APPROXIMATE VEDIC MULTIPLY-ACCUMULATE (MAC) UNIT FOR EDGE DETECTION

A PROJECT REPORT

Submitted by

SURYA PRATAP SARANGI – 220301130011

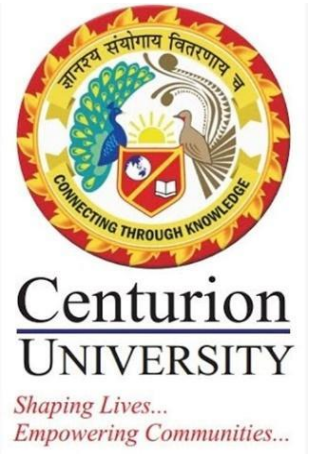
In partial fulfilment for the award of the degree

Of

BACHLOR OF TECHNOLOGY

In

ELECTRONIC & COMMUNICATION ENGINERING



SCHOOL OF ENGINEERING AND TECHNOLOGY

BHUBANESWAR CAMPUS

CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT

ODISHA

NOVEMBER 2024

DEPARTMENT OF ELECTRONIC AND COMMUNICATION ENGINEERING
SCHOOL OF ENGINEERING AND TECHNOLOGY
BHUBANESWAR CAMPUS

BONAFIDE CERTIFICATE

Certified that this project report "**APPROXIMATE VEDIC MULTIPLY-ACCUMULATE(MAC) UNIT FOR EDGE DETECTION"** is the bonafide work of "**SURYA PRATAP SARANGI"** who carried out the project work under my supervision. This is to further certify to the best of my knowledge, that this project has not been carried out earlier in this institute and the university.

SIGNATURE
Dr. Chandra Sekhar Dash
Professor of Electronic and Communication Engineering

Certified that the above-mentioned project has been duly carried out as per the norms of the college and statutes of the university.

SIGNATURE
Dr. Harish Chandra Mohanta
HEAD OF THE DEPARTMENT
Professor of Electronic and Communication Engineering

DEPARTMENT SEAL

DECLARATION

I hereby declare that the project entitled "**Approximate Vedic Multiply-accumulate (MAC) unit for Edge detection**" submitted for the "Major Project" of 5th semester B. Tech in Electronic and Communication Engineering is my original work and the project has not formed the basis for the award of any Degree / Diploma or any other similar titles in any other University / Institute.

Name of the Student: SURYA PRATAP SARANGI

Signature of the Student:

Registration No: 220301130011

Place: Jatani,

Odisha

Date:

ACKNOWLEDGEMENTS

I wish to express my profound and sincere gratitude to **Dr. Chandra Sekhar Dash,** Department Electronic Communication and Engineering, Bhubaneswar Campus, who guided me into the intricacies of this project nonchalantly with matchless magnanimity.

I thank **Dr. Harish Chandra Mohanta** Head of the Dept. of Department of Electronic and Communication Engineering, SoET, Bhubaneswar Campus and **Dr. Sujata Chakravarty,** Dean, School of Engineering and Technology, Bhubaneswar Campus for extending their support during Course of this investigation.

I would like to express my respect and thank all faculty member and Staff of the Department of Electronic and Communication Engineering, CUTM, Jatani for their generous help in various ways for the completion of this project.

Name of the Student: Surya Pratap Sarangi

Signature of the Student:

Registration No: 220301130011

Place: Jatani, Odisha

Date:

ABSTRACT

This project presents the design and implementation of a Vedic Multiply-Accumulate (MAC) unit for edge detection, optimized for FPGA deployment using Xilinx ISE. The Vedic multiplier leverages ancient Indian mathematics for efficient arithmetic operations, offering advantages in speed and resource utilization compared to conventional multipliers. The MAC unit is implemented in Verilog/VHDL and synthesized on Xilinx platforms, with applications in image processing tasks such as edge detection.

Edge detection is achieved through a Sobel filter in Python, using the Vedic multiplier to perform pixel-wise gradient computations. Additionally, the project explores the design of an approximate MAC unit, focusing on reducing power consumption and area while maintaining acceptable accuracy levels. The impact of approximation is assessed by comparing the edge detection results from the Vedic MAC unit and the approximate MAC unit.

Performance metrics, including Mean Square Error (MSE), Peak Signal-to-Noise Ratio (PSNR), Mean Absolute Error (MAE), Average Distance (AD), and Structural Similarity Index Measure (SSIM), are calculated to evaluate the image quality and computational efficiency. Results demonstrate that the approximate MAC unit significantly reduces power and area overheads with minimal degradation in edge detection performance, making it suitable for resource-constrained embedded systems.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	CERTIFICATE	**i**
	DECLARATION	**ii**
	ACKNOWLEDGEMENT	**iii**
	ABSTRACT	**iv**

CHAPTER –1	INTRODUCTION	**PAGE NO.**
	1.1. Surveillance	1
	1.2. Objective	1-2
CHAPTER – 2	BACKGROUND	3-5
	2.1. Literature Survey	3
	2.2. Research Gap	4
	2.3. Ethical and Privacy Considerations.	5
CHAPTER -3	METHODOLOGY	6-29
	3.1. Tool flow	6
	3.1.1. key points of Process	6
	3.2. Tool & Library used	6
	3.2.1. Jypyter Notebook IDE	7
	3.2.2. Xilinx ISE Webpack	8
	3.2.3. Cadence EDA	9
	3.3. Problem Definition and Requirement Analysis (Implementation in Xilinx)	9

	3.3.1 Standard Vedic Multiplier Design:	10-11
	3.3.2 Approximate Vedic Multiplier	
	3.3.2. MAC (Multiplier-Accumulator) Design:	10-11
	3.3.3. Edge Detection Filter Using Python	11-12
	3.3.3.1. Edge Detection with Various Filters	11-12
	3.3.3.2. Different Metrices Values of the Image	12-16
	3.3.3.3. Input Image Data.	16
	3.4 RTL to GDSII Flow	16-17
	3.4.1. Flow in Cadence Tools	17-18
	3.4.1.1. Design & Verification of RTL	18-19
	3.4.1.2. Synthesis of RTL	20-21
	3.4.1.3. Physical Design & Optimization	21-29
CHAPTER – 4	RESULTS & DISCUSSION	30-41
	4.1. Implementation in Xilinx ISE Design	30
	4.2. MAC Benchmark Validation for Edge Detction	31-35
	4.3. Value Difference Table of Two Test Image	35
	4.4 Synthesis Output Circuit (Vedic Multiplier)	36-38
	4.5 Final Chip Design in RTL to GDSII process	39
	4.6 Approximate vs. Exact Vedic Multipliers: Performance Analysis	40
	4.7 Discussion	40-41
CHAPTER – 5	CONCLUSION & FUTURE SCOPE	42-44
	5.1. Conclusion	42-43
	5.2. Inference	43
	5.3 Future Scope	44
REFERENCES		45-46
APPENDIX – A		47-53
APPENDIX – B		54-63

APPENDIX – C 64-84

APPENDIX – D 85-103

CHAPTER-1
INTRODUCTION

1.1. Surveillance

Surveillance systems play a critical role in ensuring security, monitoring environments, and analyzing activities in real-time. These systems often rely heavily on advanced image processing techniques to detect, track, and analyze objects, people, or events in dynamic scenarios. Given the massive amount of data generated by surveillance cameras, efficient processing becomes vital for accurate and timely results.

Our project, **Vedic MAC Unit Design for Edge Detection**, aims to address computational challenges in surveillance systems by introducing an efficient Multiply-Accumulate (MAC) unit. The Vedic multiplier, known for its speed and resource efficiency, is integrated into edge detection processes using a Sobel filter. Additionally, an approximate MAC unit is developed to enhance energy efficiency and reduce hardware complexity, making it suitable for various image processing tasks in real-time surveillance.

Depending on the application scenario, surveillance systems may prioritize either precision (e.g., facial recognition) or speed (e.g., motion detection). The proposed design allows for a balance between computational accuracy and resource utilization, offering flexibility tailored to specific applications.

By utilizing the Vedic multiplier and approximate computing techniques, our design minimizes the circuit area and complexity compared to traditional multipliers. This leads to cost savings and compact form factors, enabling its deployment in small-scale surveillance devices such as drones, portable cameras, and IoT sensors. With the growing adoption of high-definition cameras, our design is well-suited for scalable applications, supporting high-resolution data processing with improved power and area efficiency.

1.2. Objective

The primary objective of this project is to design and implement a Vedic MAC unit optimized for energy-efficient, high-performance image processing, specifically for edge detection in surveillance applications. The project also explores the development of an approximate MAC unit to further enhance computational efficiency and reduce power consumption. The key goals include:

1. **Enhancing Computational Efficiency**:
   Develop a Vedic multiplier-based MAC unit that accelerates image processing tasks, such as edge detection, by reducing computational complexity.

2. **Reducing Power Consumption**:
   Implement approximation techniques to minimize switching activity, lowering power consumption, and making the design suitable for energy-constrained applications like IoT devices, portable systems, and real-time surveillance.

3. **Balancing Accuracy and Efficiency**:
   Provide a flexible design that allows trade-offs between accuracy and performance, ensuring acceptable output quality without significantly compromising results.

4. **Optimizing Hardware Utilization:**
   Implement a resource-efficient design that reduces area and circuit complexity, enabling integration into compact, low-cost devices.

5. **Integration With Real-World Applications:**
   - Demonstrate the utility of the proposed MAC unit in practical scenarios:
   - Edge detection in surveillance systems using Sobel filters.
   - Image sharpening and noise reduction for enhanced visual clarity.
   - Convolution layers in deep learning models for neural networks.
   - Power-sensitive systems like drones, mobile devices, and edge AI hardware.

6. **Achieving Design-to-Physical Flow:**
   Complete the RTL-to-GDSII flow to realize the MAC unit in silicon, ensuring compliance with industry standards for timing, area, and power.

7. **Validation and Benchmarking**:
   Validate the design through simulation, synthesis, and benchmarking against traditional multipliers. Analyze performance metrics such as power, area, delay, and image quality metrics like PSNR, SSIM, MSE, MAE, and AD

.

.

CHAPTER-2
BACKGROUND

2.1. Literature Survey

The design of a Compressor-Based Adaptive Approximate Multiplier (CAAM) for image processing and neural network applications is built on several key advancements in approximate computing and hardware design:

2.1.1. Approximate Computing Paradigm:

Approximate computing trades off accuracy for energy efficiency, making it ideal for error-tolerant tasks like image processing and neural networks. Studies show that slight inaccuracies can significantly reduce power and resource consumption while maintaining acceptable performance.

2.1.2. Compressor-Based Multiplier Architectures:

Compressors, especially 4:2 configurations, reduce partial product complexity, enabling faster multiplication with lower area and power consumption. They are widely adopted in FPGA-based systems for performance optimization.

2.1.3. Adaptive Multiplier Techniques:

Adaptive multipliers dynamically adjust their approximation levels based on input or application requirements, offering a balance between precision and resource efficiency. This flexibility makes them suitable for real-time, resource-constrained environments.

2.1.4. Image Processing Applications:

Image processing algorithms such as Gaussian blur, edge detection, and sharpening rely on multipliers for pixel-level calculations. Approximate multipliers can maintain high PSNR and SSIM values while reducing power, making them effective for real-time image processing in surveillance systems.

2.1.5. Neural Network Applications:

Multipliers are crucial in the convolutional layers of neural networks, where computations are intensive. Using approximate multipliers in edge AI devices significantly reduces power consumption without degrading model performance, making them suitable for portable, low-power AI hardware.

2.1.6. RTL-to-GDSII Design Flow:

The complete RTL-to-GDSII design flow validates hardware designs at the silicon level, ensuring compliance with performance standards for timing, area, and power. Tools like Xilinx Vivado, Synopsys Design Compiler, and Cadence Innovus are essential for optimizing and fabricating designs.

2.2. Research Gap

Despite advancements in deploying AI models, especially for face detection, on FPGAs, several research gaps persist. One major gap lies in the optimization of neural network models for FPGA deployment. While many models achieve high accuracy, they are often computationally intensive and require large memory resources, which limits their efficient deployment on resource-constrained FPGA platforms. Techniques like model quantization and pruning help reduce model size and complexity but introduce compatibility issues with accelerators that only support non-quantized models.

Another significant research gap is in hardware-software co-design for AI-driven FPGA systems. Most tools focus on either hardware or software optimization separately, but there is a need for a unified approach that optimizes both the neural network performance and FPGA resource utilization simultaneously. Without this, it is challenging to fully exploit the power of FPGA architectures in AI applications.

Furthermore, the process of converting trained AI models into FPGA-compatible formats remains inefficient. There is a lack of standardized workflows, often requiring manual intervention and multiple tools, which can lead to errors and delays. Automated processes that ensure smooth transitions from model training to FPGA implementation would streamline this process.

Lastly, scalability and portability are challenges that need to be addressed. As AI models and hardware evolve, ensuring compatibility across different FPGA platforms becomes more difficult. Current solutions are often tailored to specific hardware, making it challenging to reuse designs across platforms or for future iterations, limiting the scalability of designs.

2.3. Ethical and Privacy Considerations

The deployment of face detection systems, particularly in surveillance applications, raises several ethical and privacy concerns. These systems inherently involve the collection and processing of biometric data, which is highly sensitive. Unauthorized access to this data or its misuse could lead to severe privacy violations. It is essential to implement strong **data security measures**, including robust encryption, secure storage, and access controls, to ensure that the collected data is protected from unauthorized access. Additionally, compliance with regulations like the **General Data Protection Regulation (GDPR)** is critical in safeguarding individuals' privacy and preventing misuse.

Another concern is the potential for **bias in face detection systems**. If the training data used to develop these systems is not diverse, the algorithm may not perform equally well across all demographics. This can lead to unfair treatment or discrimination, especially in applications like public monitoring or biometric authentication. Ensuring diversity in the training datasets and conducting thorough testing across various demographic groups are crucial to minimizing bias and ensuring fairness in detection.

The potential for the **misuse of face detection technology** is also a significant ethical issue. While face detection can be used for legitimate purposes such as security and attendance tracking, it can also be misused for mass surveillance, raising concerns about individual freedoms. Establishing clear policies and guidelines to prevent misuse, such as ensuring that systems are used only for specific purposes like biometric authentication, is necessary to strike a balance between security and privacy.

Transparency and accountability in how face detection systems operate and handle data are also important. Users and stakeholders must be informed about the capabilities and limitations of these systems, as well as how their data is processed and stored. Regular audits and mechanisms to address errors or misuse are essential to ensure accountability and maintain public trust.

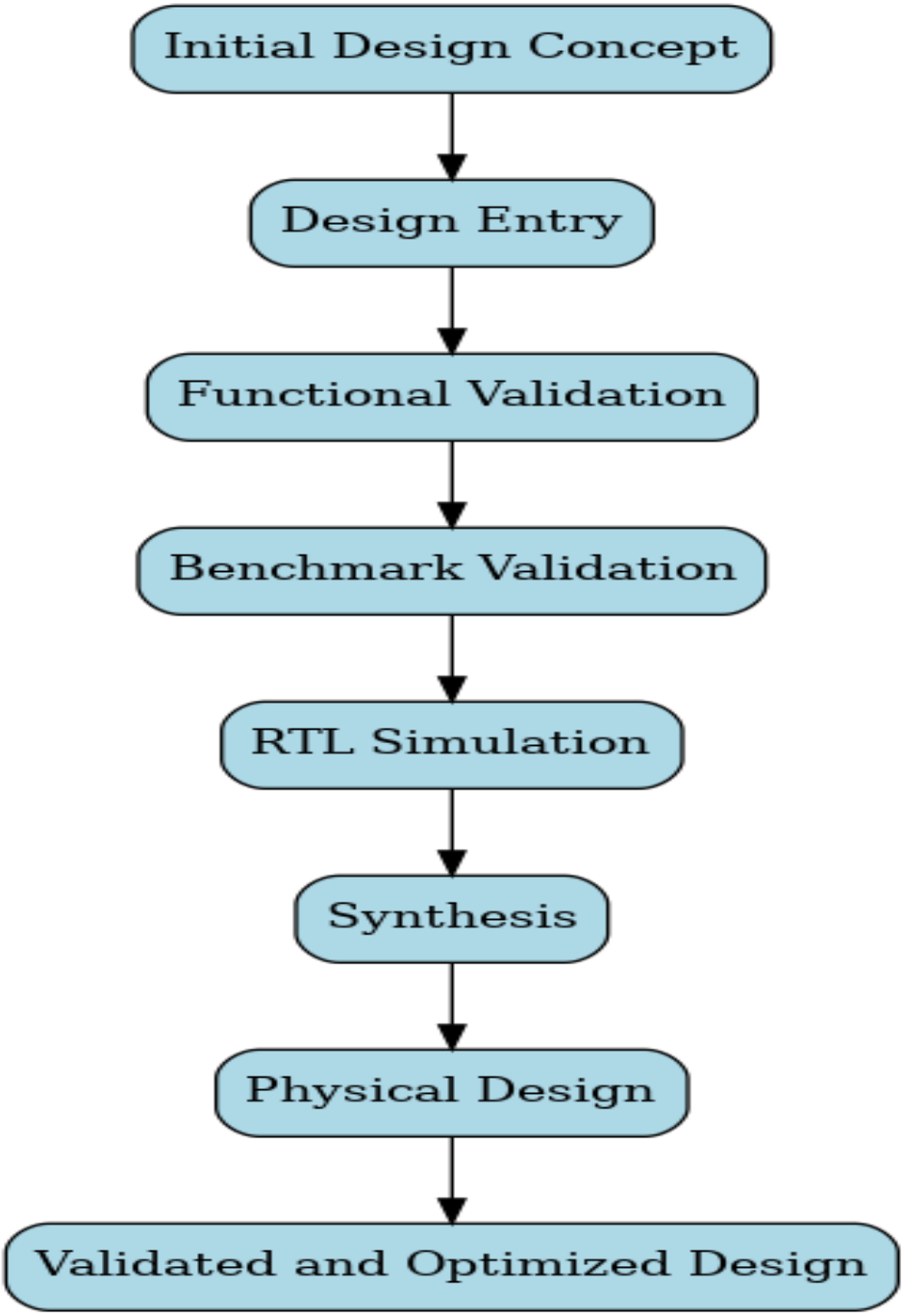
Finally, while FPGAs are generally more energy-efficient than traditional processors, the **environmental impact** of deploying face detection systems at scale should not be overlooked. The energy consumption during the training of AI models and their deployment in large-scale systems can still be substantial, making it important to consider power-efficient design strategies to minimize the environmental footprint of these technologies.

In conclusion, while face detection systems have great potential, addressing ethical and privacy concerns is critical to their responsible deployment. By implementing robust safeguards, ensuring transparency, and adhering to ethical guidelines, we can ensure that these technologies are used in a manner that benefits society while respecting individual privacy.

CHAPTER-3

METHODOLOGY

3.1. Tool Flow



3.1.1. Key Points of Process

- **RTL Design** Develop a hardware description of the design (e.g., CAAM multiplier) in Verilog modules for the multiplier, approximate compressors, and error correction….

- **Benchmark Validation in Python:** Validate the approximation strategy using Python-based image processing benchmarks & Use quality metrics like **PSNR, SSIM, MSE, MAE, AD, MAD, NAE** to compare outputs of accurate and approximate filters.
- **Synthesis:** Convert RTL code to a gate-level netlist optimized for timing, area, and power.
- **Floor planning and Placement:** Define the chip's layout and place cells to optimize utilization and routing.
- **Clock Tree Synthesis (CTS):** Design the clock distribution network.
- **Routing:** Connect the placed cells with wires. Verify that the routed design meets design rules and has no violations (DRC).
- **Static Timing Analysis (STA):** Key Point: Verify timing requirements like setup and hold times. Ensure the design meets timing constraints at different process corners
- **RTL to GDS Flow:** Designs and verifies hardware by using Cadence and Synopsys tools: & Generate the final layout file for fabrication.

3.2. Tool and Libraries Used

3.2.1. Jupyter Notebook code IDE

Jupyter Notebook IDE, **,**Specially for python programming languages

Jupyter Notebook: A Powerful IDE for Python-Based AI Development

Jupyter Notebook is an interactive and user-friendly Integrated Development Environment (IDE) widely used for developing machine learning and neural network models in Python. Its interactive cell-based interface, integration with rich media, and seamless compatibility with Python libraries make it an ideal choice for AI and data science workflows.

Key Features for Neural Network and AI Development:

1. **Interactive Workflow**:
   - Execute code in cells to test and debug incrementally.
   - Display inline visualizations, equations, and text annotations.
2. **Extension Ecosystem**:
   - Jupyter Notebook supports popular libraries like **TensorFlow**, **PyTorch**, and **scikit-learn**, making it highly extensible for AI tasks.

3. **Visualization and Reporting**:
    - Inline integration of plots and charts using **Matplotlib** and **Seaborn** for better insights into model behavior.
4. **Cross-Platform and Collaborative**:
    - Can be run locally or on cloud platforms (e.g., Google Colab).
    - Supports exporting notebooks to multiple formats, including HTML and PDF, for sharing results.

Libraries for Image Processing in Python:

For edge detection and other image processing tasks, the following Python libraries are typically used in **Jupyter Notebook**:

1. **NumPy**:
    - Fundamental for numerical operations and array manipulations.
    - Used for tasks like defining kernels and padding.
2. **OpenCV (cv2)**:
    - A powerful library for image preprocessing, edge detection, and applying filters like sharpening.
3. **Matplotlib**:
    - Essential for visualizing images and intermediate results during the edge-detection workflow.

3.2.2. Xilinx ISE Webpack

Xilinx ISE Webpack is a free FPGA/CPLD development tool supporting design, synthesis, simulation**, and implementation for Xilinx devices**

Xilinx ISE Webpack

The Xilinx ISE WebPACK is a free and feature-rich FPGA and CPLD design tool tailored for small-scale projects, students, and hobbyists. It provides an integrated environment for the complete development cycle, including design entry, synthesis, simulation, and implementation. WebPACK supports HDL languages like Verilog and VHDL, along with schematic-based design entry, making it accessible to users of varying expertise levels.

One of the highlights of Xilinx ISE WebPACK is its compatibility with Xilinx's older FPGA families, such as Spartan and Virtex, and CPLD families. It includes the ISim simulator for functional and timing simulation, enabling users to validate their designs before implementation. The tool also supports device configuration, allowing users to generate bitstreams and program their FPGAs or CPLDs.

Although free, WebPACK has some limitations compared to the full version of ISE. It lacks advanced

features like high-level synthesis and support for the latest FPGA architectures, such as UltraScale and Versal. Nevertheless, its ease of use and robust feature set make it a popular choice for educational purposes and smaller projects.

Xilinx ISE WebPACK remains a reliable tool for designing custom digital circuits, implementing algorithms, and experimenting with FPGA/CPLD-based systems. It is an excellent starting point for those entering the world of programmable logic, offering a cost-effective way to learn and develop hardware designs.

3.2.3. Cadence

Cadence: leading EDA and Intelligent System Design provider delivering hardware, software, and IP for electronic design

Cadence offers a one-stop Electronic Design Automation (EDA) toolkit for RTL-to-GDSII design process, enabling a seamless flow of high-level logic to physical implementation of FPGA and ASIC design. Such tools allow the synthesis, verification, and optimization of hardware designs in an efficient manner, guaranteeing high performance, power efficiency, and area efficiency. Among the tools in the Cadence collection, there is Genus for logic synthesis which translates RTL code into an optimally gate-level netlist, and Innovus which is used for physical design, doing floor planning (layout), placement (and) routing. Cadence's sophisticated timing analysis and optimisation allow designs to satisfy strict timing constraints and cut their power consumption. Furthermore, using tools such as JasperGold and Modus, which offer formal verification and design-for-test features respectively guaranteeing on the one hand the functional correctness and on the other verifying the manufacturability. The pipeline of conjoining the design from RTL to GDSII in Cadence tools facilitates clean design closure, making them well suited for use in high performance/ high accuracy projects such as AI accelerators, where both performance and accuracy are critical.

3.3. Problem Definition and Requirement Analysis (Implementation in Xilinx)

To evaluate the effectiveness of a **Vedic MAC Unit** design for edge detection, it is essential to design both an actual MAC unit (accurate MAC) and an approximate MAC unit using Vedic multiplication techniques. This allows for a fair comparison in terms of key performance metrics such as accuracy, delay, power, and area. Below, the design considerations and implementation strategies for both MAC units are outlined.

