reflectance Computation:

The reflectance is obtained by dividing the input image by the estimated illumination:

$$[R(x,y) = \frac{P(x,y)}{E_{ref}(x,y) + \epsilon}]$$

Where:

- (\epsilon) → Small constant to avoid division by zero

5.4 Image Enhancement:

The enhanced image is reconstructed by adjusting illumination:

$$[P_{enh}(x,y) = R(x,y) \times E_{mod}(x,y)]$$

Where:

- (E_{mod}(x,y)) → Modified illumination for brightness enhancement

5.5 Power Optimization using Clock Gating:

Dynamic power consumption in FPGA is given by:

$$[P_{dynamic} = \alpha \cdot C \cdot V^2 \cdot f]$$

Where:

- (\alpha) → Switching activity
- (C) → Load capacitance
- (V) → Supply voltage

- (f) → Clock frequency

Using clock gating, switching activity is reduced:

$$[P_{gated} < P_{dynamic}]$$

Clock gating operation:

$$[CLK_{gated} = CLK \cdot EN]$$

Where:

- (EN) → Enable signal

6. HARDWARE OPTIMIZATION TECHNIQUE

6.1 Clock Gating:

Reduction in power dissipation is an essential design issue in VLSI circuit. Few decades back designers mostly focus on area, delay and testability to optimize. While technology scaling down, we can see more power leakage and dissipation in chip. In order to reduce power dissipation and leakage power while scaling, we need to adopt the optimize techniques like clock gating, voltage scaling etc.If operations are more and more complex then power dissipation is more. The clock network is a major source of power dissipation so we can reduce significant amount of power if we can gate the clock whenever it isn't required.In our proposed work, we mainly focused on dynamic power dissipation and it reduced by making less signal activities in proposed design. The clock network is a major source of power dissipation so we can reduce significant amount of power if we can gate the clock whenever it isn't required.

6.2 Flipflop Based Clock Gating:

In many applications, latch based designs are moved to flip flop based designs. By splitting flip flop, we can see two latches from the master slave theorem. In this technique, we can see D flip flop with AND gate.

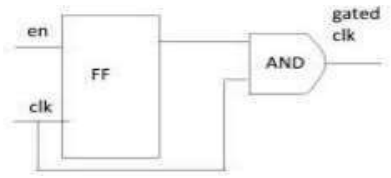

Fig2: Block Diagram of Flipflop Based Clock Gating

7. Xilinx & Verilog Implementation

7.1 History of Verilog:

Verilog is a Hardware Description Language (HDL) developed in the mid-1980s by Gateway Design Automation and later standardized as IEEE 1364. It is widely used for modeling, simulation, and synthesis of digital systems. Due to its simplicity and similarity to programming languages like C, Verilog has become a preferred choice for FPGA and VLSI design.

7.2 Design Styles:

Verilog supports three major design styles:

- **Behavioral Modeling:** Describes system functionality using procedural statements such as always and initial blocks.
- **Dataflow Modeling:** Uses continuous assignment statements (assign) to define logic flow between signals.
- **Structural Modeling:** Represents circuits by connecting logic gates and modules, similar to a schematic diagram.

These styles provide flexibility to design systems at different abstraction levels.

7.3 VLSI Design Flow:

The VLSI design process follows a structured flow:

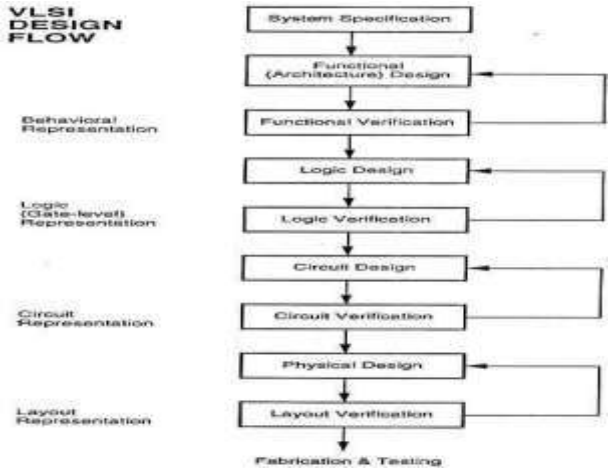

Fig3: VLSI Design Flow

1. **Design Entry:** Writing HDL code
2. **Simulation:** Verifying functionality using testbenches
3. **Synthesis:** Converting HDL into gate-level representation
4. **Implementation:** Placement and routing on FPGA
5. **Verification:** Ensuring timing and performance requirements

This flow ensures reliable and optimized hardware design.

7.4 Module:

A module is the fundamental building block in Verilog. It defines the interface (inputs and outputs) and internal functionality of a circuit. Complex systems are designed by combining multiple modules hierarchically, improving readability and reusability.

7.5 Modeling Concepts:

Verilog allows modeling at different levels:

- **Behavioral Level:** Focuses on what the system does
- **Dataflow Level:** Describes how data moves between signals
- **Structural Level:** Defines how components are interconnected

These concepts enable efficient design and optimization of digital systems.

7.6 Operators:

Verilog provides a wide range of operators that are used to perform arithmetic, logical, and bit-level operations in digital design. These operators help in describing hardware behavior efficiently and are essential for implementing combinational and sequential circuits.

7.6.1 Arithmetic Operators:

Arithmetic operators are used to perform mathematical calculations.

- Examples: +, -, *, /, %
- These operators are mainly used in data processing and computation tasks.

7.6.2 Relational Operators:

Relational operators are used to compare two values and produce a Boolean result.

- Examples: <, >, <=, >=, ==, !=

- The output is either **true (1)** or **false (0)**.

Operator	Description
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

Fig4: Relational Operators

7.6.3 Bitwise Operators:

Bitwise operators operate on individual bits of the operands.

- Examples: & (AND), | (OR), ^ (XOR), ~ (NOT)
- These are widely used in digital logic design.

Operator	Description
~	negation
&	and
|	inclusive or
^	exclusive or
^~ or ~^	exclusive nor (equivalence)

Fig5: Bitwise Operators

7.6.4 Logical Operators:

Logical operators are used to perform logical operations on expressions.

- Examples: && (AND), || (OR), ! (NOT)
- These operators are commonly used in conditional statements.

Operator	Description
!	logic negation
&&	logical and
||	logical or

Fig6: Logical Operators

7.6.5 Reduction Operators:

Reduction operators perform operations on all bits of a single operand and produce a single-bit output.

- Examples: &, |, ^, ~&, ~|, ~^
- Useful for checking conditions across multiple bits.

Operator	Description
&	and
~&	nand
|	or
~|	nor
^	xor
^~ or ~^	xnor

Fig7: Reduction Operators

7.6.6 Shift Operators:

Shift operators are used to shift bits left or right.

- Examples: << (left shift), >> (right shift)
- Used in multiplication/division and bit manipulation.

Operator	Description
<<	left shift
>>	right shift

Fig8: Shift Operators

7.6.7 Concatenation Operator:

The concatenation operator combines multiple signals into a single vector.

- Syntax: {a, b}
- Useful for grouping bits and forming larger data words.

Operator	Symbols
Unary, Multiply, Divide, Modulus	!, ~, *, /, %
Add, Subtract, Shift	+, -, <<, >>
Relation, Equality	<, >, <=, >=, ==, !=, ===, !==
Reduction	&, ~&, ^, ^~, ~^, |, ~|
Logic	&&, ||
Conditional	? :

Fig9: Operator Precedence

7.6.8 Conditional Operator:

The conditional operator is used for decision-making.

- Syntax: condition ? expression1 : expression2
- Works similar to an if-else statement.

These operators are essential for performing computations and controlling logic flow within the design.