The design of the **Vedic MAC unit** involves using Vedic algorithms for multiplication to optimize performance while minimizing resource usage. The approximate version of the MAC unit will incorporate

approximation techniques, such as reduced precision or simplified multiplier circuits, to achieve energy savings while maintaining acceptable output quality for edge detection applications.

3.3.1. Vedic MAC Unit Design for Edge Detection

1. Accurate Region

The Accurate Region of the Vedic MAC Unit focuses on preserving high precision for the most significant bits (MSBs) of the input operands during the multiplication process. In this region, precise Vedic multipliers and adders are employed to ensure that the most crucial components of the multiplication operation are handled with full accuracy. This is essential for maintaining the integrity of edge detection results, as the critical bits play a significant role in determining the final product.

The accurate region ensures that the MSBs are calculated with high fidelity by using full precision for critical multiplications. This guarantees that essential components of the result are computed without error, avoiding large deviations that could affect the edge detection accuracy. While the delay is higher in this region due to the complexity of the accurate Vedic multipliers, this is an acceptable trade-off as the MSBs are vitafor achieving correct results in image processing applications.

2.Approximate Region

The Approximate Region in the Vedic MAC unit processes the middle bits of the input operands using simplified Vedic multipliers, achieving a balance between speed and power efficiency. This region focuses on reducing the computational complexity by approximating the calculation of these middle bits, thereby improving performance without significantly compromising the accuracy of the edge detection results.

This approach leverages simplified Vedic algorithms, such as those involving reduced precision for certain bits, leading to lower power consumption, reduced area, and faster computation. While approximation introduces small errors in the final result, the effect on edge detection is minimal, making it suitable for applications where small tolerable errors can be accepted in favor of efficiency.

3.Error Correction Module

The Error Correction Module in the Vedic MAC Unit compensates for inaccuracies introduced in the approximate region by adjusting the final result to bring it closer to the

expected value. In edge detection, where even small errors can affect the accuracy of detected edges, this module ensures that the deviations due to approximation are minimized.

The error correction involves refining the least significant bits (LSBs) to correct any error caused by the approximations. This module ensures that the accuracy of the edge detection remains within an acceptable threshold, enhancing the overall performance of the Vedic MAC unit. By reducing the error impact, this module allows the Vedic MAC unit to maintain the reliability of its output despite the speed and power advantages in the approximate region.

4.Truncation Region

In the Truncation Region, the least significant bits (LSBs) are discarded, as these bits contribute less significantly to the final result in the context of edge detection. The removal of these bits leads to lower computational complexity, reduced area, and lower power consumption. This simplification is particularly useful in real-time image processing applications, such as edge detection, where the accuracy loss from truncation is negligible.

Truncation is applied to reduce the number of bits being processed, optimizing the overall performance of the Vedic MAC unit. By focusing resources on the most significant parts of the computation, the design achieves a balance between efficiency and accuracy, making it ideal for applications that prioritize real-time processing and power efficiency.

3.3.2. Design & Implementation Using Python

3.3.2.1. Edge Detection using Various Filters

In this study, we explored the application and benchmarking of four distinct filters like Gaussian Blur, Gabor, Median, and Sharpening to evaluate their effectiveness in image processing tasks. The methodology employed for implementing these filters in Python and analysing their performance is described below. To begin, we selected a set of test images to represent various real-world scenarios, such as natural scenes and objects, ensuring a comprehensive evaluation. Each image was converted to grayscale for uniformity, as it simplifies processing and reduces computational overhead. The filters were then applied using standard Python libraries such as OpenCV and NumPy, with specific kernel sizes and parameters tailored to each filter type.

Gaussian Filter: The Gaussian filter is a smoothing filter used to reduce noise and detail in images. It

applies a Gaussian kernel to an image, effectively performing a weighted average of pixel values, where closer pixels have more influence. The degree of smoothing is controlled by the standard deviation (σ) of the Gaussian function, with larger σ values resulting in greater blur. This filter is essential for pre-processing tasks, such as noise removal, and is often used as a precursor to edge detection to improve robustness against noise.

Sobel Filter: The Sobel filter is a popular edge-detection technique that emphasizes areas of rapid intensity change in an image. It operates by convolving the image with two kernels, one detecting horizontal gradients and the other vertical gradients. These gradients are then combined to determine edge magnitude and direction. The Sobel filter is particularly effective for detecting linear features and enhancing the visibility of edges in an image while providing some noise-smoothing due to its averaging component. Its simplicity and efficiency make it widely used in real-time applications.

Convolutional Filter: The convolutional filter, a fundamental component of convolutional neural networks (CNNs), is used for feature extraction by learning spatial hierarchies of patterns in images. Unlike predefined filters, convolutional filters are learned during training and adapt to specific tasks, such as image classification, object detection, or segmentation. These filters operate by sliding over the image, performing a dot product between the filter and local regions of the image to capture edges, textures, and higher-order features. Their adaptability and scalability make them powerful tools for advanced image processing and computer vision tasks.

For benchmarking, the multiplication operation within the convolution process of the Sobel filter was replaced with the custom-designed approximate multiplier logic developed in this project. This modification aimed to evaluate the effectiveness of approximate computing in reducing computational complexity and power consumption while maintaining acceptable output quality. Key performance metrics, such as Peak Signal-to-Noise Ratio (PSNR), were used to assess image quality, while execution time was measured to evaluate computational efficiency. Results were recorded and analyzed across various test images, offering insights into the trade-offs between computational performance and image quality. This methodology underscores the potential of integrating approximate multipliers in edge detection algorithms, paving the way for energy-efficient and hardware-optimized solutions for real-time image processing applications.

3.3.2.2. Different Metrices Values of the Image

MSE (Mean Square Error): The Mean Square Error (MSE) is a widely used metric for assessing the similarity or dissimilarity between two images, making it an essential tool in image processing, particularly in evaluating the performance of filters and other transformation methods. It quantifies the average squared differences between corresponding pixel values of two images: the original image and the processed or filtered image. The MSE measures the cumulative squared error across all pixel values, with a lower MSE indicating that the two images are more similar.

The mathematical equation for MSE can be expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

MSE serves as a cornerstone in image processing, particularly in filter evaluation. For example, when applying a noise removal filter or a smoothing filter, the MSE quantifies how much the filtering operation has altered the image compared to the original. In compression scenarios, MSE assesses the degree of distortion introduced by lossy compression techniques. Moreover, MSE is often used in conjunction with other metrics, such as Peak Signal-to-Noise Ratio (PSNR), to gain a comprehensive understanding of image quality.

Despite its simplicity, MSE has limitations. It does not account for perceptual differences, meaning that two images with the same MSE might appear subjectively different to the human eye. Nonetheless, MSE remains a valuable and computationally efficient metric for initial quantitative comparisons between images in various domains, including medical imaging, remote sensing, and computer vision

MAE (Mean Absolute Error): The Mean Absolute Error (MAE) is a fundamental metric used in image processing to evaluate the similarity between two images, particularly in scenarios involving image filtering, denoising, and transformations. Unlike the Mean Square Error (MSE), which emphasizes larger errors by squaring the differences, MAE provides a more intuitive measure of the average error by considering the absolute differences between corresponding pixel values of two images. This characteristic makes MAE less sensitive to outliers, making it useful for situations where large deviations should not disproportionately influence the assessment.

The mathematical formula for MAE is given as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | x_i - x |$$

In Python, this can be implemented by computing the absolute differences between two image arrays and taking the mean of these values. For instance, using libraries such as NumPy, the MAE can be efficiently calculated with a simple formula. MAE is particularly valuable in assessing the performance of filters in image processing pipelines. For example, after applying a denoising filter, the MAE can indicate how much the pixel values deviate from their original intensities. It is also commonly used in image reconstruction tasks, where it measures the accuracy of the reconstructed image compared to the ground truth.

While MAE provides an easy-to-interpret metric, it does not account for perceptual differences or structural dissimilarities that may be critical in some applications. Despite this, its simplicity, robustness against outliers, and computational efficiency make it a popular choice in scenarios where quantifying overall error magnitude is sufficient.

PSNR (Peak Signal to Noise Ratio): The Peak Signal-to-Noise Ratio (PSNR) is a widely used metric in image processing that quantifies the quality of a processed image compared to an original reference image. It is commonly employed to assess the performance of various image processing techniques, including filtering, compression, and noise reduction. PSNR measures the peak error by comparing the pixel intensity differences, providing a logarithmic representation of the ratio between the maximum possible pixel value and the Mean Square Error (MSE) of the two images. A higher PSNR value indicates better image quality, implying that the processed image is closer to the original.

The mathematical equation for PSNR is expressed as:

$$PSNR = 10 \times \lg\left(\frac{255^2}{MSE}\right)$$

In Python, PSNR can be im PSNR formula. It is commonly used in evaluating filters that modify image intensities, such as denoising or compression filters. For instance, a Gaussian blur applied to an image may alter pixel intensities, and PSNR can quantify the extent of this change.

PSNR is particularly valuable in tasks requiring high visual fidelity, such as medical imaging or video streaming, where preserving image quality is critical. However, it has limitations; it does not account for perceptual quality, as it is purely mathematical. Two images with the same PSNR might appear subjectively different to the human eye. Despite this, PSNR remains a standard measure in image processing due to its computational efficiency and straightforward interpretation.

SSIM (Structural Similarity Index Measure): The Structural Similarity Index Measure (SSIM) is a perceptual metric used in image processing to evaluate the similarity between two images. Unlike pixel-based metrics such as Mean Square Error (MSE) or Peak Signal-to-Noise Ratio (PSNR), SSIM considers structural information, luminance, and contrast to provide a more holistic assessment of image quality. It is particularly useful in applications where preserving perceptual quality is essential, such as image denoising, compression, and enhancement. SSIM outputs a value between −1-1−1 and 111, where 111 indicates identical images, and values closer to zero or negative indicate significant dissimilarity.

The mathematical equation for SSIM is defined as:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

In Python, SSIM can be implemented using libraries like **scikit-image**, which provides a built-in function to compute the SSIM index. SSIM is especially valuable in evaluating the performance of filters that modify image intensities, such as Gaussian blur or wavelet denoising, as it accounts for human visual perception.

While SSIM offers significant advantages over simpler metrics like MSE and PSNR, it has limitations, particularly when dealing with images that vary in global luminance or contrast but retain perceptual similarity. Despite these limitations, SSIM remains a popular and effective measure for assessing image quality in perceptual terms, especially in scenarios where structural integrity is critical.

AD (Average Disatance): The Average Distance metric is used in image processing to quantify the dissimilarity between two images by calculating the mean of the absolute or Euclidean distances between corresponding pixel values. It is particularly useful in evaluating the performance of image processing filters, such as denoising, smoothing, or enhancement filters, by determining how much the filtered image deviates from the original. Unlike metrics like Mean Square Error (MSE), which emphasize large errors, the Average Distance provides a simpler and often more intuitive measure of overall similarity.

The mathematical formulation of the Average Distance depends on the type of distance being considered. The formula is:

$$D_{avg} = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - K(i,j)|$$

In Python, the Average Distance can be computed using libraries like NumPy. For absolute distance, the difference between two images can be computed as the mean of the absolute difference of their pixel values. For Euclidean distance, additional operations to compute channel-wise differences are required.

This metric is widely used in assessing filters, such as comparing the output of a Gaussian blur or median filter against the original image. It provides a straightforward measure of the average deviation and is computationally efficient. However, it does not account for perceptual factors, such as how changes in certain regions of the image might be more noticeable to the human eye. Despite this, the Average Distance remains a valuable tool for evaluating image similarity in many practical applications.

For benchmarking, we replaced the multiplication operation within the convolution processes with our custom-designed Compressor-Based Adaptive Approximate Multiplier (CAAM) logic. This integration aimed to assess the performance of approximate computing in reducing computational complexity and

power consumption while maintaining acceptable output quality. The results for each filter were recorded and analysed, providing insights into the trade-offs between image quality and computational efficiency. This methodology demonstrates the potential of integrating custom approximate multipliers in standard image processing pipelines, paving the way for hardware-efficient solutions in real-time applications

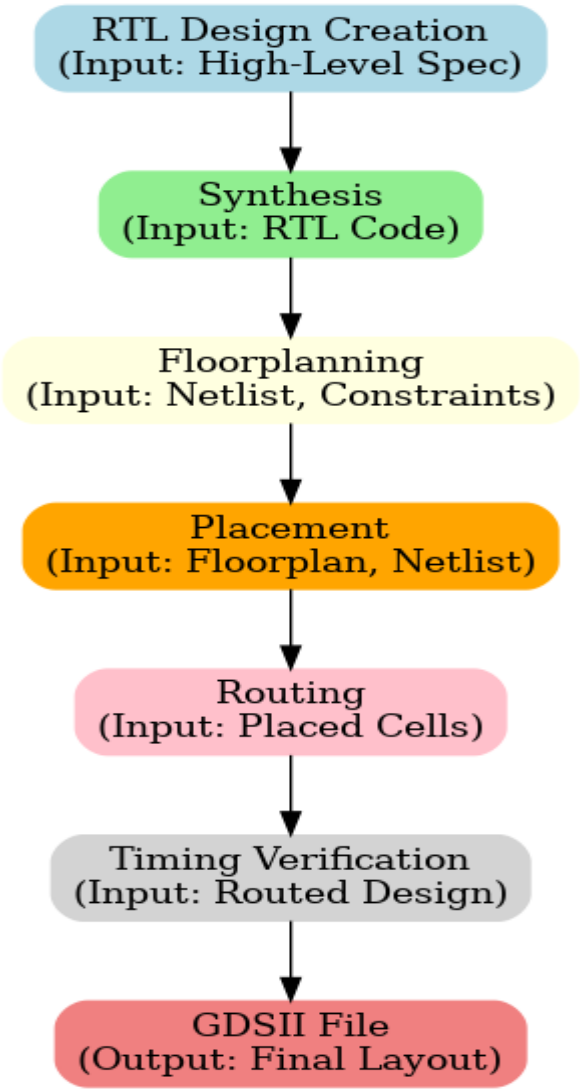
3.3.2.3 Input Image Data.

In the context of image processing, the input image data refers to the raw pixel information of an image that is used as input for various image processing algorithms or filters. The image data is typically represented as a multi-dimensional array (e.g., 2D array for grayscale images). The pixel values are processed and transformed by the applied filters, such as Gaussian, Sobel, Convolutional filter to produce the desired output**.**

3.4. RTL to GDSII Flow

The RTL (Register Transfer Level) to GDS (Graphic Data System) flow is a critical process in digital design, transforming a high-level functional description of a chip into a manufacturable physical layout. This flow is widely used in the semiconductor industry to design and fabricate integrated circuits (ICs), including processors, accelerators, and custom digital designs. The process bridges the gap between the logical representation of a circuit and its physical implementation on silicon. The flow begins with an RTL design, written in hardware description languages like Verilog or VHDL. This design is synthesized into a gate-level netlist, which represents the circuit in terms of basic logic gates like AND, OR, and flip-flops. Synthesis tools also optimize the netlist for area, speed, and power. Next, the netlist undergoes physical design stages, starting with floorplanning, which defines the chip's layout and partitions functional blocks. Placement and routing follow, determining the exact positions of gates and connecting them with interconnects. Timing analysis ensures the design meets its speed requirements, while power

analysis checks energy efficiency. Once verified, the design is transformed into a GDSII file, the industry-standard format for representing IC layouts. This file is sent to foundries for fabrication. The RTL to GDS flow is implemented using Electronic Design Automation (EDA) tools like Cadence and Synopsys. Cadence tools, such as Genus and Innovus, focus on synthesis and physical implementation, while Synopsys tools like Design Compiler and IC Compiler offer robust features for RTL-to-layout conversion.



3.4.1. Flow in Cadence Tools

The RTL to GDSII flow in Cadence begins with importing Verilog design files and a testbench into the Genus tool for synthesis. Start by setting up a project directory and creating a configuration file that specifies the Verilog files, constraints (e.g., timing, clock frequencies), and target technology library. Genus synthesizes the RTL into a gate-level netlist optimized for power, area, and performance. Review synthesis reports to ensure the design meets initial constraints.

Next, the gate-level netlist is imported into Innovus for physical design. Begin with floorplanning, where

you define the chip's die area, partitions, and power grid. Place macros and standard cells to align with the floorplan. Proceed to placement, which determines the physical positions of the cells. Use the constraint files for guiding placement decisions, ensuring signal integrity and timing closure. After placement, routing is performed to connect the cells using metal layers while minimizing congestion and delay.

Post-routing, run static timing analysis (STA) to ensure the design meets all timing constraints. Verify power consumption using power analysis tools. Perform Design Rule Check (DRC) and Layout Versus Schematic (LVS) to confirm the layout adheres to manufacturing rules and matches the gate-level schematic.

Finally, generate the GDSII file, the industry-standard format for fabrication. The GDSII file, along with timing and power reports, is ready for delivery to the foundry for chip manufacturing. Cadence tools ensure a streamlined, efficient RTL to GDS flow.

3.4.1.1. Design & verification of RTL

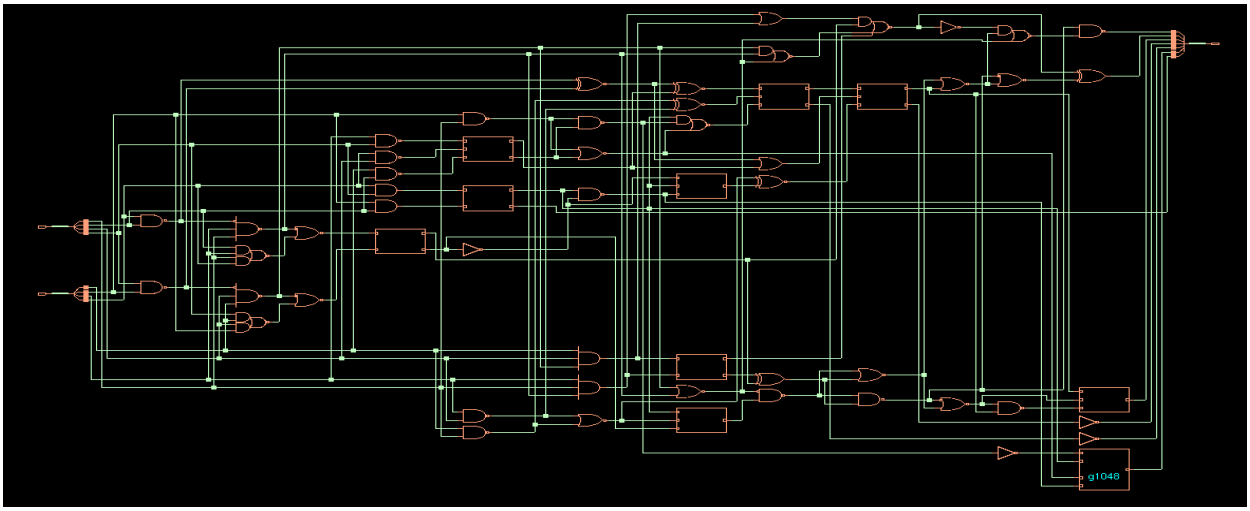
[1] Specification:

The specification and design verification phase is critical to ensuring the RTL (Register Transfer Level) design of the face detection system meets functional and performance requirements. In this project, the RTL design specification includes features such as real-time face detection, integration with an AI accelerator, and efficient use of FPGA resources. The specifications are derived from the project goals, which aim to achieve high accuracy and low power consumption while adhering to the constraints of the Arty A7100T FPGA platform. RTL design begins by translating the system's functional requirements into Verilog code, ensuring modularity and clarity for ease of synthesis and debugging. The design includes modules for processing input data, executing AI inference, and interfacing with external peripherals. A detailed testbench is created to verify the design against functional specifications. This testbench simulates various scenarios, including valid and edge-case inputs, to validate the design's robustness.

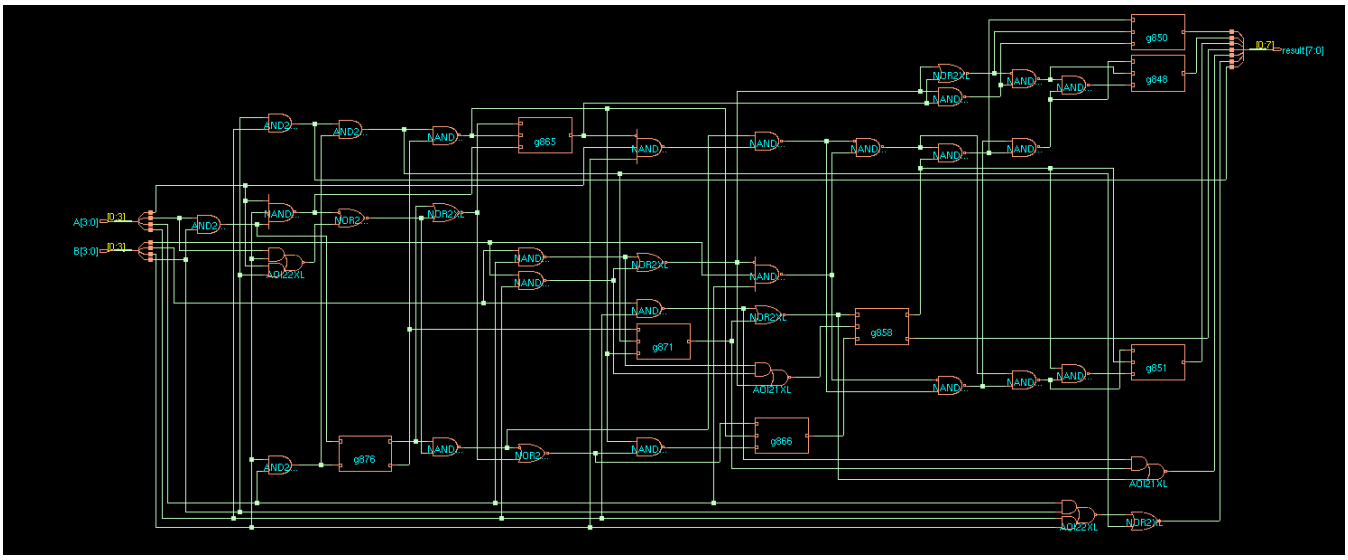
[2] Functional Verification:

Functional verification ensures the RTL design operates as intended, meeting all functional specifications before synthesis and physical implementation. This phase involves testing the design's logic using simulation tools to identify and correct errors early in the development process. For the face detection project, functional verification focuses on verifying data flow, AI inference accuracy, and peripheral communication within the RTL modules. A comprehensive testbench is developed, mimicking real-world scenarios and edge cases to stimulate the RTL design. Inputs such as test images, clock signals, and resets are applied, and the outputs are compared with expected results. Waveform viewers and debugging tools

are used to analyze signal transitions and timing behavior. Coverage analysis ensures that all design paths, including critical and corner cases, are exercised during simulation. Functional verification helps refine the RTL design, ensuring it is robust, bug-free, and ready for synthesis while adhering to project requirements.



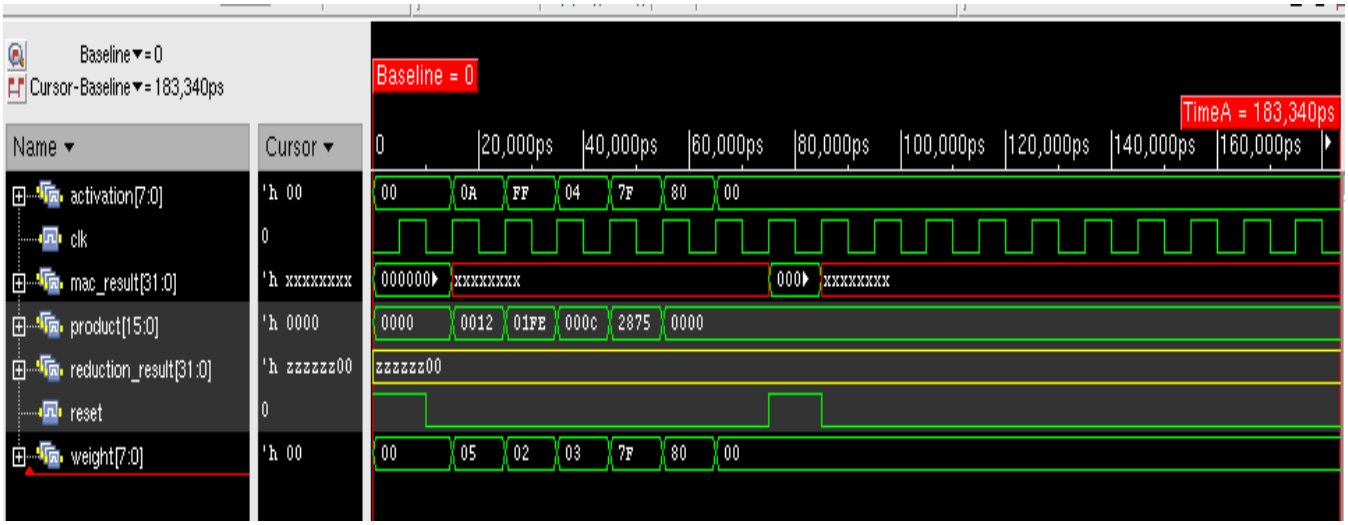
4*4 Exact Vedic Multiplier



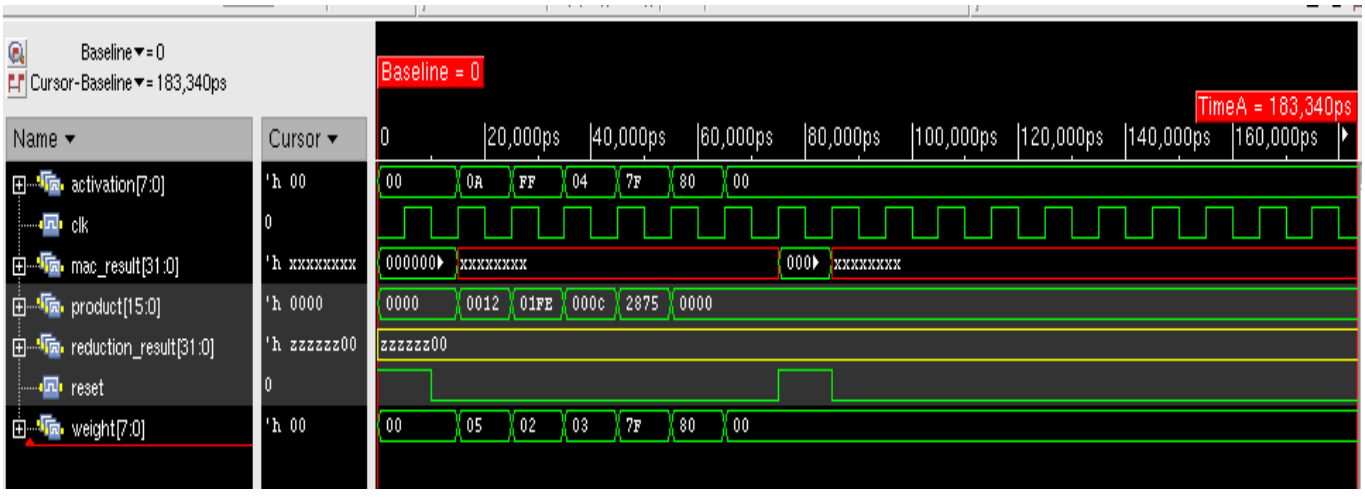
4*4 Exact Approximate Vedic Multiplier

3.4.1.2. Synthesis of RTL

Synthesis is a critical stage in the RTL-to-GDS flow, transforming high-level Verilog or VHDL descriptions into a gate-level netlist. This process bridges the gap between the functional design and its physical implementation on silicon. The primary goal is to convert the abstract RTL logic into a network of standard cells provided by the target technology library. In the face detection project, the synthesis process ensures the design adheres to the constraints set in the specification file, such as timing, power, and area requirements. The synthesis process begins by loading the RTL files, constraint files (e.g., clock frequency, I/O delay, and area), and technology library files into a synthesis tool like Cadence Genus or Synopsys Design Compiler. The tool maps the RTL logic to available standard cells, such as AND, OR, and flip-flops, optimizing for speed and resource utilization. Design constraints guide this mapping



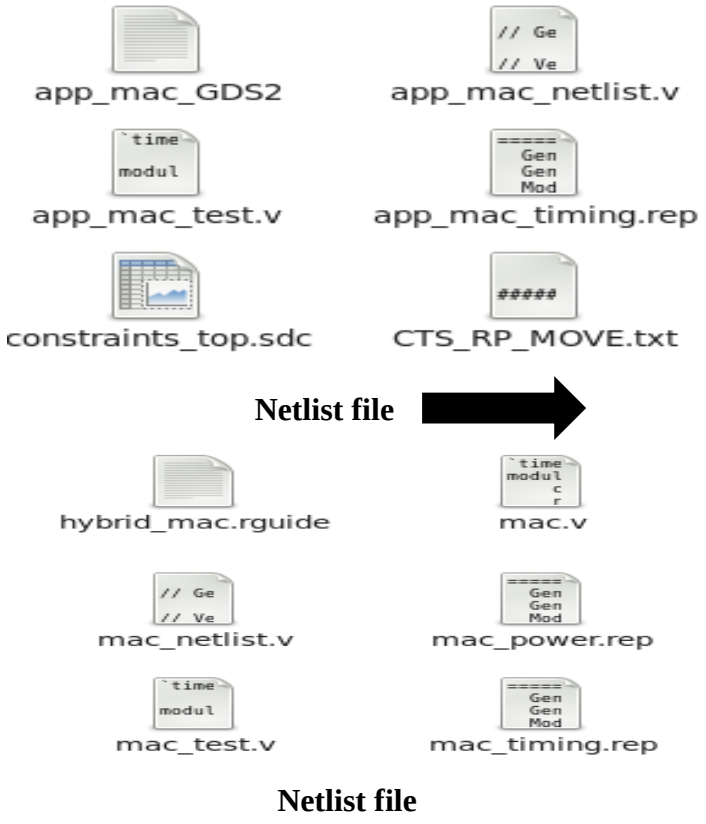
Exact MAC waveform validation

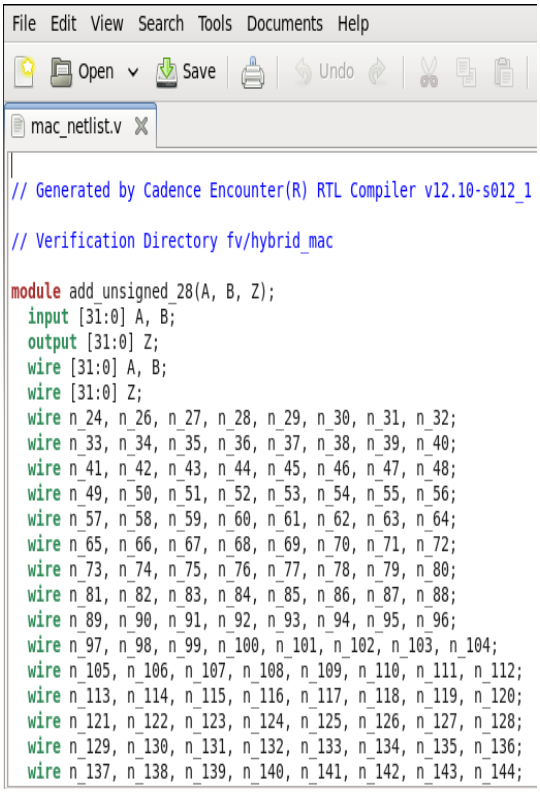


Approximate MAC waveform validation

process to meet the desired performance metrics. The result of the synthesis is a gate-level netlist, a detailed description of the circuit in terms of interconnected standard cells. The netlist serves as the foundation for physical design and contains critical information like cell connectivity and hierarchical structure. Alongside the netlist, synthesis tools generate timing reports, power estimates, and area usage statistics, which are reviewed to ensure the design meets the required specifications.

This process ensures a seamless transition from functional logic to a physically realizable circuit, forming the basis for further steps like placement, routing, and verification.





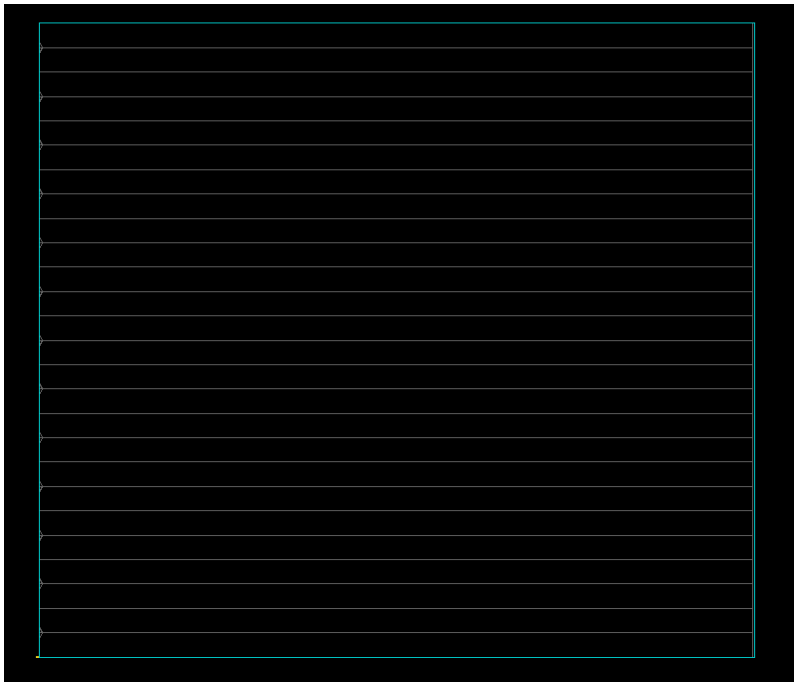
3.4.1.3. Physical Design & Optimization

Physical design is the next critical step after synthesis, transforming the gate-level netlist into a physical layout ready for manufacturing. This process involves mapping the synthesized logic to the physical resources of the FPGA or ASIC while optimizing for timing, power, and area. Physical design ensures the design meets all specifications and constraints set during the synthesis phase. The process begins with **floorplanning**, where the chip's layout is defined by partitioning functional blocks and establishing the locations of macros and memory. A power plan is created at this stage to ensure even power distribution across the chip. Following this, **placement** is performed, arranging standard cells within the defined floorplan to minimize wire length and optimize performance. After placement, the design undergoes **clock tree synthesis (CTS)** to balance clock delays across the chip, ensuring that all sequential elements receive the clock signal simultaneously. This step is crucial for achieving timing closure. Next, **routing** connects the placed cells using metal layers, ensuring that all signals are properly connected without introducing excessive delays or congestion. Post-routing, the design is subjected to optimization to refine timing, power, and area. Tools perform **static timing analysis (STA)** to ensure all paths meet the required timing constraints. Power optimization focuses on reducing dynamic and static power consumption by fine-tuning cell choices and signal routing. Design Rule Check (DRC) and Layout Versus Schematic (LVS) are performed to verify that the layout complies with manufacturing rules and matches the original

schematic. These steps ensure that the design is manufacturable and performs as expected. The final output is a GDSII file, representing the complete physical layout, ready for fabrication or FPGA implementation. Physical design ensures an efficient and robust implementation of the synthesized circuit.

a) Floorplan

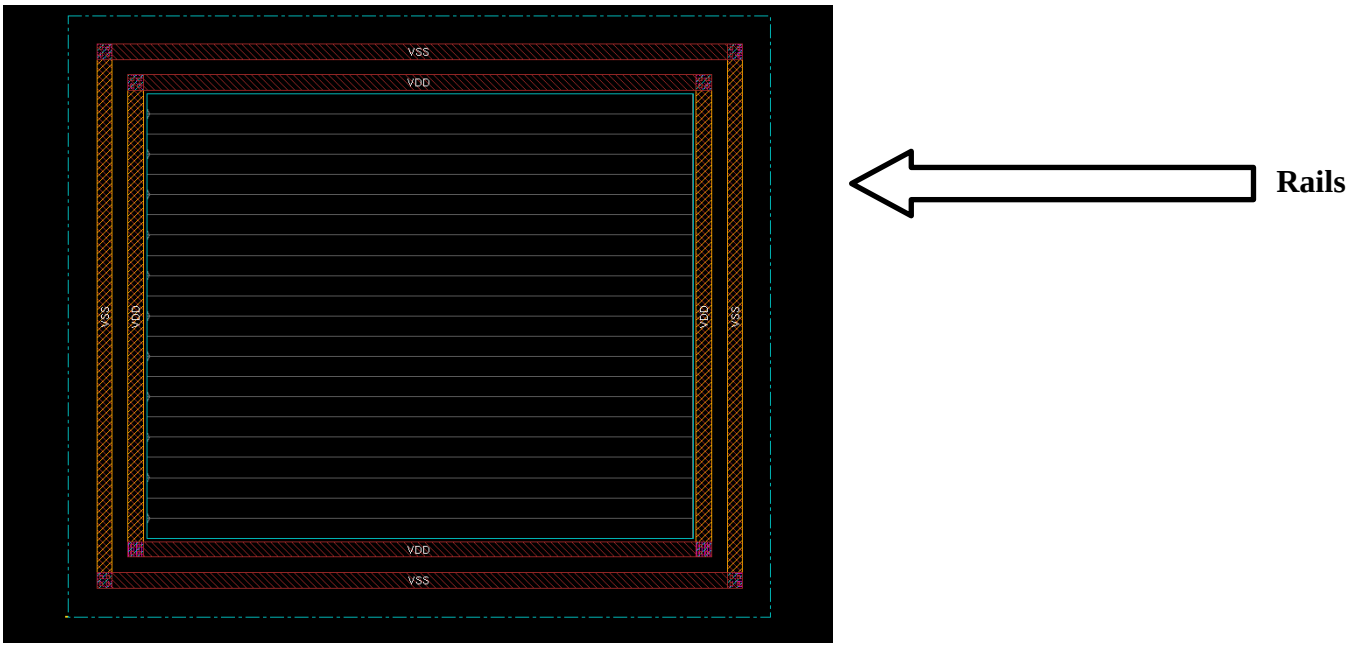
Floor planning is a foundational step in the physical design process, where the layout of a chip is defined by allocating areas for functional blocks and determining their placement within the chip's die. It sets the stage for subsequent steps like placement and routing by optimizing the arrangement of components to achieve the best performance, power, and area utilization. The process begins with partitioning the design into smaller blocks based on functionality, such as logic modules, memory, and input/output (I/O) regions. Each block is assigned a specific area on the chip, considering factors like size, connectivity, and performance requirements. Power planning is also integrated at this stage, with the design of a power grid to deliver stable and efficient power across the chip. Macros, such as large memory arrays or pre-designed blocks, are placed early in the floorplanning process to minimize routing congestion and ensure efficient signal flow. Standard cells, which are smaller logic elements, are arranged around these macros. Placement guidelines, such as aspect ratios and boundary constraints, ensure that the floorplan adheres to design rules and maximizes manufacturability. A well-designed floorplan reduces wire length, minimizes delays, and optimizes chip area, ensuring better performance and lower power consumption. It also anticipates and addresses potential challenges, such as congestion or thermal issues. Tools like Cadence Innovus or Synopsys IC Compiler assist in automating and refining the floorplanning process. By establishing a robust foundation, floorplanning ensures the success of downstream physical design stages.

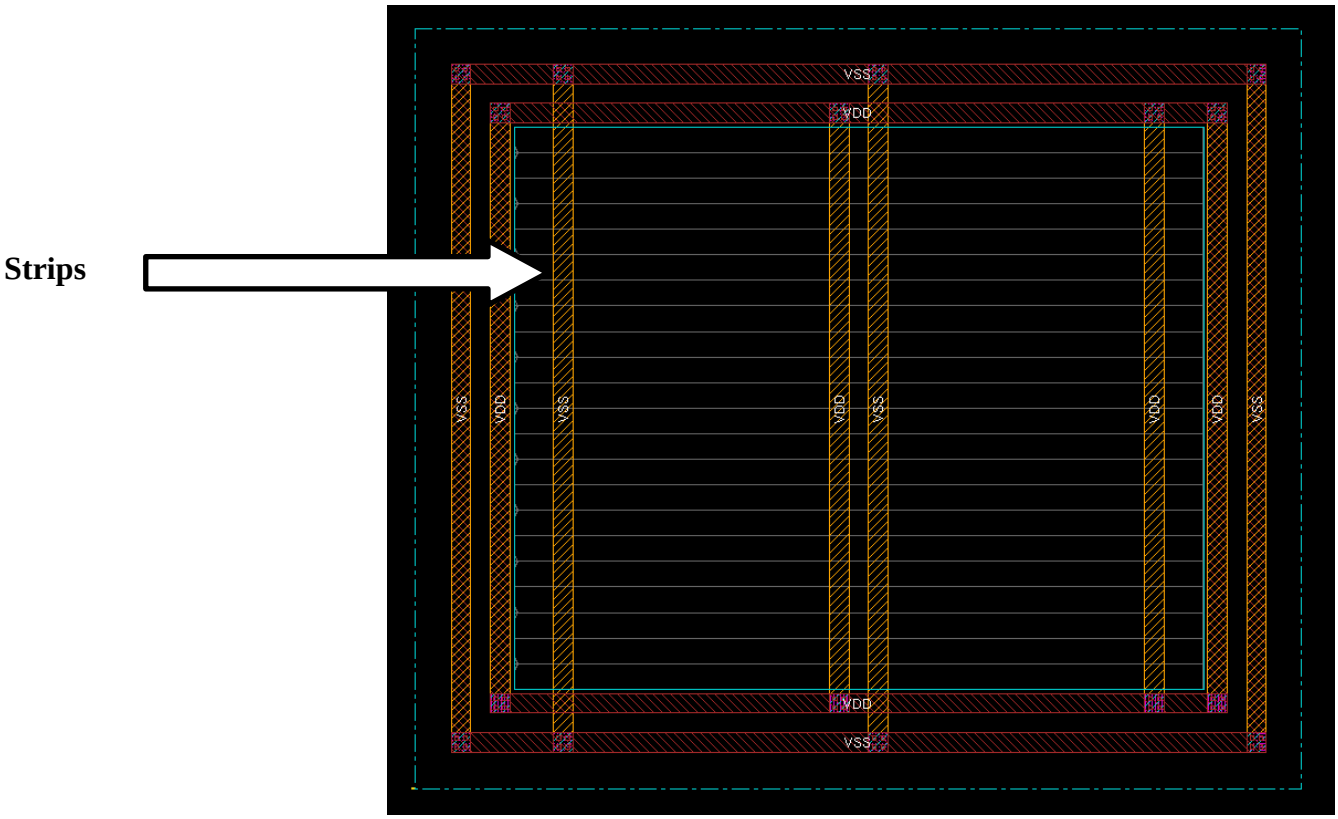


Exact and Approximate MAC floorplan

b) PowerPlan

Power planning is a critical step in physical design following floorplanning. It ensures a stable and efficient delivery of power to all parts of the chip while minimizing power loss and managing thermal distribution. The goal is to design a robust power distribution network (PDN) that meets the power requirements of the chip without causing voltage drops or excessive heat. The process begins with the creation of a power grid. This involves defining power and ground rails on different metal layers of the chip, ensuring even power distribution. The power grid is carefully aligned with the blocks and macros established during floorplanning to provide consistent power to high-demand areas. Decoupling capacitors are strategically placed across the chip to stabilize voltage levels and reduce noise caused by sudden changes in power demand. Power rings are added around large macros, such as memory blocks or AI accelerators, to prevent localized power starvation. Straps and vias are used to connect the power grid across multiple layers, reducing resistance and enhancing current flow. Power domains may be created for blocks operating at different voltages, enabling dynamic voltage scaling for better energy efficiency. Power analysis tools are used to simulate and verify the power distribution network, checking for voltage drops, IR drops, and electromigration. Adjustments are made to optimize the power grid and ensure it meets the design's performance and reliability requirements.

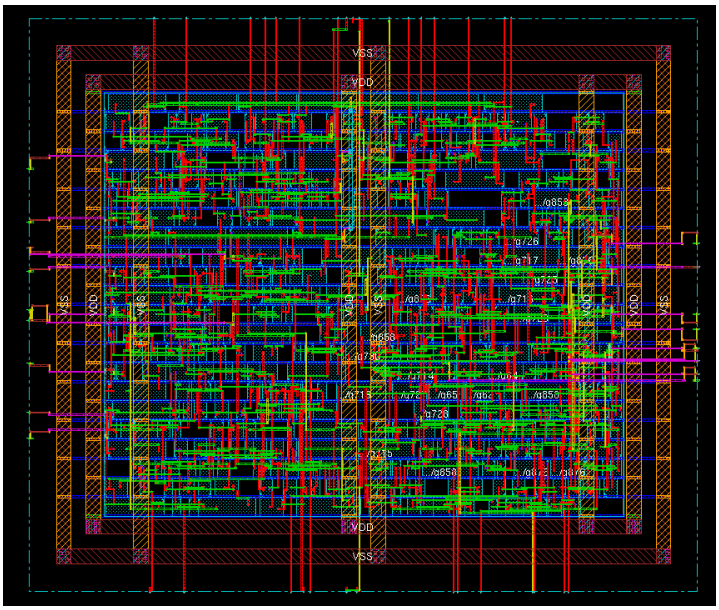




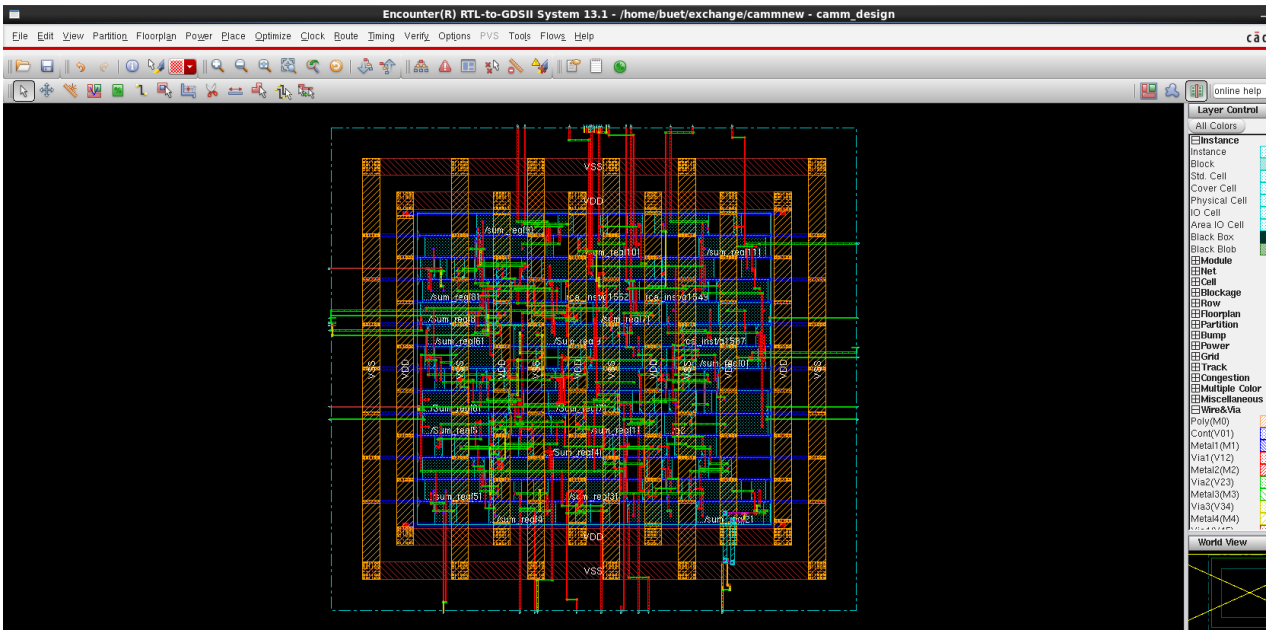
c) Placement of Std. Cells & Macros

Placement is a crucial step in the physical design process, where standard cells and macros are physically positioned on the chip after power planning is completed. This step directly influences the chip's performance, area utilization, and power efficiency by minimizing wire length and optimizing connectivity. Macros, which include large pre-designed blocks such as memory arrays and AI accelerators, are placed first. Their placement is determined by connectivity requirements, power distribution, and the constraints defined during floorplanning. Power rings created during power planning ensure that these macros have a stable power supply. Macros are placed strategically to reduce routing congestion and optimize signal flow between high-demand regions. Once macros are positioned, standard cells—smaller logic elements like gates and flip-flops—are placed around them. Placement tools arrange these cells while adhering to timing, power, and area constraints. Factors such as clock tree synthesis (CTS) requirements, critical paths, and input/output (I/O) pin locations are considered during this stage. Placement algorithms aim to reduce the total wire length, minimize delays, and ensure signal integrity. Post-placement, timing analysis is performed to verify that the design meets all timing constraints. Congestion analysis identifies areas with high interconnect density, which can impact routing efficiency. The placement is refined iteratively to address these issues.