7.7 Xilinx Verilog HDL Tutorial:

Xilinx Vivado is used for designing and implementing FPGA-based systems. It provides an integrated environment for writing HDL code, performing simulation, synthesizing the design, and generating the final bitstream. It also includes debugging and analysis tools for improving design performance.

7.8 Programmable Logic Device:

A Programmable Logic Device (PLD), such as an FPGA, allows hardware configuration based on user requirements. FPGAs consist of logic blocks, interconnects, and I/O blocks, enabling parallel processing and high-speed operation. They are widely used in real-time applications due to their flexibility and reconfigurability.

7.9 Creating a New Project:

In Xilinx Vivado, a new project is created by selecting the target FPGA device, adding HDL source files, and defining design constraints. The user configures simulation settings and organizes the design hierarchy before proceeding to synthesis and implementation.

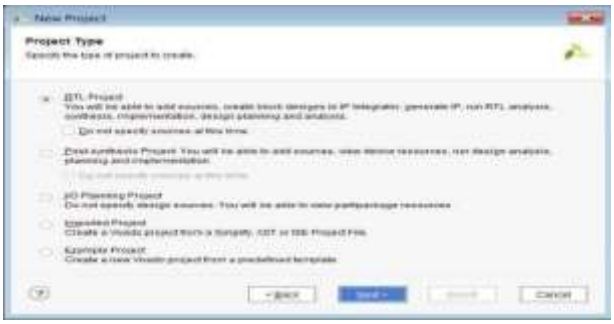

Fig10: New Project Wizard-Project Type Page

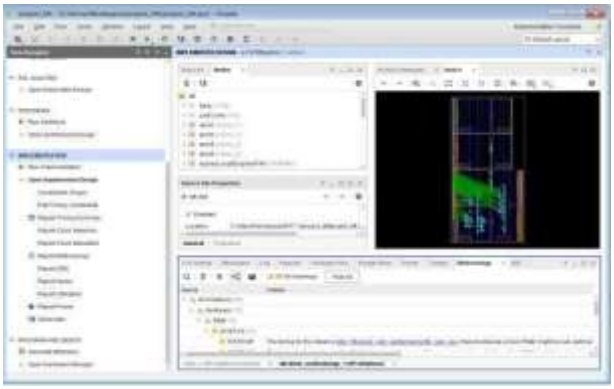

Fig11: Implemented Design

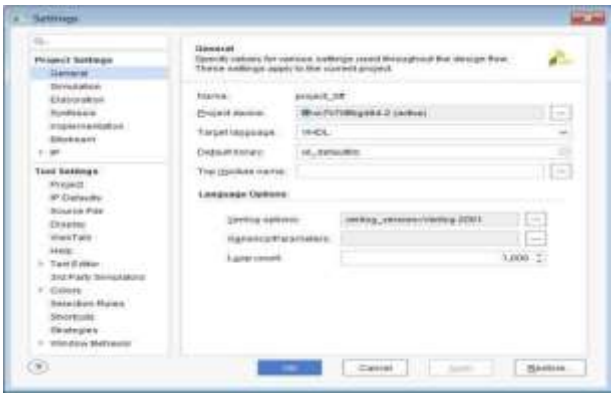

Fig12: Settings Dialog Box—Project Settings General Category

7.10 Xilinx Vivado Simulation Procedure:

Simulation is performed to verify the functionality of the design before hardware implementation. The steps include compiling the HDL code, applying test inputs using a testbench, and analyzing output waveforms. This ensures that the design operates correctly and meets required specifications. Simulation is done by using schematic window and project summary.