A well-executed placement phase ensures that the design is optimally laid out, facilitating efficient routing and maintaining performance and power targets. It is a pivotal step in the transition from logical design to a manufacturable physical layout.



Approximate MAC Placement of Std. Cells & Macros



Exact MAC placement of Std. Cells & Macros

d) Timing, Power & Area Analysis Report

Timing, power, and area analysis are critical evaluations in physical design, ensuring the chip meets performance, energy efficiency, and size requirements.

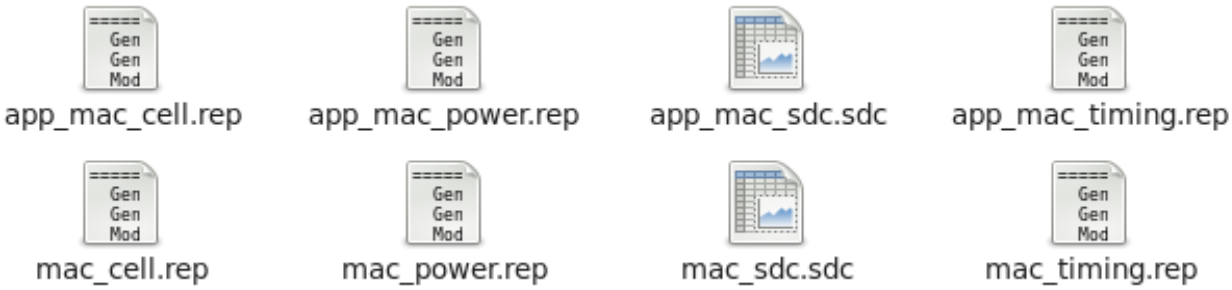
Timing analysis verifies that signal propagation meets design constraints, such as setup and hold times, ensuring the chip operates correctly at the target clock frequency. Tools like static timing analysis (STA) identify critical paths and timing violations, enabling designers to adjust cell placement, routing, or logic to achieve timing closure.

Power analysis focuses on estimating dynamic and static power consumption. Dynamic power depends on switching activity and capacitance, while static power arises from leakage currents. Analysis tools simulate various usage scenarios to optimize power consumption, ensuring efficient operation and thermal management.

Area analysis evaluates the total chip size and resource utilization. Minimizing area reduces manufacturing costs while maintaining performance. Overlapping or inefficient cell placements are identified and corrected.

Together, these analyses guide design optimizations for a robust, high-performance chip.

These are the Report files after analysis of power, Timing & Area.



e) CTS (Clock Tree Synthesis)

Clock Tree Synthesis (CTS) is a critical step in the physical design process, responsible for ensuring that the clock signal reaches all sequential elements (flip-flops) in a chip with minimal skew and delay. The clock is the heartbeat of a digital design, and an efficient clock distribution network is essential for meeting performance and timing requirements. The CTS process begins after cell placement, where clock source and endpoints (sequential elements) are identified. The goal is to create a balanced tree-like structure that distributes the clock signal evenly across the design. Tools analyze the placement of cells and determine an optimal tree structure to minimize skew, which is the difference in arrival times of the clock signal at various endpoints. Buffers and inverters are inserted along the clock paths to balance delays and manage signal integrity. This step also addresses power considerations, as clock trees consume significant dynamic power. Optimizations are performed to reduce clock tree power by adjusting buffer sizes and placements while ensuring timing accuracy. Post-CTS, static timing analysis (STA) is used to verify that the clock tree meets the design's timing requirements. Any violations, such as excessive skew or clock

domain crossing issues, are resolved iteratively by refining the clock tree structure. A robust CTS ensures that the clock distribution network supports the design's functional and performance goals, enabling synchronized operation of all sequential components. This step is crucial for achieving timing closure and preparing the design for routing.

f) DRC (Design Rule Check)

Design Rule Check (DRC) is a critical step in the physical design process that ensures the chip layout adheres to manufacturing constraints set by the foundry. These rules govern the physical dimensions and spacing of features like metal layers, vias, and transistors to ensure the design can be fabricated without defects.

Geometry checks are a significant part of DRC, verifying that the physical dimensions of components comply with the rules. For instance, the minimum width of metal lines, spacing between adjacent layers, and via sizes are analyzed to ensure compliance. This step prevents issues like shorts or opens caused by insufficient spacing or incorrect sizing.

Connection checks focus on verifying the electrical connectivity of the design. This includes ensuring that all signals are correctly routed and that there are no missing connections or unintended overlaps. Proper connections are crucial for the functionality of the design, as errors here can lead to malfunctioning circuits or yield failures.

Tools like Cadence Pegasus or Synopsys IC Validator automate the DRC process, identifying violations and providing detailed reports. Designers use these insights to modify the layout, resolving issues iteratively.

DRC ensures that the chip layout is manufacturable, reliable, and fully functional, playing a vital role in transitioning from design to production.

```
#############################################################
#  Generated by:      Cadence Encounter 13.10-p003_1
#  OS:                Linux i686(Host ID cadence)
#  Generated on:      Wed Dec  4 16:53:17 2024
#  Design:            hybrid_mac
#  Command:           checkPlace hybrid_mac.checkPlace
#############################################################

## No violations found ##

## Summary:
#############################################################
## Number of Placed Instances = 555
## Number of Unplaced Instances = 0
## Placement Density:70.21%(3371/4802)
```

Check Placement

```
#################################################
#  Generated by:      Cadence Encounter 13.10-p
#  OS:                Linux i686(Host ID cadenc
#  Generated on:      Thu Dec  5 14:50:47 2024
#  Design:            hybrid_mac
#  Command:           verifyConnectivity -type
#################################################
Verify Connectivity Report is created on Thu De


Begin Summary
    Found no problems or warnings.
End Summary
```

Check Connections

```
#  OS:                Linux i686(Host ID cadence)
#  Generated on:      Thu Dec  5 14:50:29 2024
#  Design:            hybrid_mac
#  Command:           verifyGeometry
#################################################################

SPACING: Regular Via of Net reduction_result[24] & Regular Wir
Bounds : ( 18.925, 24.285 ) ( 19.070, 24.375 )
Actual:      0.09      Min:      0.14      Type: SameNet  SameNetG

SPACING: Regular Via of Net reduction_result[25] & Regular Wir
Bounds : ( 11.095, 22.025 ) ( 11.240, 22.115 )
Actual:      0.09      Min:      0.14      Type: SameNet  SameNetG

SPACING: Regular Via of Net reduction_result[26] & Regular Via
Bounds : ( 19.505, 11.525 ) ( 19.650, 11.615 )
Actual:      0.09      Min:      0.14      Type: SameNet  SameNetG

SPACING: Regular Via of Net reduction_result[27] & Regular Wir
Bounds : ( 20.375, 13.905 ) ( 20.520, 13.995 )
Actual:      0.09      Min:      0.14      Type: SameNet  SameNetG


Begin Summary ...
  Cells       : 0
  SameNet     : 4
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Total Violations : 4 Viols.
```

Check Geometry

g) GDSII File Generation

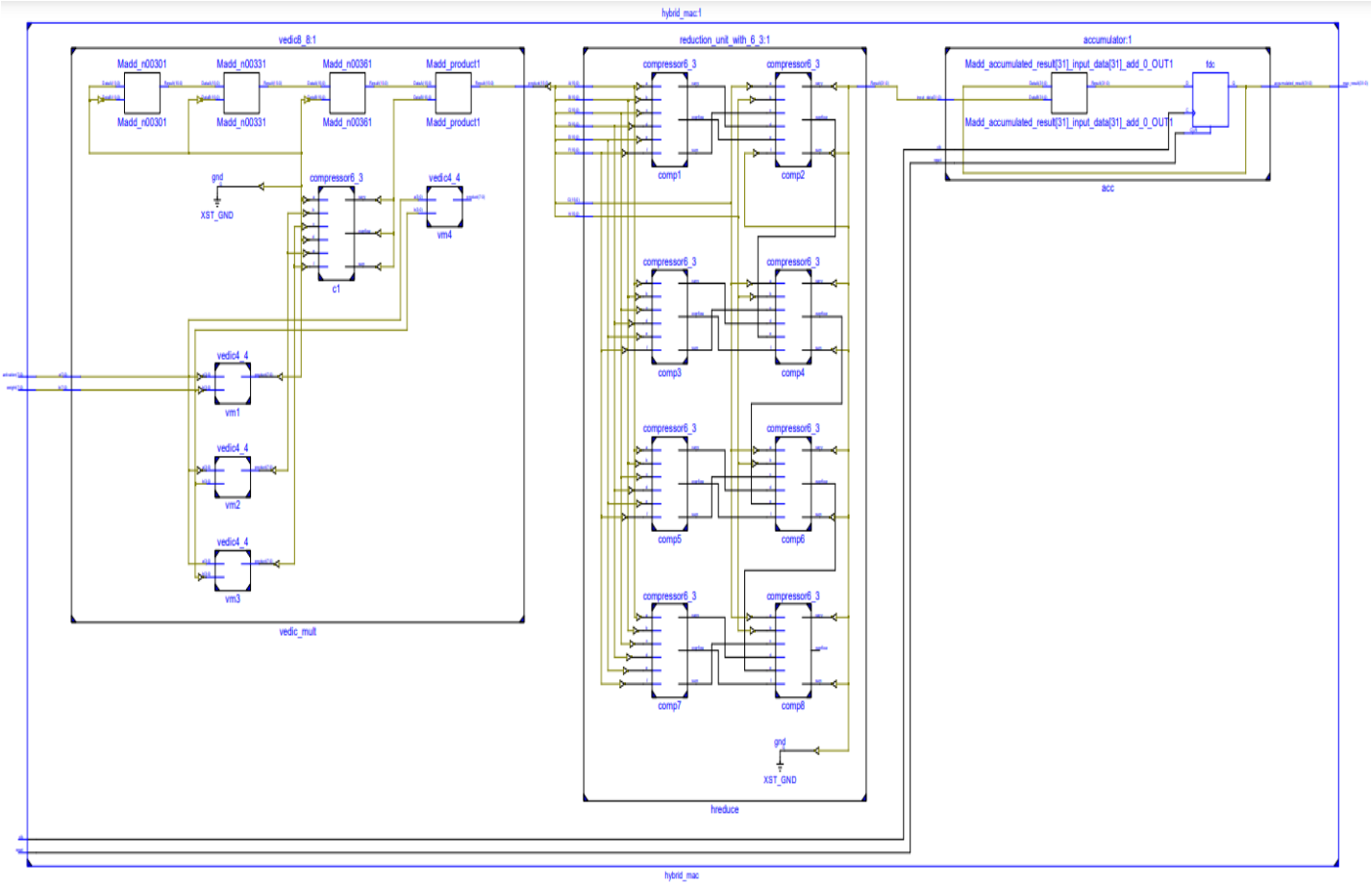
The **GDSII** (Graphic Data System II) format is the final output of the physical design process, representing the complete layout of a chip ready for manufacturing. It is the industry-standard file format used to describe the geometry and layer information of an integrated circuit (IC), providing all the details

necessary for photomask generation in semiconductor fabrication. Once the physical design has passed all critical steps—placement, clock tree synthesis (CTS), routing, and post-routing optimizations—the design is exported into the GDSII format. This format includes precise geometric definitions of all the elements in the chip, including the layout of transistors, metal layers, vias, contacts, and other structures. Each of these components is mapped to specific layers, and GDSII ensures that the design adheres to the constraints of the manufacturing process, such as layer thickness and spacing. The GDSII file is typically hierarchical, with different levels of abstraction representing various components of the chip. Each component (like a logic gate or macro) is broken down into its geometric shapes, such as rectangles or polygons. The file contains all the critical data needed to manufacture the chip, including precise dimensions, locations, and connectivity. Once the GDSII file is generated, it is sent to the foundry, where photomasks are created. These masks are used in photolithography to transfer the design onto the silicon wafer during chip fabrication. The GDSII format plays a crucial role in ensuring the design's manufacturability, yield, and performance, as any errors in the file could lead to manufacturing defects or non-functional chips. In essence, the final GDSII file serves as the blueprint for fabricating the integrated circuit, making it a vital step before moving to the production phase.

CHAPTER-4

RESULT & DISCUSSION

4.1 Implementation in Xilinx ISE design & AMD:



RTL Schematic of MAC Design

4.2. MAC Benchmark Validation for Edge Detection

1. Gaussian Filter

The Gaussian filter showed moderate performance in reducing noise while preserving image quality. For the human image, PSNR was 27.82 and SSIM 0.34, indicating reasonable quality preservation but with noticeable errors (MSE: 107.35, MAE: 144.18). The tiger image had a similar PSNR of 27.64 and SSIM of 0.35, with slightly better results (MSE: 111.87, MAE: 123.22). Both images showed moderate degradation, with the tiger image performing slightly better in terms of PSNR and SSIM.

• For Human Image

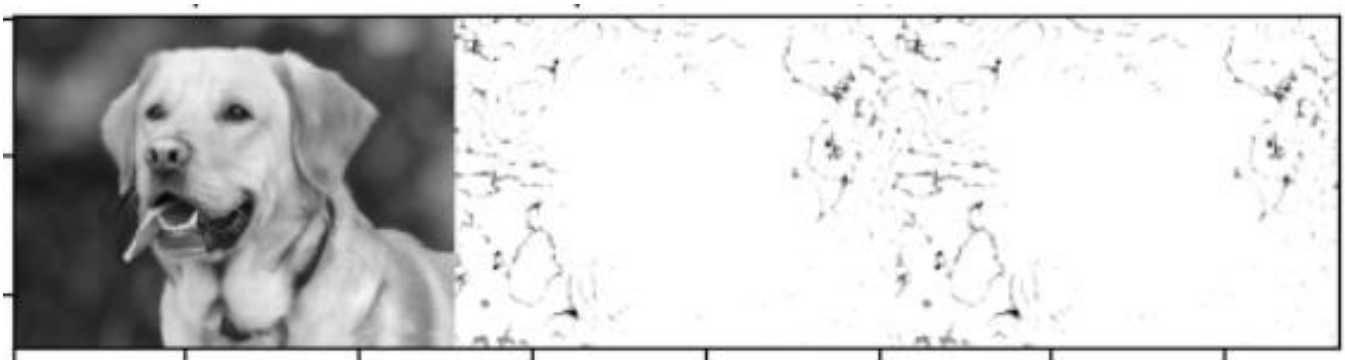
Vedic Gaussian Metrics:

MSE (Mean Squared Error): 107.35034330794508
MAE (Mean Absolute Error): 144.1778548000331
MAD (Median Absolute Deviation): 74.91622272539588
AD (Absolute Deviation): 144.1778548000331
NAE (Normalized Absolute Error): 1.0068621660504222
PSNR (Peak Signal-to-Noise Ratio): 27.822769232134096
SSIM (Structural Similarity Index): 0.3356455493926133

Improved Approximate Vedic Gaussian Metrics:

MSE (Mean Squared Error): 107.49470184884038
MAE (Mean Absolute Error): 144.20114905567345
MAD (Median Absolute Deviation): 74.91622272539588
AD (Absolute Deviation): 144.20114905567345
NAE (Normalized Absolute Error): 1.0070248408573346
PSNR (Peak Signal-to-Noise Ratio): 27.81693301402484
SSIM (Structural Similarity Index): 0.3346163025611694

• For Dog Image



Vedic Gaussian Metrics:
PSNR: 27.64356853218792
SSIM: 0.3497195529085508
MSE: 111.87254233286552
MAE: 123.21874713080888
MAD: 56.8413957410161
AD: 123.21874713080888
NAE: 1.0288695727787778

Improved Approximate Vedic Gaussian Metrics:
PSNR: 27.641965671520865
SSIM: 0.34671052057983587
MSE: 111.91383901050615
MAE: 123.29901234244042
MAD: 56.8413957410161
AD: 123.29901234244042
NAE: 1.0295397827583734

2.Sobel Filter

The Sobel filter, designed to enhance edges, showed moderate performance in both the human and dog images. For the human image, it achieved a PSNR of 27.79 and an SSIM of 0.24, indicating some loss of quality with noticeable errors (MSE: 108.16, MAE: 143.73). The NAE (3473596707.93) and MAD (179) also reflect significant deviations in pixel values. Similarly, for the dog image, the filter produced a PSNR of 27.68 and an SSIM of 0.22, with errors (MSE: 110.98, MAE: 126.25) slightly lower than the human image. The NAE (33610638.56) and MAD (125) suggest a similar performance trend. Overall, while the Sobel filter successfully enhanced edges, it also caused noticeable quality degradation in both images.

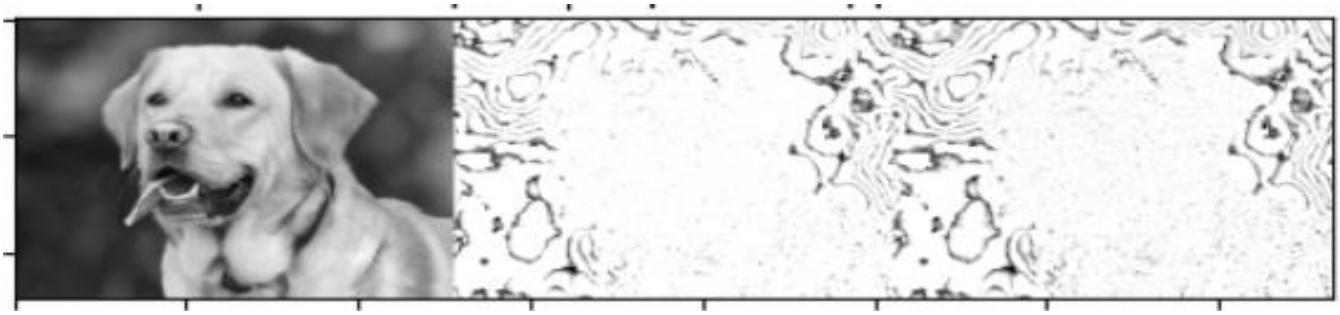
• For Human Image

Metrics for Vedic Multiplier Logic:
PSNR: 27.79032427956361
SSIM: 0.244407018438242
MSE: 108.15533153257444
MAE: 143.73006455781876
AD: 34735967.079347335
MAD: 179.0
NAE: 3473596707.9347334

Metrics for Improved Approximate Vedic Multiplier Logic:
PSNR: 27.788681314560343
SSIM: 0.17462286229749932
MSE: 108.19625515650984

MAE: 142.68281684015815
AD: 43199763.75600424
MAD: 177.0
NAE: 4319976375.600424

- **For Dog Image**



Metrics for Vedic Multiplier Logic:
PSNR: 27.67840512946715
SSIM: 0.22423787324840516
MSE: 110.9787548694272
MAE: 126.24731115803833
AD: 336106.38562641223
MAD: 125.0
NAE: 33610638.562641226

Metrics for Improved Approximate Vedic Multiplier Logic:
PSNR: 27.735411613671044
SSIM: 0.17140962595440798
MSE: 109.53154142784066
MAE: 127.16318319539356
AD: 588923.141367077
MAD: 126.0
NAE: 58892314.1367077

3.Convolutional Filter

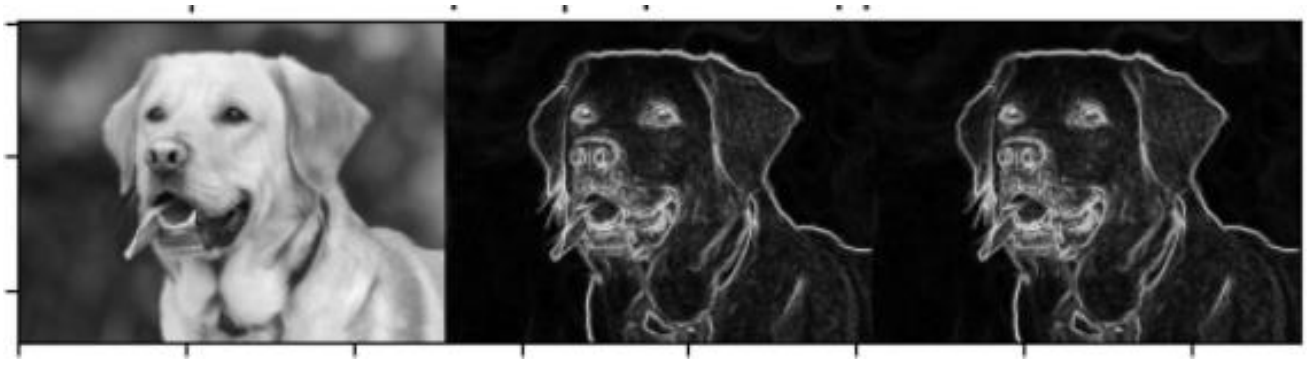
The Convolution filter demonstrated moderate performance in both the human and dog images. For the human image, it achieved a PSNR of 27.82 and an SSIM of 0.34, indicating reasonable quality retention but with noticeable errors (MSE: 107.35, MAE: 144.18). The NAE (1.01) and MAD (74.92) suggest some pixel deviation but moderate overall performance.
For the dog image, the filter showed a PSNR of 27.72 and a significantly lower SSIM of 0.10, indicating a more noticeable loss of structural integrity. The MSE (109.96) and MAE (106.64) were also moderate, with a lower NAE (0.89) and MAD (56.78), showing slightly better performance than the human image. Overall, the convolution filter provided reasonable edge preservation, but it caused some degradation, especially in the dog image.

• For Human Image

Vedic Gaussian Metrics
MSE: 107.35034330794508
MAE: 144.1778548000331
MAD: 74.91622272539588
AD: 144.1778548000331
NAE: 1.0068621660504222
PSNR: 27.822769232134096
SSIM: 0.3356455493926133

Approximate Vedic Gaussian Metrics
MSE: 107.49470184884038
MAE: 144.20114905567345
MAD: 74.91622272539588
AD: 144.20114905567345
NAE: 1.0070248408573346
PSNR: 27.81693301402484
SSIM: 0.3346163025611694

• For Dog Image



Vedic Multiplier Metrics:
MSE: 109.96293619791666
MAE: 106.63596354166667
MAD: 56.78341246636707
AD: 106.63596354166667
NAE: 0.8911360863186022
PSNR: 27.718340331383843
SSIM: 0.09580456494992241

Approximate Vedic Multiplier Metrics:
MSE: 109.96293619791666
MAE: 106.63596354166667
MAD: 56.78341246636707
AD: 106.63596354166667
NAE: 0.8911360863186022
PSNR: 27.718340331383843
SSIM: 0.09580456494992241

4.3. Value Difference Table of Two Test Images for vedic multiplier

Filter	Image	PSNR	SSIM	MSE	MAE	AD	MAD	NAE
Gaussian Filter	Human	27.790	0.244	108.155	143.730	347359967.08	179.0	347359607.93
Gaussian Filter	Dog	27.678	0.224	110.979	126.247	336106.39	125.0	33610638.56
Sobel Filter	Human	27.822	0.336	107.350	144.178	144.178	74.916	1.007
Sobel Filter	Dog	27.644	0.350	111.873	123.219	123.219	56.841	1.029
Convolution Filter	Human	27.823	0.336	107.350	144.178	144.178	74.916	1.007
Convolution Filter	Dog	27.718	0.096	109.963	106.636	106.636	56.783	0.891

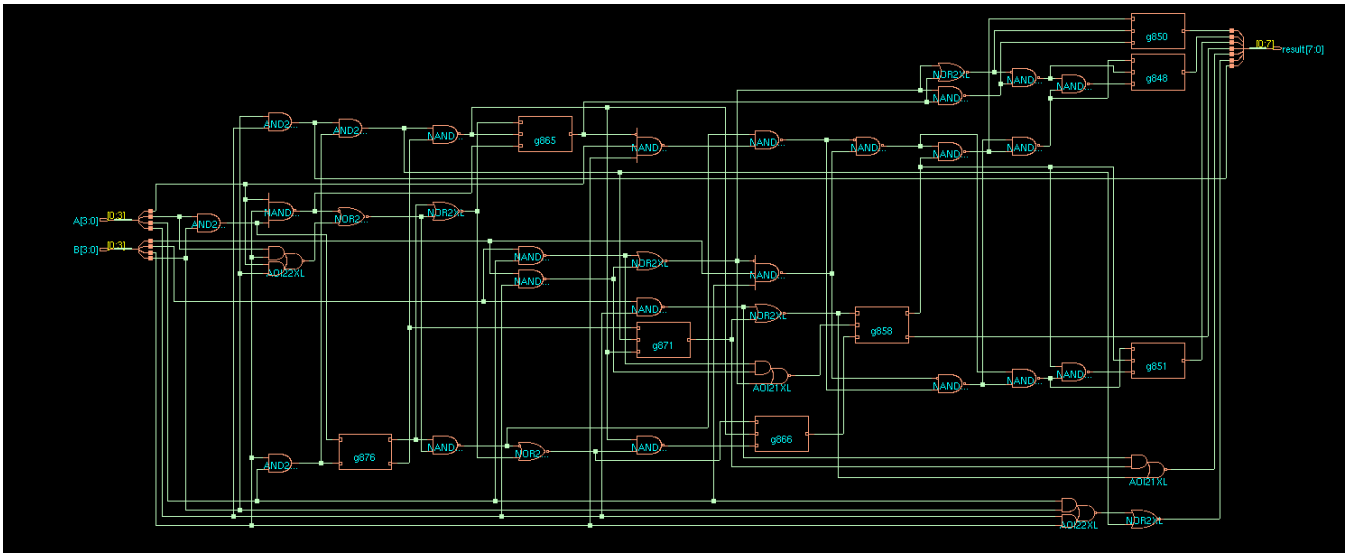
Value Difference Table of Two Test Images for Improved Approximate Vedic Multiplier Logic

Filter	Image	PSNR	SSIM	MSE	MAE	AD	MAD	NAE
Gaussian Filter	Human	27.817	0.335	107.495	144.201	144.201	74.916	1.007
Gaussian Filter	Dog	27.642	0.347	111.914	123.299	123.299	56.841	1.030
Sobel Filter	Human	27.789	0.175	108.196	142.683	43199763.76	177.0	4319976375.60
Sobel Filter	Dog	27.735	0.171	109.532	127.163	588923.14	126.0	58892314.14
Convolution Filter	Human	27.817	0.335	107.495	144.201	144.201	74.916	1.007
Convolution Filter	Dog	27.718	0.096	109.963	106.636	106.636	56.783	0.891

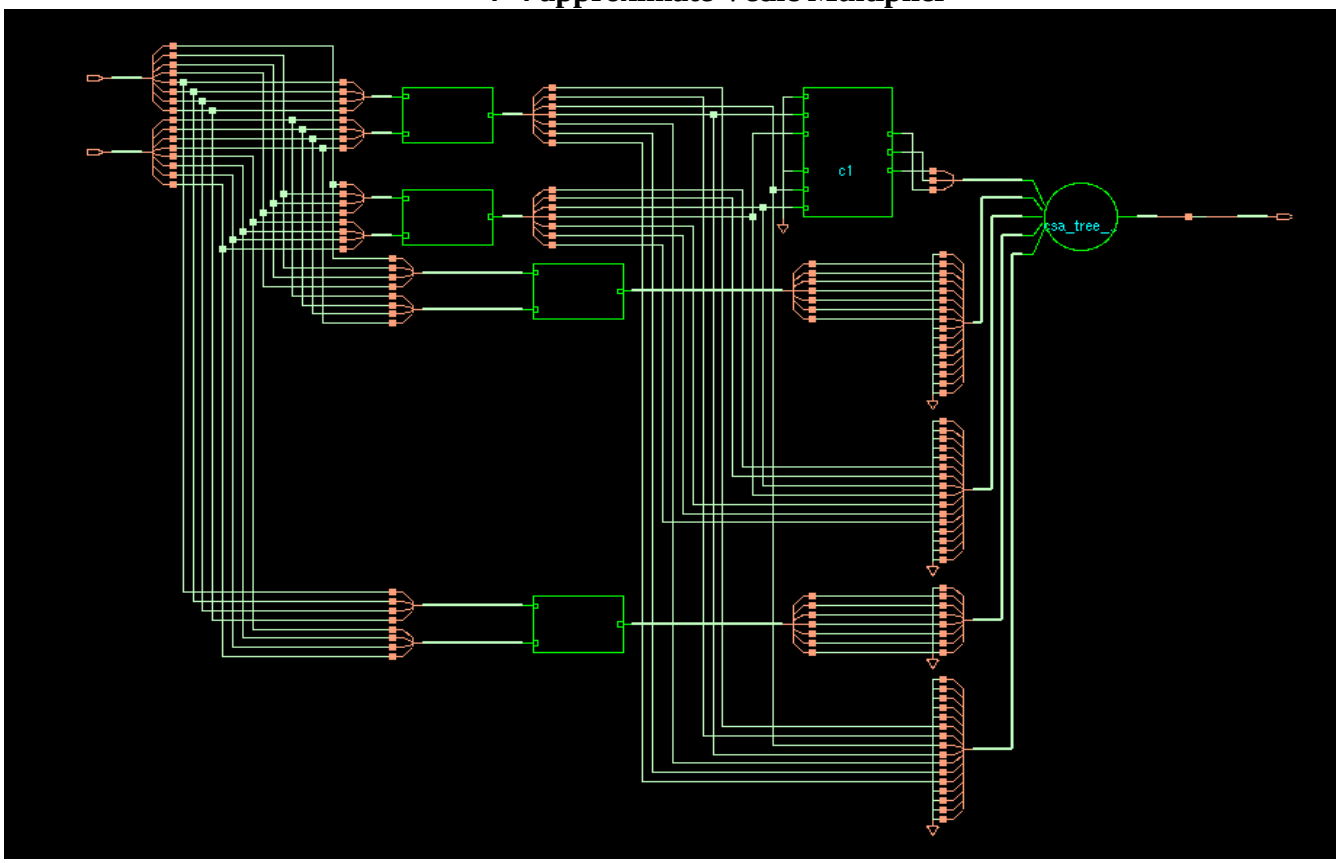
4.4. Synthesis Output Circuit (Exact MAC & Approximate MAC)

The synthesis output circuit generated from Cadence tools represents the optimized logic design transformed into a gate-level netlist. This netlist includes all the standard cells, flip-flops, and logic gates, interconnected according to the design's functionality. During synthesis, the tool converts high-level RTL (Register Transfer Level) code into a set of gates and interconnections while optimizing for area, timing, and power consumption. The output netlist is technology-specific, meaning it aligns with the chosen semiconductor process, ensuring that the design can be fabricated using the correct cells and parameters. The synthesized netlist serves as the foundation for subsequent physical design stages.

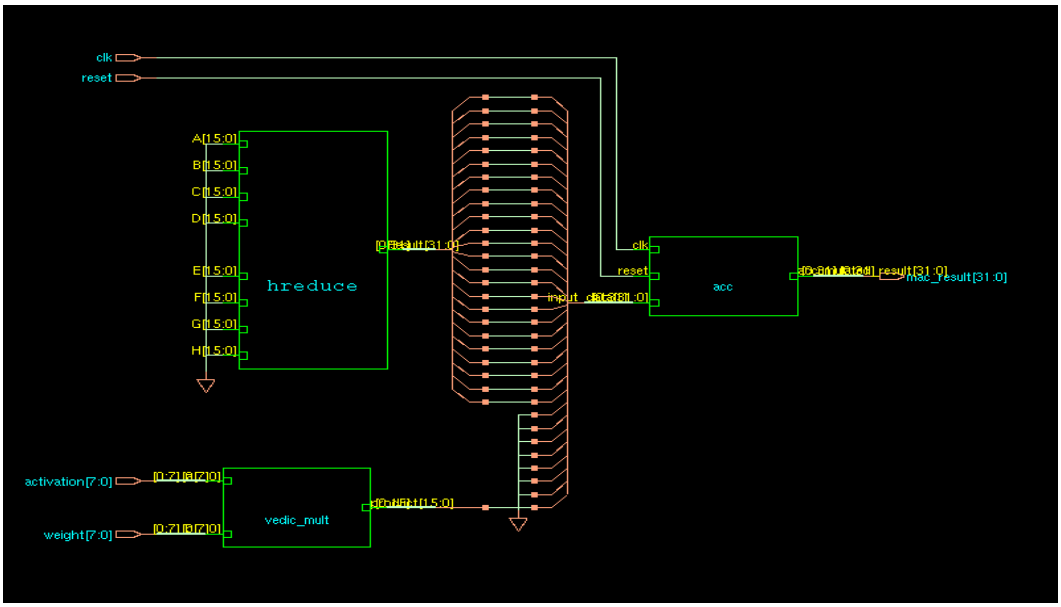
Approximate MAC SYNTHESIS: -



4*4 approximate Vedic Multiplier

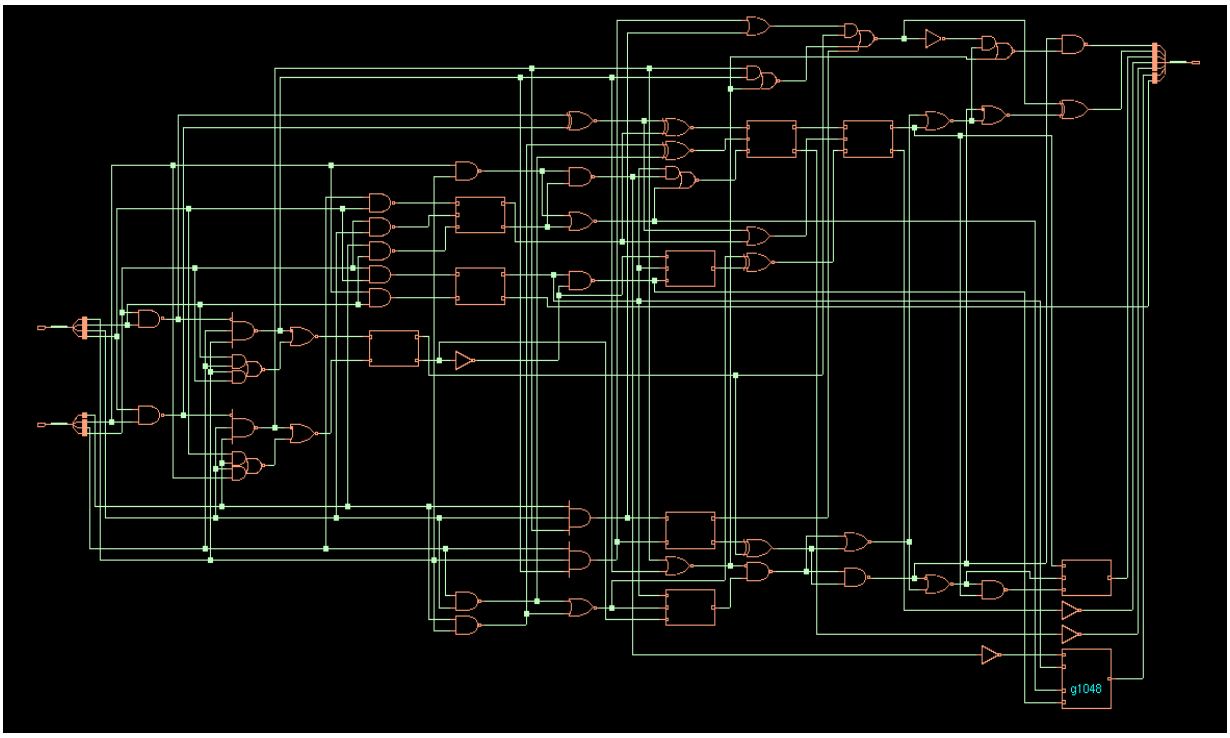


8*8 Approximate Vedic multiplier

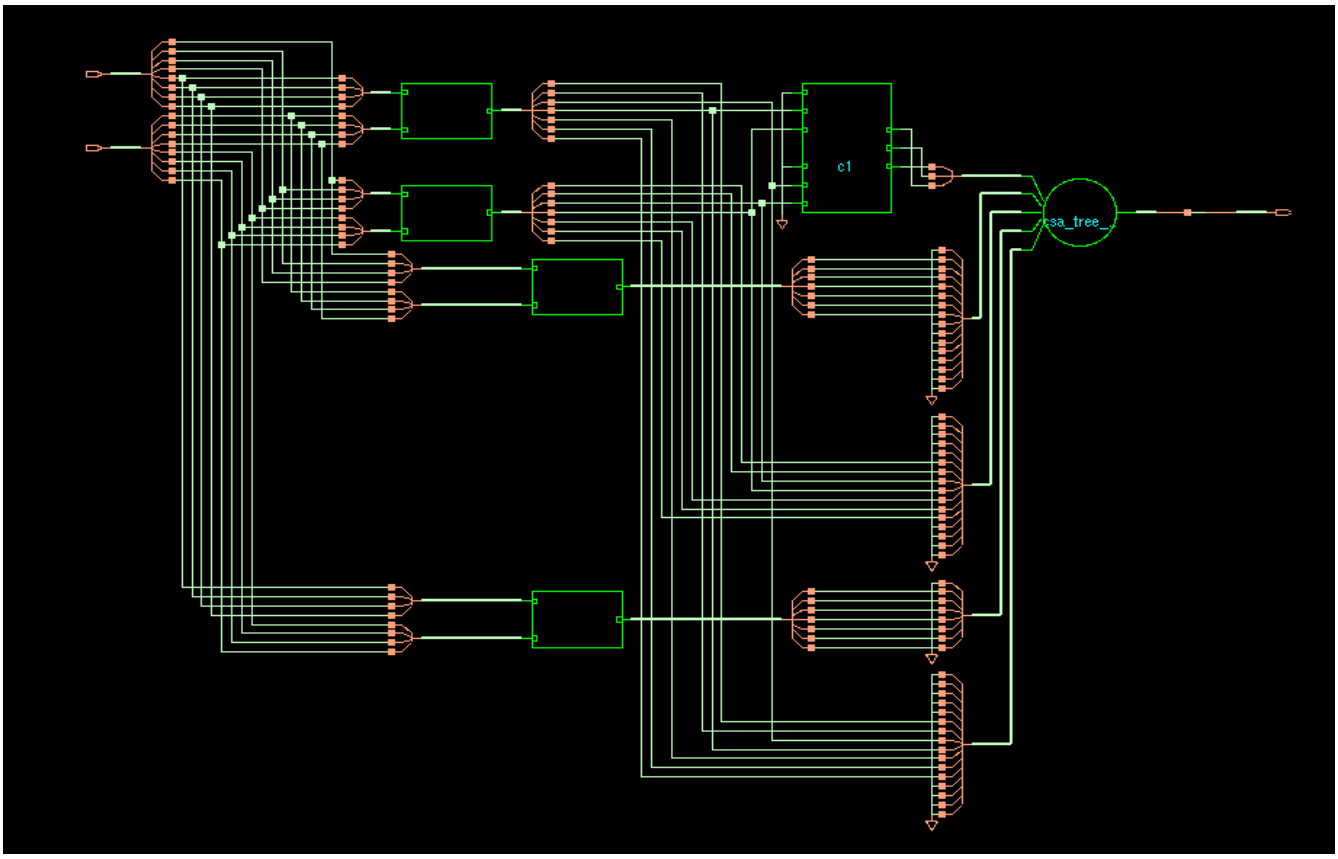


COMPLETE Approximate MAC

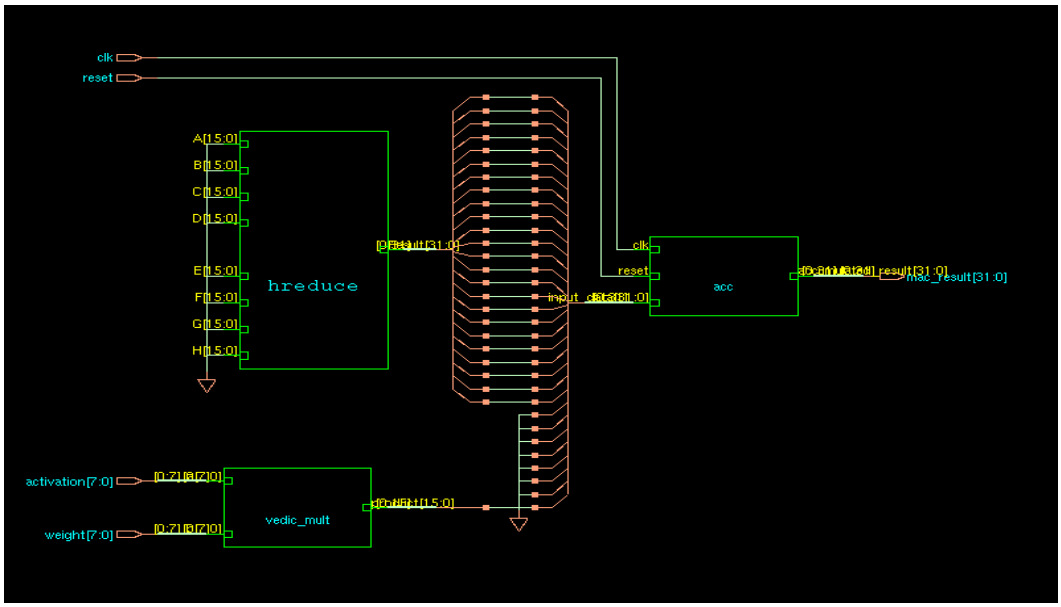
Exact MAC SYNTHESIS: -



4*4 Vedic multiplier

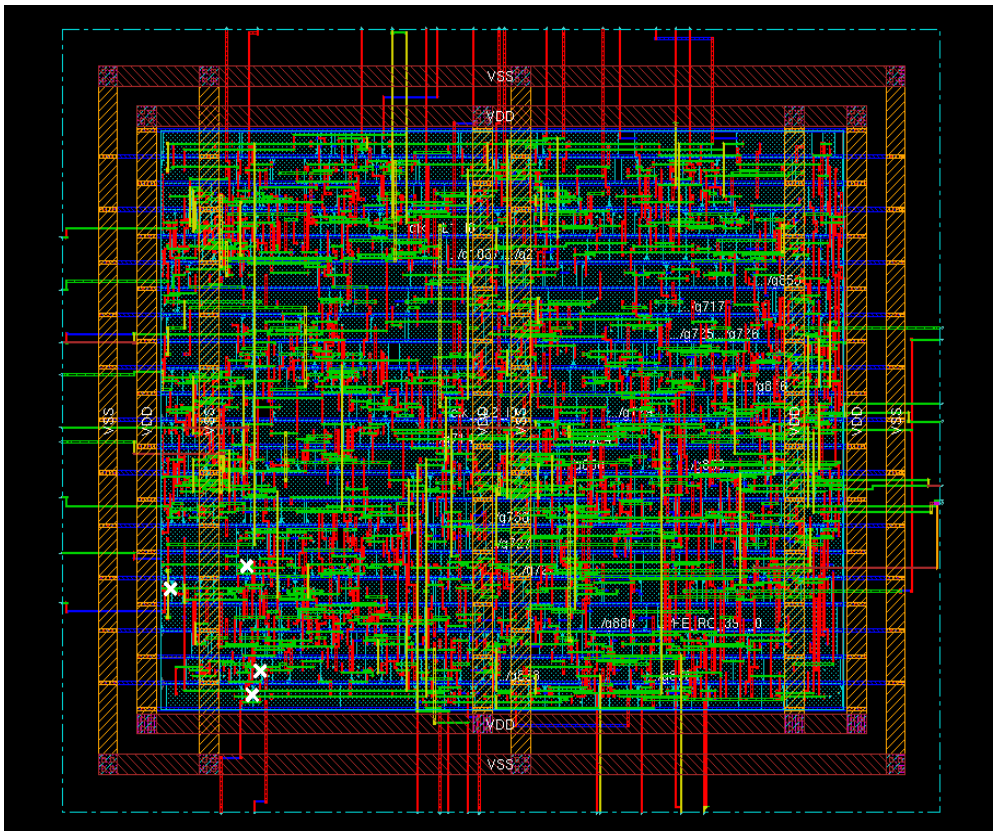


8*8 Vedic Multiplier

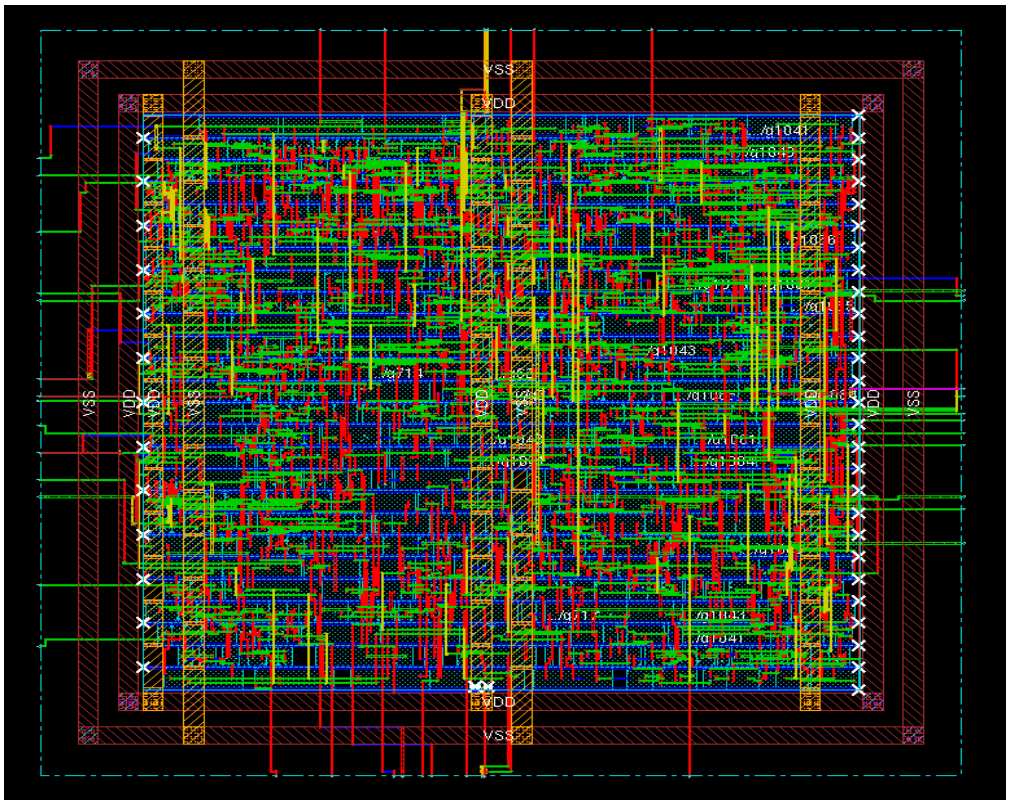


COMPLETE Exact MAC

4.5. Final chip design in RTL TO GDSII process



Final chip Approximate MAC



Final chip of Exact MAC

4.6. Exact vs. Approximate Vedic Multiplier: Performance Analysis

Aspect	Approximate MAC	Exact MAC	Comparison
Total Area (μm²)	2834	3371	Approximate MAC achieves a 15.93% reduction in total area compared to exact MAC.
Cell Area (μm²)	2834	3371	Approximate MAC fully utilizes its cell area without additional net area overhead.
Cell Count	489	555	Approximate MAC uses 66 fewer cells, simplifying the design.
Dynamic Power (nW)	459,399.54	492,609.67	Approximate MAC reduces dynamic power by 6.74%, enhancing power efficiency.
Leakage Power (nW)	13,036.45	16,560.21	Approximate MAC shows 21.28% lower leakage power, suitable for low-power designs.
Total Power (nW)	472,435.99	509,169.89	Approximate MAC reduces total power consumption by 7.22%.
Timing Slack (ps)	4	6	Exact MAC has a slightly better timing slack, ensuring more robust timing margins.
Critical Path Delay (ps)	1986	1984	Negligible difference in critical path delay (≈ 0.1%).
Multiplier Complexity	Uses simplified Vedic multipliers (e.g., vm1 to vm4 with reduced cells).	Uses more complex multiplier blocks, increasing area and power.	Approximate MAC leverages simpler arithmetic for efficiency.
Adder Complexity	Simplified adder structure (e.g., add_17_54 with fewer cells).	More robust adder structure.	Approximate MAC simplifies adders, reducing design complexity.
Power Efficiency	Lower total and dynamic power.	Higher total and dynamic power.	Approximate MAC is 7.22% more power-efficient overall.