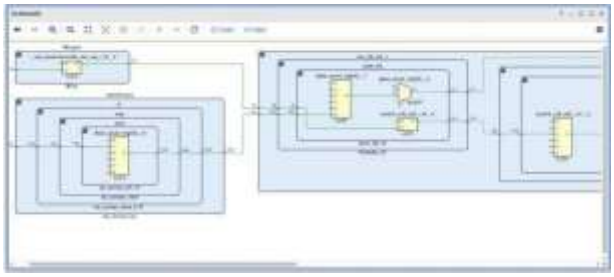

Fig13: Schematic Window

Fig14: Project Summary

8. RESULTS

Fig15: Power of Proposed Block Diagram

Name	Slice LUTs (134600)	Slice Registers (269200)	Bonded IOB (400)	BUFGCTRL (32)
image_processing	53	51	21	2

Fig16: Area of Proposed Block Diagram

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	2	2	1	pixel_out_reg[0]/C	pixel_out[0]	3.521	2.876	0.646

Fig17: Delay of Proposed Block Diagram

Fig18: Matlab Output

8.1: Comparision Table:

Parameter	Existing Method	Optimized Clock gated
Area [LUT's]	52	53
Power[W]	6.102	0.206
Static Power[W]	0.152	0.131
Dynamic Power[W]	5.950	0.075
Delay[ns]	3.521	3.521
Net Delay[ns]	0.646	0.646
Logic Delay[ns]	2.876	2.876

9. ADVANTAGES & APPLICATIONS

9.1. Advantages:

- Improves image brightness and contrast
- Preserves important image details and edges

- Supports real-time processing using FPGA
- Minimizes switching activity in hardware
- Ensures efficient resource utilization
- Offers modular and scalable design
- Suitable for high-speed applications
- Reduces overall hardware complexity

9.2. Applications:

- Surveillance systems for night monitoring
- Autonomous vehicles for better night vision
- Robotics for vision-based operations
- Night vision systems
- Security and defense applications
- Smart city monitoring systems
- Remote sensing and satellite imaging
- Industrial inspection systems

10. CONCLUSION AND FUTURE SCOPE

This work introduces an energy-efficient FPGA-based approach for enhancing low-light images using a Retinex model combined with flip-flop driven clock gating. The proposed system successfully improves visual quality by increasing brightness and contrast while retaining essential image features. Implementation on FPGA enables high-speed processing due to its inherent parallelism. Furthermore, the use of clock gating effectively lowers dynamic power consumption by reducing unnecessary signal transitions. The design achieves a balance between performance and power efficiency, making it suitable for real-time image processing applications.

The system can be further improved by incorporating advanced techniques such as deep learning-based enhancement models to achieve better visual quality. Future developments may include implementation on more advanced FPGA platforms to support higher resolution images and faster processing speeds. Additional optimization methods can be explored to further reduce power consumption and hardware complexity. The proposed approach can also be extended to applications such as intelligent surveillance, autonomous navigation, and smart vision systems.

REFERENCES

1. 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 Transactions on Image Processing*, vol. 6, no. 7, pp. 965–976, 1997.

2. X. Guo, Y. Li, and H. Ling, “LIME: Low-Light Image Enhancement via Illumination Map Estimation,” *IEEE Transactions on Image Processing*, vol. 26, no. 2, pp. 982–993, 2017.

3. S. Mittal, “A Survey of Techniques for Improving Energy Efficiency in Embedded Computing Systems,” *International Journal of Computer Aided Engineering and Technology*, 2014.

4. N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Pearson Education, 2011.

5. Xilinx, “Vivado Design Suite User Guide,” 2020.

6. R. Gonzalez and R. Woods, *Digital Image Processing*, 3rd Edition, Pearson, 2008.

7. J. Cong and Y. Zou, "Energy-Efficient FPGA Design Techniques," *ACM Transactions on Design Automation*, 2012.

8. IEEE, "IEEE Standard for Verilog Hardware Description Language," IEEE Std 1364, 2005.

9. K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits," *Proceedings of the IEEE*, 2003.

10. W. Wang et al., "FPGA-Based Low-Light Image Enhancement Using Retinex Algorithm," *IEEE Access*, 2020.