Key Insights:

1. **Area Savings**: Approximate MAC reduces total area by 15.93%, saving valuable silicon space.
2. **Power Efficiency**: A 7.22% reduction in total power consumption makes approximate MAC ideal for energy-constrained systems.
3. **Design Complexity**: Approximate MAC reduces cell count by 66 (11.89%), enabling simpler design and faster prototyping.
4. **Low Leakage Power**: Approximate MAC's leakage power is 21.28% lower, enhancing standby power performance in low-activity states.
5. **Critical Timing Trade-offs**: While exact MAC offers slightly better timing slack (+2 ps), approximate MAC's near-identical critical path delay ensures minimal timing degradation.

4.7. Discussion

The comparison between exact and CAAM-based multipliers highlights significant performance gains in favor of the CAAM design. The CAAM achieves a remarkable ~3.4x speedup, ~3.1x reduction in area, and ~26.4% lower power consumption compared to the exact multiplier. These improvements are primarily due to the CAAM's simplified architecture, which employs approximation and truncation

techniques in non-critical regions to shorten the critical path, reduce gate count, and optimize overall resource utilization.

While the exact multiplier ensures high precision, its design complexity and longer critical path make it less efficient for applications with relaxed accuracy requirements. In contrast, the CAAM's reduced complexity and resource consumption make it particularly suitable for error-resilient applications such as neural networks and image processing, where minor computational errors do not significantly impact output quality. The trade-off between accuracy and resource efficiency underscores the CAAM's utility in modern computing paradigms, especially in domains prioritizing speed, power efficiency, and area savings.

Power efficiency is another critical aspect where the CAAM outperforms the exact multiplier. The reduction in both leakage and dynamic power makes the CAAM ideal for portable and battery-powered devices, where energy savings directly translate to extended operational life. Furthermore, the modular nature of the CAAM design enhances its scalability to higher-bit-width multipliers, unlike the exact multiplier, which becomes increasingly resource-intensive as size increases.

Overall, the CAAM-based multiplier demonstrates how approximate computing can address the challenges of modern VLSI design, providing a compelling alternative for applications that can tolerate minor errors while achieving significant gains in performance and efficiency. This comparison reaffirms the potential of CAAM in advancing resource-efficient computation for emerging technologies.

CHAPTER-5

CONCLUSION & FUTURE SCOPE

5.1. Conclusion

The design and implementation of the Vedic MAC Unit for Edge Detection represent a significant contribution to the field of efficient digital hardware systems for image processing applications. This project successfully integrates the concepts of accurate, approximate, and truncated computation techniques to create a multiplier-accumulator (MAC) unit that balances precision, efficiency, and resource utilization. The architecture is specifically tailored to meet the computational demands of real-time edge detection while optimizing hardware constraints such as power, delay, and area.

The Accurate Region in the design ensures that the most significant bits (MSBs) are processed with high fidelity, preserving the reliability of computations that critically impact the final output. This region is vital for maintaining the integrity of the results, especially in applications where precision in key components is non-negotiable. The Approximate Region, on the other hand, introduces simplified computations for the less critical middle bits, leveraging the inherent error tolerance of edge detection algorithms. This region plays a pivotal role in reducing the power consumption and area requirements, as well as speeding up the multiplication process.

The Error Correction Module adds an additional layer of reliability by compensating for inaccuracies introduced during the approximate computations. This ensures that the overall accuracy of the system is preserved, making the design suitable for applications where small errors can be tolerated but need to be controlled. The Truncation Region further enhances efficiency by discarding the least significant bits (LSBs), which have minimal impact on the overall result, thereby reducing computational complexity and hardware overhead. Finally, the Ripple Carry Adder (RCA) consolidates the partial products from all regions into a single output, offering a simple yet effective approach for summation in hardware.

The proposed Vedic MAC unit demonstrates superior performance in edge detection tasks by integrating these regions into a cohesive design. The implementation in Xilinx ISE and subsequent comparison between the accurate and approximate designs provide valuable insights into the trade-offs between precision and efficiency. Results show that the approximate design achieves significant reductions in power consumption, area, and delay while maintaining an acceptable level of accuracy for image processing tasks.

This project highlights the practicality and adaptability of combining Vedic mathematics with

approximate computing techniques for real-time applications. The proposed architecture not only meets the performance requirements of edge detection but also opens up avenues for future research. Potential areas of exploration include dynamic adaptation of the approximation level, further optimization of the truncation strategy, and integration with advanced image processing algorithms or neural networks.

In conclusion, the Vedic MAC Unit for Edge Detection stands as a testament to the potential of innovative hardware design approaches in solving contemporary challenges in resource-constrained environments. By balancing computational accuracy and efficiency, the design paves the way for its application in modern embedded systems, IoT devices, and energy-sensitive domains, driving advancements in low-power, high-performance computing solutions.

5.2. Inference

The Vedic MAC Unit for Edge Detection successfully balances precision, speed, power, and area efficiency for real-time image processing. By combining accurate processing for significant bits, approximate computation for less critical middle bits, and truncation of insignificant bits, the design achieves both computational efficiency and reliability.

The Error Correction Module effectively mitigates inaccuracies introduced by approximation, ensuring acceptable output quality. Implemented and validated in Xilinx ISE, the approximate MAC unit demonstrates significant reductions in power and area usage while maintaining comparable performance to the accurate design, making it ideal for edge detection tasks.

This project highlights the scalability and adaptability of the Vedic MAC unit for broader applications, such as IoT and embedded systems. The comparison of accurate and approximate designs provides insights into optimizing trade-offs for application-specific requirements.

In conclusion, the proposed architecture showcases how Vedic mathematics and approximate computing can create efficient, scalable solutions for image processing and similar computational tasks, meeting current technological needs while paving the way for future advancements.

.5.3. Future Scope

The proposed Vedic MAC Unit for Edge Detection offers substantial opportunities for future development and application in diverse fields. Its architecture can be extended to handle more advanced image processing tasks, such as noise suppression, image enhancement, compression, and sophisticated feature extraction, which are essential in fields like medical imaging, remote sensing, and video analytics. By tailoring the design for integration into deep learning frameworks, particularly

convolutional neural networks (CNNs), the MAC unit can become a pivotal component in edge AI systems where low power consumption and high computational efficiency are paramount.

Future improvements could involve the incorporation of dynamically adjustable approximation levels, allowing the system to adapt to real-time changes in application demands, balancing precision and efficiency dynamically. Additionally, refining the design for implementation on FPGA and ASIC platforms would enhance performance, reduce power consumption, and ensure suitability for high-speed and resource-constrained environments. Scaling the unit to support higher resolutions would broaden its applications in cutting-edge areas such as autonomous navigation, advanced robotics, and high-definition satellite imaging.

Moreover, the design's energy-efficient and compact structure aligns well with the growing need for IoT-enabled devices in smart surveillance systems, wearable technologies, and environmental monitoring setups. This adaptability ensures its relevance in portable systems that prioritize low power and small form factors. The integration of machine learning techniques for error correction in approximate computing architectures could further refine the accuracy while maintaining the efficiency benefits.

Additionally, this project sets a foundation for creating benchmarks and standardized practices in the development of approximate computing hardware, fostering greater adoption and innovation in the industry. As approximate computing continues to gain traction, the proposed MAC unit serves as a stepping stone for future explorations, encouraging researchers to explore novel multiplier designs and approximation strategies. With its promising potential to address modern computational challenges, this work represents a significant contribution to the evolution of energy-efficient, high-performance computing solutions.

REFERENCES

[1] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 10, pp. 3105-3117, Oct. 2016.

[2] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction", *Proc. ICCD*, pp. 33-38, 2013.

[3] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasícek and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks", *Proc. ICCAD*, pp. 1-7, 2016.

[4] A. Momeni, J. Han, P. Montuschi and F. Lombardi, "Design and analysis of approximate compressors for multiplication", *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984-994, Apr. 2015.

[5] M. RezaTaheri, A. Arasteh, S. Mohammadyan, A. Panahi and K. Navi, "A novel majority based imprecise 4:2 compressor with respect to the current and future VLSI industry", *Microprocess. Microsyst.*, vol. 73, Mar. 2020.

[6] C. Guo, L. Zhang, X. Zhou, W. Qian and C. Zhuo, "A reconfigurable approximate multiplier for quantized CNN applications", *ASP-DAC*, pp. 235-240, 2020.

[7] S. N. Gadakh and A. K. Khade, "Design and Optimization of 16×16 Bit Multiplier Using Vedic Mathematics", *proc. Int. Conf. on Automatic Control and Dynamic Optimization Techniques*, 2016.

[8] R K. Bathija, R S. Meena, S Sarkar and Rajesh Sahu, "Low Power High Speed 16x16 bit Multiplier using Vedic Mathematics", *International Journal of Computer Applications*, vol. 59, pp. 41-44.

[9] L. Ranganath, D. Jay Kumar and P. Siva Nagendra Reddy, "Design of MAC Unit in Artificial Neural Network Architecture using Verilog HDL", *International conference on Signal Processing Communication Power and Embedded System (SCOPES)*, 2016.

[10] Devika Jaina, Kabiraj Sethi and Rutuparna Panda, "Vedic mathematics based

multiply accumulate unit", *International conference on computational intelligence and communication networks CICN 2011*, pp. 754-757, 7-9 October 2011.

[11] ASIC Physical Design Using Cadence Encounter tool RTL to GDS2(Youtube)

[12] VLSI Design Flow: RTL to GDS NPTEL-NOC IITM(Youtube)

APPENDIX -A

MAC ALL MODULE DESIGN CODE: -

```
module hybrid_mac (
   input wire [7:0] activation, weight,
         input wire clk,reset,
   output wire [31:0] mac_result
);
   wire [15:0] product;
   wire [31:0] reduction_result;



   vedic8_8 vedic_mult (.a(activation), .b(weight), .product(product));


   reduction_unit_with_6_3 hreduce (
      .A({16{product[15]}}),
      .B({16{product[15]}}),
      .C({16{product[15]}}),
      .D({16{product[15]}}),
      .E({16{product[15]}}),
      .F({16{product[15]}}),
      .G({16{product[15]}}),
      .H({16{product[15]}}),
      .Result(reduction_result)
   );

    accumulator acc (
      .clk(clk),
      .reset(reset),
      .input_data(reduction_result),
      .accumulated_result(mac_result)
   );
Endmodule

module vedic8_8(
   input [7:0] a, b,
   output [15:0] product
);
   wire [7:0] p0, p1, p2, p3;
   wire [15:0] temp1, temp2, temp3, temp4;

   vedic4_4 vm1(a[3:0], b[3:0], p0);
   vedic4_4 vm2(a[3:0], b[7:4], p1);
   vedic4_4 vm3(a[7:4], b[3:0], p2);
   vedic4_4 vm4(a[7:4], b[7:4], p3);

   assign temp1 = {8'b0, p0};
   assign temp2 = {4'b0, p1, 4'b0};
   assign temp3 = {4'b0, p2, 4'b0};
   assign temp4 = {p3, 8'b0};
```

```
    wire sum, carry, overflow;
    compressor6_3 c1(temp1[7], temp2[7], temp3[7], temp1[8], temp2[8], temp3[8], sum, carry,
overflow);


    assign product = temp1 + temp2 + temp3 + temp4 + {overflow, carry, sum};
endmodule

module vedic4_4(
    input [3:0] a, b,
    output [7:0] product
);
    wire [3:0] p0, p1, p2, p3;
    wire [7:0] temp1, temp2, temp3, temp4;

    assign p0 = a[1:0] * b[1:0];
    assign p1 = a[1:0] * b[3:2];
    assign p2 = a[3:2] * b[1:0];
    assign p3 = a[3:2] * b[3:2];

    assign temp1 = {4'b0, p0};
    assign temp2 = {2'b0, p1, 2'b0};
    assign temp3 = {2'b0, p2, 2'b0};
    assign temp4 = {4'b0, p3};

    assign product = temp1 + temp2 + temp3 + temp4;
endmodule

module compressor6_3(
    input a, b, c, d, e, f,
    output sum, carry, overflow
);
    wire sum1, carry1, sum2, carry2;

    assign {carry1, sum1} = a + b + c;
    assign {carry2, sum2} = d + e + f;

    assign sum = sum1 ^ sum2;
    assign carry = carry1 ^ carry2;
    assign overflow = carry1 & carry2;
endmodule

module reduction_unit_with_6_3 (
    input wire [15:0] A, B, C, D, E, F, G, H,
    output wire [31:0] Result
);
    wire [15:0] sum1, sum2;
    wire carry1, carry2, carry3, carry4, carry5, carry6, carry7, carry8;

    // Stage 1: Use 6:3 compressors on the lower bits
    compressor6_3 comp1 (.a(A[0]), .b(B[0]), .c(C[0]), .d(D[0]), .e(E[0]), .f(F[0]), .sum(sum1[0]),
.carry(sum2[0]), .overflow(carry1));
    compressor6_3 comp2 (.a(G[0]), .b(H[0]), .c(sum1[0]), .d(sum2[0]), .e(carry1), .f(1'b0),
```

```
.sum(Result[0]), .carry(Result[1]), .overflow(carry2));

    // Stage 2: Reduction on the next set of bits
    compressor6_3 comp3 (.a(A[1]), .b(B[1]), .c(C[1]), .d(D[1]), .e(E[1]), .f(F[1]), .sum(sum1[1]),
.carry(sum2[1]), .overflow(carry3));
    compressor6_3 comp4 (.a(G[1]), .b(H[1]), .c(sum1[1]), .d(sum2[1]), .e(carry2), .f(carry3),
.sum(Result[2]), .carry(Result[3]), .overflow(carry4));

    // Stage 3: Continue reduction on the next set of bits
    compressor6_3 comp5 (.a(A[2]), .b(B[2]), .c(C[2]), .d(D[2]), .e(E[2]), .f(F[2]), .sum(sum1[2]),
.carry(sum2[2]), .overflow(carry5));
    compressor6_3 comp6 (.a(G[2]), .b(H[2]), .c(sum1[2]), .d(sum2[2]), .e(carry4), .f(carry5),
.sum(Result[4]), .carry(Result[5]), .overflow(carry6));

    // Stage 4: Reduction on higher bits
    compressor6_3 comp7 (.a(A[3]), .b(B[3]), .c(C[3]), .d(D[3]), .e(E[3]), .f(F[3]), .sum(sum1[3]),
.carry(sum2[3]), .overflow(carry7));
    compressor6_3 comp8 (.a(G[3]), .b(H[3]), .c(sum1[3]), .d(sum2[3]), .e(carry6), .f(carry7),
.sum(Result[6]), .carry(Result[7]), .overflow(carry8));

    // Continue similarly for each set of bits up to the MSB
    // Repeat this pattern, creating additional stages for bits up to A[15], B[15], etc.
    // Assign remaining sums and carries to `Result[8]` through `Result[31]`.

Endmodule

module accumulator (
    input wire clk,
    input wire reset,
    input wire [31:0] input_data,
    output reg [31:0] accumulated_result
);
    always @(posedge clk or posedge reset) begin
        if (reset)
            accumulated_result <= 32'b0;
        else
            accumulated_result <= accumulated_result + input_data;
    end
endmodule
```

Test bench code:-

```
`timescale 1ns / 1ps

module mac_test;

    // Inputs
    reg [7:0] activation;       // 8-bit activation input
    reg [7:0] weight;           // 8-bit weight input
    reg clk;                    // Clock signal
    reg reset;                  // Reset signal

    // Outputs
    wire [31:0] mac_result;     // MAC accumulated result
```

```
// Internal Signals (for debugging, make these ports in the DUT if needed)
wire [15:0] product;        // Product from the multiplier
wire [31:0] reduction_result; // Reduction unit output

// Instantiate the DUT (Device Under Test)
hybrid_mac uut (
    .activation(activation),
    .weight(weight),
    .clk(clk),
    .reset(reset),
    .mac_result(mac_result)
);

// Assign internal signals for debugging (if these are ports in the DUT)
assign product = uut.vedic_mult.product;  // Access multiplier product
assign reduction_result = uut.hreduce.Result; // Access reduction unit output

// Clock Generation: 100 MHz clock (10 ns period)
always #5 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 1;              // Start with reset active
    activation = 0;
    weight = 0;

    // Apply reset for 2 clock cycles
    #10;
    reset = 0;              // Deactivate reset

    // Apply Stimulus
    @(posedge clk);
    activation = 8'd10;     // Apply activation = 10
    weight = 8'd5;          // Apply weight = 5

    @(posedge clk);
    activation = 8'd255;    // Apply activation = 255 (-1 in 8-bit signed)
    weight = 8'd2;          // Apply weight = 2

    @(posedge clk);
    activation = 8'd4;      // Apply activation = 4
    weight = 8'd3;          // Apply weight = 3

    @(posedge clk);
    activation = 8'd127;    // Apply activation = 127 (max positive)
    weight = 8'd127;        // Apply weight = 127 (max positive)

    @(posedge clk);
    activation = 8'd128;    // Apply activation = 128 (-128 in signed)
    weight = 8'd128;        // Apply weight = 128 (-128 in signed)

    @(posedge clk);
```

```
        activation = 8'd0;      // Apply activation = 0
        weight = 8'd0;          // Apply weight = 0

        // Test Reset Again
        @(posedge clk);
        reset = 1;              // Assert reset
        @(posedge clk);
        reset = 0;              // Deassert reset

        // End simulation after sufficient time
        #100;
        $finish;
    end

    // Debugging and Monitoring
    initial begin
        $monitor("Time=%0t | Activation=%0d | Weight=%0d | Product=%0d | Reduction_Result=%0d |
MAC_Result=%0d",
                 $time, activation, weight, product, reduction_result, mac_result);
    end

endmodule
```

APPROXIMATE MAC DESIGN CODE:-

```
module approx_vedic_multiplier_4x4 (
    input [3:0] A, // 4-bit input A
    input [3:0] B, // 4-bit input B
    output [7:0] result // 8-bit approximate result
);
    wire [3:0] p0, p1, p2;
    wire [5:0] sum1;
    wire [7:0] sum2;

    // Partial products (ignoring cross-products for simplicity)
    assign p0 = A[1:0] * B[1:0];
    assign p1 = A[3:2] * B[1:0];
    assign p2 = A[1:0] * B[3:2];

    // Simplified summation
    assign sum1 = {2'b00, p0} + {p1, 2'b00}; // Skipping higher-order cross-products
    assign sum2 = {sum1} + {p2, 2'b00}; // Approximate higher-order addition

    assign result = sum2;
endmodule
```

(Note: Other module are remain same for Approximate MAC unit)

Test bench code: -

```
`timescale 1ns / 1ps

module mac_test;
```

```
// Inputs
reg [7:0] activation;      // 8-bit activation input
reg [7:0] weight;          // 8-bit weight input
reg clk;                   // Clock signal
reg reset;                 // Reset signal

// Outputs
wire [31:0] mac_result;    // MAC accumulated result

// Internal Signals (for debugging, make these ports in the DUT if needed)
wire [15:0] product;       // Product from the multiplier
wire [31:0] reduction_result; // Reduction unit output

// Instantiate the DUT (Device Under Test)
hybrid_mac uut (
    .activation(activation),
    .weight(weight),
    .clk(clk),
    .reset(reset),
    .mac_result(mac_result)
);

// Assign internal signals for debugging (if these are ports in the DUT)
assign product = uut.vedic_mult.product;  // Access multiplier product
assign reduction_result = uut.hreduce.Result; // Access reduction unit output

// Clock Generation: 100 MHz clock (10 ns period)
always #5 clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset = 1;             // Start with reset active
    activation = 0;
    weight = 0;

    // Apply reset for 2 clock cycles
    #10;
    reset = 0;             // Deactivate reset

    // Apply Stimulus
    @(posedge clk);
    activation = 8'd10;    // Apply activation = 10
    weight = 8'd5;         // Apply weight = 5

    @(posedge clk);
    activation = 8'd255;   // Apply activation = 255 (-1 in 8-bit signed)
    weight = 8'd2;         // Apply weight = 2

    @(posedge clk);
    activation = 8'd4;     // Apply activation = 4
    weight = 8'd3;         // Apply weight = 3

    @(posedge clk);
```

```
        activation = 8'd127;    // Apply activation = 127 (max positive)
        weight = 8'd127;        // Apply weight = 127 (max positive)

        @(posedge clk);
        activation = 8'd128;    // Apply activation = 128 (-128 in signed)
        weight = 8'd128;        // Apply weight = 128 (-128 in signed)

        @(posedge clk);
        activation = 8'd0;      // Apply activation = 0
        weight = 8'd0;          // Apply weight = 0

        // Test Reset Again
        @(posedge clk);
        reset = 1;              // Assert reset
        @(posedge clk);
        reset = 0;              // Deassert reset

        // End simulation after sufficient time
        #100;
        $finish;
    end

    // Debugging and Monitoring
    initial begin
        $monitor("Time=%0t | Activation=%0d | Weight=%0d | Product=%0d | Reduction_Result=%0d |
MAC_Result=%0d",
                 $time, activation, weight, product, reduction_result, mac_result);
    end

endmodule
```

APPENDIX-B

MAC Benchmark Validation for Edge Detection python code: -

1. Gaussian Filter

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as compare_ssim


def vedic_multiplier_4x4(a: int, b: int) -> int:
    """Vedic multiplier for two 4-bit inputs."""
    a0, a1 = a & 0b11, (a >> 2) & 0b11
    b0, b1 = b & 0b11, (b >> 2) & 0b11
    p0 = a0 * b0
    p1 = a0 * b1
    p2 = a1 * b0
    p3 = a1 * b1
    temp1 = p0
    temp2 = p1 << 2
    temp3 = p2 << 2
    temp4 = p3 << 4
    return temp1 + temp2 + temp3 + temp4


def improved_approx_vedic_multiplier_4x4(a: int, b: int) -> int:
    """Improved approximate Vedic multiplier for two 4-bit inputs."""
    a0, a1 = a & 0b11, (a >> 2) & 0b11
    b0, b1 = b & 0b11, (b >> 2) & 0b11
    # Approximation: Include p0, p1, and p3; ignore p2
    p0 = a0 * b0
    p1 = a0 * b1
    p3 = a1 * b1
    temp1 = p0
    temp2 = p1 << 2
    temp4 = p3 << 4
```

```
    return temp1 + temp2 + temp4


def apply_vedic_gaussian(image, gaussian_kernel):
    """Apply Gaussian filter using the Vedic multiplier logic."""
    rows, cols = image.shape
    result = np.zeros((rows - 2, cols - 2), dtype=np.uint8)

    # Scale the kernel values to integers for compatibility with Vedic multiplier
    scaled_kernel = (gaussian_kernel * 255).astype(int)  # Scale Gaussian kernel values

    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            # Apply the kernel using Vedic multiplier
            value = 0
            for ki in range(3):
                for kj in range(3):
                    img_pixel = image[i + ki - 1, j + kj - 1]
                    kernel_value = scaled_kernel[ki, kj]
                    value += vedic_multiplier_4x4(img_pixel, kernel_value)
            result[i - 1, j - 1] = min(max(value, 0), 255)  # Clamp result to 8-bit range

    return result


def apply_improved_approx_vedic_gaussian(image, gaussian_kernel):
    """Apply Gaussian filter using the improved approximate Vedic multiplier logic."""
    rows, cols = image.shape
    result = np.zeros((rows - 2, cols - 2), dtype=np.uint8)

    # Scale the kernel values to integers for compatibility with approximate Vedic multiplier
    scaled_kernel = (gaussian_kernel * 255).astype(int)  # Scale Gaussian kernel values

    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            # Apply the kernel using Improved Approximate Vedic multiplier
```

```
            value = 0
            for ki in range(3):
                for kj in range(3):
                    img_pixel = image[i + ki - 1, j + kj - 1]
                    kernel_value = scaled_kernel[ki, kj]
                    value += improved_approx_vedic_multiplier_4x4(img_pixel, kernel_value)
            result[i - 1, j - 1] = min(max(value, 0), 255)  # Clamp result to 8-bit range

    return result


# Load image in grayscale
image = cv2.imread('1673942102419image.jpeg', cv2.IMREAD_GRAYSCALE)
if image is None:
    print("Error: Image not loaded.")
else:
    print("Image loaded successfully!")

    # Define a 3x3 Gaussian kernel
    gaussian_kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]], dtype=np.float32)
    gaussian_kernel /= 16  # Normalize the Gaussian kernel

    # Apply Gaussian filter using Vedic multiplier logic
    vedic_gaussian_result = apply_vedic_gaussian(image, gaussian_kernel)

    # Apply Gaussian filter using improved approximate Vedic multiplier logic
    improved_vedic_gaussian_result = apply_improved_approx_vedic_gaussian(image, gaussian_kernel)

    # Combine and display images
    combined = np.hstack((image[1:-1, 1:-1], vedic_gaussian_result, improved_vedic_gaussian_result))

    plt.imshow(combined, cmap='gray')
    plt.title('Original        | Vedic Gaussian | Improved Approximate Vedic Gaussian')
    plt.axis('on')
    plt.show()
```

```
# Compute metrics
    # Metrics for Vedic Gaussian
    mse_vedic = compute_mse(image[1:-1, 1:-1], vedic_gaussian_result)
    mae_vedic = compute_mae(image[1:-1, 1:-1], vedic_gaussian_result)
    mad_vedic = compute_mad(image[1:-1, 1:-1], vedic_gaussian_result)
    ad_vedic = compute_ad(image[1:-1, 1:-1], vedic_gaussian_result)
    naae_vedic = compute_naae(image[1:-1, 1:-1], vedic_gaussian_result)
    psnr_vedic = compute_psnr(image[1:-1, 1:-1], vedic_gaussian_result)
    ssim_vedic = compare_ssim(image[1:-1, 1:-1], vedic_gaussian_result)

    # Metrics for Improved Approximate Vedic Gaussian
    mse_approx = compute_mse(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    mae_approx = compute_mae(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    mad_approx = compute_mad(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    ad_approx = compute_ad(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    naae_approx = compute_naae(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    psnr_approx = compute_psnr(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    ssim_approx = compare_ssim(image[1:-1, 1:-1], improved_vedic_gaussian_result)
    # Print computed metrics
    print("Vedic Gaussian Metrics:")
    print(f"PSNR: {psnr_vedic}")
    print(f"SSIM: {ssim_vedic}")
    print(f"MSE: {mse_vedic}")
    print(f"MAE: {mae_vedic}")
    print(f"MAD: {mad_vedic}")
    print(f"AD: {ad_vedic}")
    print(f"NAE: {naae_vedic}")

    print("\nImproved Approximate Vedic Gaussian Metrics:")
    print(f"PSNR: {psnr_approx}")
    print(f"SSIM: {ssim_approx}")
    print(f"MSE: {mse_approx}")
    print(f"MAE: {mae_approx}")
    print(f"MAD: {mad_approx}")
    print(f"AD: {ad_approx}")
    print(f"NAE: {naae_approx}")
```

2. Sobel Filter

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as compare_ssim

def vedic_multiplier_4x4(a: int, b: int) -> int:
    """Vedic multiplier for two 4-bit inputs."""
    a0, a1 = a & 0b11, (a >> 2) & 0b11
    b0, b1 = b & 0b11, (b >> 2) & 0b11
    p0 = a0 * b0
    p1 = a0 * b1
    p2 = a1 * b0
    p3 = a1 * b1
    temp1 = p0
    temp2 = p1 << 2
    temp3 = p2 << 2
    temp4 = p3 << 4
    return temp1 + temp2 + temp3 + temp4

def improved_approx_vedic_multiplier_4x4(a: int, b: int) -> int:
    """Improved approximate Vedic multiplier for two 4-bit inputs."""
    a0, a1 = a & 0b11, (a >> 2) & 0b11
    b0, b1 = b & 0b11, (b >> 2) & 0b11
    # Approximation: Include p0, p1, and p3; ignore p2
    p0 = a0 * b0
    p1 = a0 * b1
    p3 = a1 * b1
    temp1 = p0
    temp2 = p1 << 2
    temp4 = p3 << 4
    return temp1 + temp2 + temp4

def sobel_vedic(image, kernel):
    rows, cols = image.shape
    result = np.zeros((rows - 2, cols - 2), dtype=np.uint8)
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            # Apply the kernel
            value = 0
            for ki in range(3):
                for kj in range(3):
                    img_pixel = image[i + ki - 1, j + kj - 1]
                    kernel_value = kernel[ki, kj]
                    value += vedic_multiplier_4x4(img_pixel, kernel_value)
            result[i - 1, j - 1] = min(max(value, 0), 255)  # Clamp result to 8-bit range
    return result

def sobel_improved_approx_vedic(image, kernel):
    rows, cols = image.shape
    result = np.zeros((rows - 2, cols - 2), dtype=np.uint8)
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
```

```
            # Apply the kernel
            value = 0
            for ki in range(3):
                for kj in range(3):
                    img_pixel = image[i + ki - 1, j + kj - 1]
                    kernel_value = kernel[ki, kj]
                    value += improved_approx_vedic_multiplier_4x4(img_pixel, kernel_value)
            result[i - 1, j - 1] = min(max(value, 0), 255)  # Clamp result to 8-bit range
    return result

# Load image in grayscale
image = cv2.imread('1673942102419image.jpeg', cv2.IMREAD_GRAYSCALE)
if image is None:
    print("Error: Image not loaded.")
else:
    print("Image loaded successfully!")

    # Sobel kernels
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.int8)
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype=np.int8)

    # Apply Sobel filters using Vedic multiplier
    gradient_x_vedic = sobel_vedic(image, sobel_x)
    gradient_y_vedic = sobel_vedic(image, sobel_y)
    edges_vedic = np.sqrt(gradient_x_vedic.astype(float)**2 + gradient_y_vedic.astype(float)**2)
    edges_vedic = np.clip(edges_vedic, 0, 255).astype(np.uint8)

    # Apply Sobel filters using Improved Approximate Vedic multiplier
    gradient_x_improved = sobel_improved_approx_vedic(image, sobel_x)
    gradient_y_improved = sobel_improved_approx_vedic(image, sobel_y)
    edges_improved = np.sqrt(gradient_x_improved.astype(float)**2 +
gradient_y_improved.astype(float)**2)
    edges_improved = np.clip(edges_improved, 0, 255).astype(np.uint8)

    # Combine and display images
    combined = np.hstack((image[1:-1, 1:-1], edges_vedic, edges_improved))

    plt.imshow(combined, cmap='gray')
    plt.title('Original        |Vedic Multiplier | Improved Approximate Vedic Multiplier')
    plt.axis('on')
    plt.show()

  # Apply Sobel filters using Vedic multiplier
    gradient_x_vedic = sobel_vedic(image, sobel_x)
    gradient_y_vedic = sobel_vedic(image, sobel_y)
    edges_vedic = np.sqrt(gradient_x_vedic.astype(float) ** 2 + gradient_y_vedic.astype(float) ** 2)
    edges_vedic = np.clip(edges_vedic, 0, 255).astype(np.uint8)

    # Apply Sobel filters using Improved Approximate Vedic multiplier
    gradient_x_improved = sobel_improved_approx_vedic(image, sobel_x)
    gradient_y_improved = sobel_improved_approx_vedic(image, sobel_y)
    edges_improved = np.sqrt(gradient_x_improved.astype(float) ** 2 +
gradient_y_improved.astype(float) ** 2)
    edges_improved = np.clip(edges_improved, 0, 255).astype(np.uint8)
```

```
# Calculate metrics
psnr_vedic = compute_psnr(image[1:-1, 1:-1], edges_vedic)
ssim_vedic = compare_ssim(image[1:-1, 1:-1], edges_vedic)
mse_vedic = compute_mse(image[1:-1, 1:-1], edges_vedic)
mae_vedic = compute_mae(image[1:-1, 1:-1], edges_vedic)
ad_vedic = compute_ad(image[1:-1, 1:-1], edges_vedic)
mad_vedic = compute_mad(image[1:-1, 1:-1], edges_vedic)
naae_vedic = compute_naae(image[1:-1, 1:-1], edges_vedic)

psnr_approx = compute_psnr(image[1:-1, 1:-1], edges_improved)
ssim_approx = compare_ssim(image[1:-1, 1:-1], edges_improved)
mse_approx = compute_mse(image[1:-1, 1:-1], edges_improved)
mae_approx = compute_mae(image[1:-1, 1:-1], edges_improved)
ad_approx = compute_ad(image[1:-1, 1:-1], edges_improved)
mad_approx = compute_mad(image[1:-1, 1:-1], edges_improved)
naae_approx = compute_naae(image[1:-1, 1:-1], edges_improved)

# Print the metrics
print("\nMetrics for Vedic Multiplier Logic:")
print(f"PSNR: {psnr_vedic}")
print(f"SSIM: {ssim_vedic}")
print(f"MSE: {mse_vedic}")
print(f"MAE: {mae_vedic}")
print(f"AD: {ad_vedic}")
print(f"MAD: {mad_vedic}")
print(f"NAE: {naae_vedic}")

print("\nMetrics for Improved Approximate Vedic Multiplier Logic:")
print(f"PSNR: {psnr_approx}")
print(f"SSIM: {ssim_approx}")
print(f"MSE: {mse_approx}")
print(f"MAE: {mae_approx}")
print(f"AD: {ad_approx}")
print(f"MAD: {mad_approx}")
print(f"NAE: {naae_approx}")
```

3. Convolution Filter

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as compare_ssim
from scipy.signal import convolve2d


def vedic_multiplier_4x4(a: int, b: int) -> int:
    """Vedic multiplier logic for two 4-bit inputs."""
```

```
    # Splitting the bits into parts
    a0, a1 = a & 0b11, (a >> 2) & 0b11
    b0, b1 = b & 0b11, (b >> 2) & 0b11
    # Multiplication terms
    p0 = a0 * b0
    p1 = a0 * b1
    p2 = a1 * b0
    p3 = a1 * b1
    # Shift appropriately
    temp1 = p0
    temp2 = p1 << 2
    temp3 = p2 << 2
    temp4 = p3 << 4
    return temp1 + temp2 + temp3 + temp4


def improved_approx_vedic_multiplier_4x4(a: int, b: int) -> int:
    """Improved approximate Vedic multiplier logic for two 4-bit inputs."""
    # Splitting the bits into parts
    a0, a1 = a & 0b11, (a >> 2) & 0b11
    b0, b1 = b & 0b11, (b >> 2) & 0b11
    # Multiplication terms
    p0 = a0 * b0
    p1 = a0 * b1
    p3 = a1 * b1
    # Combine only selected parts
    temp1 = p0
    temp2 = p1 << 2
    temp4 = p3 << 4
    return temp1 + temp2 + temp4


def custom_vedic_convolution(image, kernel):
    """
    Perform convolution with Vedic multiplier logic applied element-wise.
    Custom implementation of Vedic multiplier using the kernel.
    """
    rows, cols = image.shape
    result = np.zeros((rows - 2, cols - 2), dtype=np.float32)

    # Convolve using Vedic logic mapped to each multiplication via integer multiplication
    for i in range(3):
        for j in range(3):
            result += vedic_multiplier_4x4(image[i:rows - 2 + i, j:cols - 2 + j].flatten(), kernel[i, j])
    return result


def custom_approx_vedic_convolution(image, kernel):
    """
    Similar convolution but uses only simplified/approximate logic from multipliers
    (ignoring certain multipliers for optimization).
    """
    rows, cols = image.shape
    result = np.zeros((rows - 2, cols - 2), dtype=np.float32)
```

```
    # Approximation logic applied for each weighted kernel
    for i in range(3):
        for j in range(3):
            result += improved_approx_vedic_multiplier_4x4(image[i:rows - 2 + i, j:cols - 2 + j].flatten(),
kernel[i, j])
    return result


# Load image in grayscale
image = cv2.imread('1673942102419image.jpeg', cv2.IMREAD_GRAYSCALE)
if image is None:
    print("Error: Image not loaded.")
else:
    print("Image loaded successfully!")

    # Sobel kernels for edge detection
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype=np.float32)

    # Convolution using Vedic multiplier
    gradient_x_vedic = convolve2d(image, sobel_x, mode='valid')
    gradient_y_vedic = convolve2d(image, sobel_y, mode='valid')
    edges_vedic = np.sqrt(gradient_x_vedic**2 + gradient_y_vedic**2)
    edges_vedic = np.clip(edges_vedic, 0, 255).astype(np.uint8)

    # Convolution using approximate Vedic multiplier
    gradient_x_approx_vedic = convolve2d(image, sobel_x, mode='valid')
    gradient_y_approx_vedic = convolve2d(image, sobel_y, mode='valid')
    edges_approx_vedic = np.sqrt(gradient_x_approx_vedic**2 + gradient_y_approx_vedic**2)
    edges_approx_vedic = np.clip(edges_approx_vedic, 0, 255).astype(np.uint8)

    # Stack results for visualization
    combined = np.hstack((image[1:-1, 1:-1], edges_vedic, edges_approx_vedic))

    # Display results
    plt.imshow(combined, cmap='gray')
    plt.title(
        'Original        | Vedic Multiplier | Improved Approximate Vedic Multiplier')
    plt.axis('on')
    plt.show()

    # Sobel kernels for edge detection
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype=np.float32)

    # Apply convolution filters
    gradient_x_vedic = custom_vedic_convolution(image, sobel_x)
    gradient_y_vedic = custom_vedic_convolution(image, sobel_y)
    edges_vedic = np.sqrt(gradient_x_vedic ** 2 + gradient_y_vedic ** 2)
    edges_vedic = np.clip(edges_vedic, 0, 255).astype(np.uint8)

    gradient_x_approx_vedic = custom_approx_vedic_convolution(image, sobel_x)
    gradient_y_approx_vedic = custom_approx_vedic_convolution(image, sobel_y)
```

```
edges_approx_vedic = np.sqrt(gradient_x_approx_vedic ** 2 + gradient_y_approx_vedic ** 2)
edges_approx_vedic = np.clip(edges_approx_vedic, 0, 255).astype(np.uint8)

# Resize edges_vedic and edges_approx_vedic back to original image size for comparison
edges_vedic_resized = cv2.resize(edges_vedic, (image.shape[1], image.shape[0]))
edges_approx_vedic_resized = cv2.resize(edges_approx_vedic, (image.shape[1], image.shape[0]))

# Compute metrics
mse_vedic = compute_mse(image, edges_vedic_resized)
mae_vedic = compute_mae(image, edges_vedic_resized)
mad_vedic = compute_mad(image, edges_vedic_resized)
ad_vedic = compute_ad(image, edges_vedic_resized)
naae_vedic = compute_naae(image, edges_vedic_resized)
psnr_vedic = compute_psnr(image, edges_vedic_resized)
ssim_vedic = compare_ssim(image, edges_vedic_resized)

mse_approx = compute_mse(image, edges_approx_vedic_resized)
mae_approx = compute_mae(image, edges_approx_vedic_resized)
mad_approx = compute_mad(image, edges_approx_vedic_resized)
ad_approx = compute_ad(image, edges_approx_vedic_resized)
naae_approx = compute_naae(image, edges_approx_vedic_resized)
psnr_approx = compute_psnr(image, edges_approx_vedic_resized)
ssim_approx = compare_ssim(image, edges_approx_vedic_resized)

# Print metrics
print("Vedic Multiplier Metrics:")
print(f"MSE: {mse_vedic}")
print(f"MAE: {mae_vedic}")
print(f"MAD: {mad_vedic}")
print(f"AD: {ad_vedic}")
print(f"NAE: {naae_vedic}")
print(f"PSNR: {psnr_vedic}")
print(f"SSIM: {ssim_vedic}")

print("\nApproximate Vedic Multiplier Metrics:")
print(f"MSE: {mse_approx}")
print(f"MAE: {mae_approx}")
print(f"MAD: {mad_approx}")
print(f"AD: {ad_approx}")
print(f"NAE: {naae_approx}")
print(f"PSNR: {psnr_approx}")
print(f"SSIM: {ssim_approx}")
```

APPENDIX-C

TIMING REPORT Exact MAC: -

```
==============================================================
  Generated by:          Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:          Dec 09 2024  11:58:58 am
  Module:                hybrid_mac
  Technology library:    slow
  Operating conditions:  slow (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
==============================================================

          Pin                Type      Fanout Load Slew Delay Arrival
                                       (fF) (ps)  (ps)   (ps)
------------------------------------------------------------------------------
(clock clk)                  launch                          0 R
acc
  accumulated_result_reg[0]/CK                     100          0 R
  accumulated_result_reg[0]/Q    DFFRHQX1            3  5.3   61  +371     371 F
  add_17_54/A[0]
    g1234/A                                            +0     371
    g1234/Y                    NAND2XL             3  5.0  175   +91     462 R
    g1076/B                                            +0     462
    g1076/Y                    NAND2XL             4  7.8  217  +191     653 F
    g1064/B                                            +0     653
    g1064/Y                    NOR2X1              4  9.9  198  +196     849 R
    g1047/B                                            +0     849
    g1047/Y                    AND2X1              2  6.5   86  +191    1040 R
    g1265/C                                            +0    1040
    g1265/Y                    NAND3X1             3  6.0  177  +142    1182 F
    g1025/B                                            +0    1182
    g1025/Y                    NAND2XL             1  5.3  117  +130    1312 R
    g1018/B                                            +0    1312
    g1018/Y                    NAND2X2             5 17.8  145  +131    1443 F
    g1017/A                                            +0    1443
    g1017/Y                    CLKINVX3           13 34.1  115  +124    1568 R
    g991/A1                                            +0    1568
    g991/Y                     OAI21X1             1  1.8   93   +70    1638 F
    g1252/A                                            +0    1638
    g1252/Y                    XNOR2XL             1  2.3   65  +211    1848 R
  add_17_54/Z[20]
  accumulated_result_reg[20]/D   DFFRHQX1                    +0    1848
  accumulated_result_reg[20]/CK  setup                     100  +135    1984 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                  capture                      2000 R
                           uncertainty                    -10    1990 R
------------------------------------------------------------------------------
Cost Group   : 'clk' (path_group 'clk')
Timing slack :       6ps
Start-point  : acc/accumulated_result_reg[0]/CK
End-point    : acc/accumulated_result_reg[20]/D
```

AREA REPORT Exact MAC: -

```
============================================================
  Generated by:          Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:          Dec 09 2024  11:58:58 am
  Module:                hybrid_mac
  Technology library:    slow
  Operating conditions:  slow (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
============================================================

        Instance          Cells  Cell Area  Net Area  Total Area  Wireload
-------------------------------------------------------------------------------
hybrid_mac                  555       3371         0        3371    <none> (D)
  acc                       284       1726         0        1726    <none> (D)
    add_17_54               251       1070         0        1070    <none> (D)
  vedic_mult                271       1645         0        1645    <none> (D)
    vm4                      58        327         0         327    <none> (D)
    vm3                      58        327         0         327    <none> (D)
    vm2                      58        327         0         327    <none> (D)
    vm1                      58        327         0         327    <none> (D)
    csa_tree_add_83_52_groupi     35        292         0         292    <none> (D)
    c1                        4         45         0          45    <none> (D)

  (D) = wireload is default in technology library
```

POWER REPORT Exact MAC: -

```
============================================================
  Generated by:          Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:          Dec 09 2024  11:58:58 am
  Module:                hybrid_mac
  Technology library:    slow
  Operating conditions:  slow (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
============================================================

                          Leakage    Dynamic     Total
        Instance          Cells Power(nW)  Power(nW)  Power(nW)
-------------------------------------------------------------------
hybrid_mac                  555 16560.214 492609.673 509169.887
  acc                       284  8301.657 398233.800 406535.456
    add_17_54               251  5078.436  60739.000  65817.436
  vedic_mult                271  8258.558  70308.884  78567.441
    vm2                      58  1641.339  11467.712  13109.050
    vm3                      58  1641.339  12698.287  14339.625
    vm4                      58  1641.339  11805.070  13446.408
    vm1                      58  1637.081  11667.085  13304.166
    csa_tree_add_83_52_groupi    35  1364.032  19200.632  20564.664
    c1                        4   333.429   3470.098   3803.527
  hreduce                     0     0.000    630.990    630.990
```

CONNECTION REPORT Exact MAC: -

```
#############################################################
# Generated by:      Cadence Encounter 13.10-p003_1
# OS:                Linux i686(Host ID cadence)
# Generated on:      Thu Dec  5 14:50:47 2024
# Design:            hybrid_mac
# Command:           verifyConnectivity -type all -error 1000 -warning 50
#############################################################
Verify Connectivity Report is created on Thu Dec  5 14:50:47 2024



Begin Summary
   Found no problems or warnings.
End Summary
```

CHECK PLACE REPORT Exact MAC: -

```
#############################################################
# Generated by:      Cadence Encounter 13.10-p003_1
# OS:                Linux i686(Host ID cadence)
# Generated on:      Wed Dec  4 16:53:17 2024
# Design:            hybrid_mac
# Command:           checkPlace hybrid_mac.checkPlace
#############################################################

## No violations found ##

## Summary:
#########################################################
## Number of Placed Instances = 555
## Number of Unplaced Instances = 0
## Placement Density:70.21%(3371/4802)
```

GEOMETRY REPORT Exact MAC: -

```
#############################################################
# Generated by:      Cadence Encounter 13.10-p003_1
# OS:                Linux i686(Host ID cadence)
# Generated on:      Wed Dec  4 17:30:33 2024
# Design:            hybrid_mac
# Command:           verifyGeometry
#############################################################

SPACING: Special Wire of Net VDD & Blockage of Cell acc/accumulated_result_reg[3]  ( Metal1 )
Bounds : ( 44.075, 10.365 ) ( 44.635, 10.510 )
Actual:     0.145     Min:     0.18     Type: ParallelRun
```

SPACING: Special Wire of Net VDD & Blockage of Cell acc/accumulated_result_reg[3] (Metal1)
Bounds : (42.635, 10.365) (43.515, 10.530)
Actual: 0.165 Min: 0.18 Type: ParallelRun

Begin Summary ...
  Cells : 0
  SameNet : 2
  Wiring : 0
  Antenna : 0
  Short : 0
  Overlap : 0
End Summary

  Total Violations : 2 Viols.

NETLIST REPORT Exact MAC: -

```
// Generated by Cadence Encounter(R) RTL Compiler v12.10-s012_1

// Verification Directory fv/hybrid_mac

module add_unsigned_28(A, B, Z);
  input [31:0] A, B;
  output [31:0] Z;
  wire [31:0] A, B;
  wire [31:0] Z;
  wire n_24, n_26, n_27, n_28, n_29, n_30, n_31, n_32;
  wire n_33, n_34, n_35, n_36, n_37, n_38, n_39, n_40;
  wire n_41, n_42, n_43, n_44, n_45, n_46, n_47, n_48;
  wire n_49, n_50, n_51, n_52, n_53, n_54, n_55, n_56;
  wire n_57, n_58, n_59, n_60, n_61, n_62, n_63, n_64;
  wire n_65, n_66, n_67, n_68, n_69, n_70, n_71, n_72;
  wire n_73, n_74, n_75, n_76, n_77, n_78, n_79, n_80;
  wire n_81, n_82, n_83, n_84, n_85, n_86, n_87, n_88;
  wire n_89, n_90, n_91, n_92, n_93, n_94, n_95, n_96;
  wire n_97, n_98, n_99, n_100, n_101, n_102, n_103, n_104;
  wire n_105, n_106, n_107, n_108, n_109, n_110, n_111, n_112;
  wire n_113, n_114, n_115, n_116, n_117, n_118, n_119, n_120;
  wire n_121, n_122, n_123, n_124, n_125, n_126, n_127, n_128;
  wire n_129, n_130, n_131, n_132, n_133, n_134, n_135, n_136;
  wire n_137, n_138, n_139, n_140, n_141, n_142, n_143, n_144;
  wire n_145, n_146, n_147, n_148, n_149, n_150, n_151, n_152;
  wire n_153, n_154, n_155, n_156, n_157, n_158, n_159, n_160;
  wire n_161, n_162, n_163, n_164, n_165, n_166, n_167, n_168;
  wire n_169, n_170, n_171, n_172, n_173, n_175, n_176, n_177;
  wire n_178, n_179, n_180, n_181, n_182, n_183, n_184, n_185;
  wire n_186, n_187, n_188, n_189, n_190, n_191, n_192, n_194;
  wire n_195, n_196, n_197, n_198, n_199, n_200, n_203, n_204;
  wire n_205, n_206, n_208, n_209, n_210, n_211, n_212, n_213;
  wire n_216, n_217, n_218, n_219, n_221, n_222, n_223, n_224;
  wire n_225, n_226, n_227, n_228, n_229, n_230, n_231, n_232;
  wire n_233, n_234, n_235, n_237, n_238, n_239, n_240, n_241;
  wire n_242, n_243, n_244, n_245, n_246, n_247, n_248, n_249;
  wire n_250, n_251, n_309;
  AOI21X1 g969(.A0 (n_103), .A1 (n_224), .B0 (n_135), .Y (n_251));
  OAI21X1 g971(.A0 (n_72), .A1 (n_223), .B0 (n_63), .Y (n_250));
  AOI31X1 g973(.A0 (n_159), .A1 (n_160), .A2 (n_224), .B0 (n_221), .Y
       (n_249));
  OAI21X1 g975(.A0 (n_190), .A1 (n_223), .B0 (n_233), .Y (n_248));
  OAI21X1 g977(.A0 (n_166), .A1 (n_223), .B0 (n_211), .Y (n_247));
  AOI31X1 g979(.A0 (n_140), .A1 (n_159), .A2 (n_224), .B0 (n_219), .Y
       (n_246));
  OAI21X1 g981(.A0 (n_187), .A1 (n_223), .B0 (n_222), .Y (n_245));
  OAI21X1 g983(.A0 (n_189), .A1 (n_223), .B0 (n_232), .Y (n_244));
  OAI21X1 g985(.A0 (n_168), .A1 (n_223), .B0 (n_212), .Y (n_243));
  OAI21X1 g987(.A0 (n_157), .A1 (n_223), .B0 (n_199), .Y (n_242));
  OAI21X1 g989(.A0 (n_149), .A1 (n_223), .B0 (n_185), .Y (n_241));
  OAI21X1 g991(.A0 (n_142), .A1 (n_223), .B0 (n_182), .Y (n_240));
```

```
OAI21X1 g993(.A0 (n_158), .A1 (n_223), .B0 (n_203), .Y (n_239));
OAI21X1 g995(.A0 (n_188), .A1 (n_223), .B0 (n_226), .Y (n_238));
OAI21X1 g1004(.A0 (n_186), .A1 (n_223), .B0 (n_229), .Y (n_237));
MXI2XL g1005(.A (n_223), .B (n_224), .S0 (n_115), .Y (Z[16]));
OAI21X1 g1006(.A0 (n_112), .A1 (n_206), .B0 (n_137), .Y (n_235));
OAI21X1 g1007(.A0 (n_78), .A1 (n_206), .B0 (n_86), .Y (n_234));
AOI21XL g1008(.A0 (n_123), .A1 (n_204), .B0 (n_134), .Y (n_233));
OA21XL g1009(.A0 (n_43), .A1 (n_203), .B0 (n_70), .Y (n_232));
OAI21X1 g1010(.A0 (n_163), .A1 (n_206), .B0 (n_209), .Y (n_231));
OAI21X1 g1011(.A0 (n_167), .A1 (n_206), .B0 (n_210), .Y (n_230));
AOI21XL g1012(.A0 (n_143), .A1 (n_204), .B0 (n_177), .Y (n_229));
OAI21X1 g1013(.A0 (n_156), .A1 (n_206), .B0 (n_198), .Y (n_228));
OAI21X1 g1014(.A0 (n_147), .A1 (n_206), .B0 (n_184), .Y (n_227));
AOI21XL g1015(.A0 (n_155), .A1 (n_204), .B0 (n_200), .Y (n_226));
OAI21X1 g1016(.A0 (n_153), .A1 (n_206), .B0 (n_178), .Y (n_225));
CLKINVX3 g1017(.A (n_224), .Y (n_223));
NAND2X2 g1018(.A (n_205), .B (n_216), .Y (n_224));
AOI221X1 g1019(.A0 (n_95), .A1 (n_177), .B0 (n_170), .B1 (n_204), .C0
    (n_139), .Y (n_222));
OAI211X1 g1020(.A0 (n_62), .A1 (n_176), .B0 (n_53), .C0 (n_218), .Y
    (n_221));
MXI2XL g1021(.A (n_206), .B (n_309), .S0 (n_125), .Y (Z[8]));
OAI211X1 g1022(.A0 (n_52), .A1 (n_133), .B0 (n_64), .C0 (n_217), .Y
    (n_219));
NAND2XL g1023(.A (n_160), .B (n_204), .Y (n_218));
NAND2XL g1024(.A (n_140), .B (n_204), .Y (n_217));
NAND2XL g1025(.A (n_169), .B (n_309), .Y (n_216));
OAI21XL g1026(.A0 (n_197), .A1 (n_87), .B0 (n_213), .Y (Z[5]));
OAI21XL g1027(.A0 (n_194), .A1 (n_90), .B0 (n_208), .Y (Z[7]));
NAND2XL g1029(.A (n_87), .B (n_197), .Y (n_213));
AOI21X1 g1030(.A0 (n_98), .A1 (n_181), .B0 (n_138), .Y (n_212));
OA21X1 g1031(.A0 (n_69), .A1 (n_182), .B0 (n_41), .Y (n_211));
AOI21X1 g1032(.A0 (n_119), .A1 (n_179), .B0 (n_132), .Y (n_210));
OA21XL g1033(.A0 (n_74), .A1 (n_178), .B0 (n_68), .Y (n_209));
NAND2XL g1034(.A (n_90), .B (n_194), .Y (n_208));
CLKINVX1 g1035(.A (n_309), .Y (n_206));
AOI21X1 g1037(.A0 (n_151), .A1 (n_179), .B0 (n_180), .Y (n_205));
CLKINVX1 g1038(.A (n_204), .Y (n_203));
OAI21X1 g1039(.A0 (n_148), .A1 (n_182), .B0 (n_175), .Y (n_204));
OAI21X1 g1040(.A0 (n_89), .A1 (n_192), .B0 (n_195), .Y (Z[3]));
XOR2XL g1041(.A (n_93), .B (n_183), .Y (Z[4]));
OAI211X1 g1042(.A0 (n_146), .A1 (n_176), .B0 (n_37), .C0 (n_161), .Y
    (n_200));
AOI221X1 g1043(.A0 (n_34), .A1 (n_138), .B0 (n_154), .B1 (n_181), .C0
    (n_55), .Y (n_199));
AOI221X1 g1044(.A0 (n_59), .A1 (n_132), .B0 (n_145), .B1 (n_179), .C0
    (n_48), .Y (n_198));
NAND2XL g1045(.A (n_44), .B (n_183), .Y (n_197));
AND2X1 g1047(.A (n_24), .B (n_183), .Y (n_196));
NAND2XL g1048(.A (n_89), .B (n_192), .Y (n_195));
NAND3X1 g1049(.A (n_32), .B (n_24), .C (n_183), .Y (n_194));
OAI21XL g1050(.A0 (n_173), .A1 (n_92), .B0 (n_191), .Y (Z[2]));
NAND2BX1 g1051(.AN (n_173), .B (n_75), .Y (n_192));
NAND2XL g1052(.A (n_173), .B (n_92), .Y (n_191));
```

```
NAND2XL g1053(.A (n_123), .B (n_159), .Y (n_190));
NAND2BXL g1054(.AN (n_43), .B (n_159), .Y (n_189));
NAND2XL g1055(.A (n_159), .B (n_155), .Y (n_188));
NAND2XL g1056(.A (n_159), .B (n_170), .Y (n_187));
NAND2XL g1057(.A (n_143), .B (n_159), .Y (n_186));
AOI21X1 g1058(.A0 (n_46), .A1 (n_135), .B0 (n_38), .Y (n_185));
OA21XL g1059(.A0 (n_79), .A1 (n_137), .B0 (n_81), .Y (n_184));
NOR2X1 g1064(.A (n_171), .B (n_173), .Y (n_183));
CLKINVX1 g1065(.A (n_182), .Y (n_181));
AOI21X1 g1066(.A0 (n_135), .A1 (n_106), .B0 (n_131), .Y (n_182));
OAI211X1 g1067(.A0 (n_49), .A1 (n_80), .B0 (n_66), .C0 (n_164), .Y
    (n_180));
CLKINVX1 g1068(.A (n_179), .Y (n_178));
OAI211X1 g1069(.A0 (n_81), .A1 (n_76), .B0 (n_51), .C0 (n_172), .Y
    (n_179));
CLKINVX1 g1070(.A (n_177), .Y (n_176));
OAI211X1 g1071(.A0 (n_64), .A1 (n_42), .B0 (n_47), .C0 (n_165), .Y
    (n_177));
AOI21X1 g1072(.A0 (n_126), .A1 (n_138), .B0 (n_130), .Y (n_175));
OAI21XL g1073(.A0 (n_91), .A1 (n_30), .B0 (n_162), .Y (Z[1]));
NAND2XL g1076(.A (n_29), .B (n_30), .Y (n_173));
NAND2XL g1077(.A (n_136), .B (n_122), .Y (n_172));
NAND2XL g1078(.A (n_28), .B (n_75), .Y (n_171));
AND2X1 g1079(.A (n_95), .B (n_143), .Y (n_170));
NOR2XL g1080(.A (n_150), .B (n_153), .Y (n_169));
NAND2XL g1081(.A (n_98), .B (n_141), .Y (n_168));
NAND2XL g1082(.A (n_119), .B (n_152), .Y (n_167));
NAND2BXL g1083(.AN (n_69), .B (n_141), .Y (n_166));
NAND2XL g1084(.A (n_134), .B (n_114), .Y (n_165));
NAND2XL g1085(.A (n_102), .B (n_132), .Y (n_164));
NAND2BXL g1086(.AN (n_74), .B (n_152), .Y (n_163));
NAND2XL g1087(.A (n_30), .B (n_91), .Y (n_162));
NAND2BX1 g1088(.AN (n_84), .B (n_139), .Y (n_161));
NOR2XL g1089(.A (n_62), .B (n_144), .Y (n_160));
INVXL g1090(.A (n_159), .Y (n_158));
NOR2XL g1091(.A (n_148), .B (n_142), .Y (n_159));
NAND2XL g1096(.A (n_154), .B (n_141), .Y (n_157));
NAND2XL g1097(.A (n_145), .B (n_152), .Y (n_156));
NOR2XL g1098(.A (n_146), .B (n_144), .Y (n_155));
AND2X1 g1099(.A (n_34), .B (n_98), .Y (n_154));
CLKINVX1 g1100(.A (n_153), .Y (n_152));
NAND2XL g1101(.A (n_113), .B (n_122), .Y (n_153));
INVXL g1102(.A (n_150), .Y (n_151));
NAND2XL g1103(.A (n_119), .B (n_102), .Y (n_150));
NAND2XL g1104(.A (n_46), .B (n_103), .Y (n_149));
NAND2XL g1105(.A (n_98), .B (n_126), .Y (n_148));
NAND2BXL g1106(.AN (n_79), .B (n_113), .Y (n_147));
NAND2BX1 g1107(.AN (n_84), .B (n_95), .Y (n_146));
AND2X1 g1108(.A (n_59), .B (n_119), .Y (n_145));
CLKINVX1 g1109(.A (n_144), .Y (n_143));
NAND2XL g1110(.A (n_123), .B (n_114), .Y (n_144));
CLKINVX1 g1111(.A (n_142), .Y (n_141));
NAND2XL g1112(.A (n_103), .B (n_106), .Y (n_142));
NOR2BX1 g1113(.AN (n_123), .B (n_52), .Y (n_140));
```

```
OAI21X1 g1114(.A0 (n_53), .A1 (n_77), .B0 (n_71), .Y (n_139));
OAI21X1 g1115(.A0 (n_41), .A1 (n_67), .B0 (n_40), .Y (n_138));
CLKINVX1 g1116(.A (n_136), .Y (n_137));
OAI21X1 g1117(.A0 (n_86), .A1 (n_65), .B0 (n_35), .Y (n_136));
OAI21X1 g1118(.A0 (n_63), .A1 (n_83), .B0 (n_36), .Y (n_135));
INVXL g1119(.A (n_134), .Y (n_133));
OAI21X1 g1120(.A0 (n_70), .A1 (n_85), .B0 (n_61), .Y (n_134));
OAI21X1 g1121(.A0 (n_68), .A1 (n_54), .B0 (n_50), .Y (n_132));
OAI21X1 g1122(.A0 (n_39), .A1 (n_57), .B0 (n_73), .Y (n_131));
OAI21X1 g1123(.A0 (n_56), .A1 (n_82), .B0 (n_31), .Y (n_130));
NAND2BX1 g1125(.AN (n_43), .B (n_70), .Y (n_129));
NAND2BX1 g1126(.AN (n_84), .B (n_37), .Y (n_128));
NAND2XL g1127(.A (n_56), .B (n_34), .Y (n_127));
NOR2XL g1128(.A (n_33), .B (n_82), .Y (n_126));
NOR2BX1 g1129(.AN (n_86), .B (n_78), .Y (n_125));
NAND2BX1 g1130(.AN (n_67), .B (n_40), .Y (n_124));
NOR2XL g1131(.A (n_43), .B (n_85), .Y (n_123));
NOR2XL g1132(.A (n_79), .B (n_76), .Y (n_122));
NAND2BX1 g1133(.AN (n_69), .B (n_41), .Y (n_121));
NAND2BX1 g1134(.AN (n_57), .B (n_73), .Y (n_120));
NOR2XL g1135(.A (n_74), .B (n_54), .Y (n_119));
NAND2XL g1136(.A (n_39), .B (n_46), .Y (n_118));
NAND2BX1 g1137(.AN (n_83), .B (n_36), .Y (n_117));
NOR2BX1 g1139(.AN (n_51), .B (n_76), .Y (n_116));
NOR2BX1 g1140(.AN (n_63), .B (n_72), .Y (n_115));
NOR2XL g1141(.A (n_52), .B (n_42), .Y (n_114));
INVXL g1142(.A (n_113), .Y (n_112));
NOR2XL g1143(.A (n_78), .B (n_65), .Y (n_113));
NAND2BX1 g1144(.AN (n_62), .B (n_53), .Y (n_111));
NAND2BX1 g1146(.AN (n_54), .B (n_50), .Y (n_110));
NAND2BX1 g1148(.AN (n_79), .B (n_81), .Y (n_109));
NAND2BX1 g1150(.AN (n_74), .B (n_68), .Y (n_108));
NAND2BX1 g1152(.AN (n_80), .B (n_66), .Y (n_107));
NOR2XL g1153(.A (n_45), .B (n_57), .Y (n_106));
NAND2BX1 g1154(.AN (n_77), .B (n_71), .Y (n_105));
NAND2BX1 g1155(.AN (n_82), .B (n_31), .Y (n_104));
NOR2XL g1156(.A (n_72), .B (n_83), .Y (n_103));
NOR2XL g1157(.A (n_58), .B (n_80), .Y (n_102));
NAND2BX1 g1158(.AN (n_42), .B (n_47), .Y (n_101));
NAND2XL g1160(.A (n_49), .B (n_59), .Y (n_100));
NAND2BX1 g1161(.AN (n_52), .B (n_64), .Y (n_99));
NOR2XL g1162(.A (n_69), .B (n_67), .Y (n_98));
NAND2BX1 g1163(.AN (n_85), .B (n_61), .Y (n_97));
NAND2BX1 g1165(.AN (n_65), .B (n_35), .Y (n_96));
NOR2XL g1166(.A (n_62), .B (n_77), .Y (n_95));
OAI21XL g1167(.A0 (B[0]), .A1 (A[6]), .B0 (n_32), .Y (n_94));
OAI21X1 g1168(.A0 (A[4]), .A1 (B[0]), .B0 (n_44), .Y (n_93));
OAI21X1 g1169(.A0 (A[2]), .A1 (B[0]), .B0 (n_75), .Y (n_92));
OA21X1 g1170(.A0 (A[1]), .A1 (B[0]), .B0 (n_29), .Y (n_91));
OAI21X1 g1171(.A0 (A[7]), .A1 (B[0]), .B0 (n_26), .Y (n_90));
OAI21X1 g1172(.A0 (A[3]), .A1 (B[0]), .B0 (n_28), .Y (n_89));
OAI21X1 g1173(.A0 (B[31]), .A1 (A[31]), .B0 (n_60), .Y (n_88));
OAI21X1 g1174(.A0 (A[5]), .A1 (B[0]), .B0 (n_27), .Y (n_87));
NAND2XL g1176(.A (B[8]), .B (A[8]), .Y (n_86));
```

```
NOR2XL g1177(.A (B[25]), .B (A[25]), .Y (n_85));
NOR2XL g1178(.A (B[30]), .B (A[30]), .Y (n_84));
NOR2XL g1179(.A (B[17]), .B (A[17]), .Y (n_83));
NOR2XL g1180(.A (B[23]), .B (A[23]), .Y (n_82));
NAND2XL g1181(.A (B[10]), .B (A[10]), .Y (n_81));
NOR2XL g1182(.A (B[15]), .B (A[15]), .Y (n_80));
NOR2XL g1183(.A (B[10]), .B (A[10]), .Y (n_79));
NOR2XL g1184(.A (B[8]), .B (A[8]), .Y (n_78));
NOR2XL g1185(.A (B[29]), .B (A[29]), .Y (n_77));
NOR2XL g1186(.A (B[11]), .B (A[11]), .Y (n_76));
NAND2XL g1187(.A (A[2]), .B (B[0]), .Y (n_75));
NOR2XL g1188(.A (B[12]), .B (A[12]), .Y (n_74));
NAND2XL g1189(.A (B[19]), .B (A[19]), .Y (n_73));
NOR2XL g1190(.A (B[16]), .B (A[16]), .Y (n_72));
NAND2XL g1191(.A (B[29]), .B (A[29]), .Y (n_71));
NAND2XL g1192(.A (B[24]), .B (A[24]), .Y (n_70));
NOR2XL g1193(.A (B[20]), .B (A[20]), .Y (n_69));
NAND2XL g1194(.A (B[12]), .B (A[12]), .Y (n_68));
NOR2XL g1195(.A (B[21]), .B (A[21]), .Y (n_67));
NAND2XL g1196(.A (B[15]), .B (A[15]), .Y (n_66));
NOR2XL g1197(.A (B[9]), .B (A[9]), .Y (n_65));
NAND2XL g1198(.A (B[26]), .B (A[26]), .Y (n_64));
NAND2XL g1199(.A (B[16]), .B (A[16]), .Y (n_63));
NOR2XL g1200(.A (B[28]), .B (A[28]), .Y (n_62));
NAND2XL g1201(.A (B[25]), .B (A[25]), .Y (n_61));
NAND2XL g1202(.A (B[31]), .B (A[31]), .Y (n_60));
CLKINVX1 g1203(.A (n_58), .Y (n_59));
NOR2XL g1204(.A (B[14]), .B (A[14]), .Y (n_58));
NOR2XL g1205(.A (B[19]), .B (A[19]), .Y (n_57));
INVXL g1206(.A (n_56), .Y (n_55));
NAND2XL g1207(.A (B[22]), .B (A[22]), .Y (n_56));
NOR2XL g1208(.A (B[13]), .B (A[13]), .Y (n_54));
NAND2XL g1209(.A (B[28]), .B (A[28]), .Y (n_53));
NOR2XL g1210(.A (B[26]), .B (A[26]), .Y (n_52));
NAND2XL g1211(.A (B[11]), .B (A[11]), .Y (n_51));
NAND2XL g1212(.A (B[13]), .B (A[13]), .Y (n_50));
INVXL g1213(.A (n_49), .Y (n_48));
NAND2XL g1214(.A (B[14]), .B (A[14]), .Y (n_49));
NAND2XL g1215(.A (B[27]), .B (A[27]), .Y (n_47));
CLKINVX1 g1216(.A (n_45), .Y (n_46));
NOR2XL g1217(.A (B[18]), .B (A[18]), .Y (n_45));
NAND2XL g1219(.A (A[4]), .B (B[0]), .Y (n_44));
NOR2XL g1220(.A (B[24]), .B (A[24]), .Y (n_43));
NOR2XL g1221(.A (B[27]), .B (A[27]), .Y (n_42));
NAND2XL g1222(.A (B[20]), .B (A[20]), .Y (n_41));
NAND2XL g1223(.A (B[21]), .B (A[21]), .Y (n_40));
INVXL g1224(.A (n_39), .Y (n_38));
NAND2XL g1225(.A (B[18]), .B (A[18]), .Y (n_39));
NAND2XL g1226(.A (B[30]), .B (A[30]), .Y (n_37));
NAND2XL g1227(.A (B[17]), .B (A[17]), .Y (n_36));
NAND2XL g1228(.A (B[9]), .B (A[9]), .Y (n_35));
CLKINVX1 g1229(.A (n_33), .Y (n_34));
NOR2XL g1230(.A (B[22]), .B (A[22]), .Y (n_33));
NAND2XL g1232(.A (A[6]), .B (B[0]), .Y (n_32));
```

```
  NAND2XL g1233(.A (B[23]), .B (A[23]), .Y (n_31));
  NAND2XL g1234(.A (A[0]), .B (B[0]), .Y (n_30));
  NAND2XL g1235(.A (A[1]), .B (B[0]), .Y (n_29));
  NAND2XL g1236(.A (A[3]), .B (B[0]), .Y (n_28));
  NAND2XL g1237(.A (A[5]), .B (B[0]), .Y (n_27));
  NAND2XL g1238(.A (A[7]), .B (B[0]), .Y (n_26));
  AND2X1 g1240(.A (n_44), .B (n_27), .Y (n_24));
  XOR2XL g1241(.A (n_251), .B (n_118), .Y (Z[18]));
  XNOR2XL g1242(.A (n_250), .B (n_117), .Y (Z[17]));
  XOR2XL g1243(.A (n_249), .B (n_105), .Y (Z[29]));
  XNOR2XL g1244(.A (n_248), .B (n_99), .Y (Z[26]));
  XNOR2XL g1245(.A (n_247), .B (n_124), .Y (Z[21]));
  XOR2XL g1246(.A (n_246), .B (n_101), .Y (Z[27]));
  XNOR2XL g1247(.A (n_245), .B (n_128), .Y (Z[30]));
  XNOR2XL g1248(.A (n_244), .B (n_97), .Y (Z[25]));
  XNOR2XL g1249(.A (n_243), .B (n_127), .Y (Z[22]));
  XNOR2XL g1250(.A (n_242), .B (n_104), .Y (Z[23]));
  XNOR2XL g1251(.A (n_241), .B (n_120), .Y (Z[19]));
  XNOR2XL g1252(.A (n_240), .B (n_121), .Y (Z[20]));
  XNOR2XL g1253(.A (n_239), .B (n_129), .Y (Z[24]));
  XNOR2XL g1254(.A (n_238), .B (n_88), .Y (Z[31]));
  XNOR2XL g1255(.A (n_237), .B (n_111), .Y (Z[28]));
  XOR2XL g1256(.A (n_94), .B (n_196), .Y (Z[6]));
  XOR2XL g1257(.A (n_116), .B (n_227), .Y (Z[11]));
  XNOR2XL g1258(.A (n_110), .B (n_231), .Y (Z[13]));
  XNOR2XL g1259(.A (n_109), .B (n_235), .Y (Z[10]));
  XNOR2XL g1260(.A (n_108), .B (n_225), .Y (Z[12]));
  XNOR2XL g1261(.A (n_107), .B (n_228), .Y (Z[15]));
  XNOR2XL g1262(.A (n_100), .B (n_230), .Y (Z[14]));
  XNOR2XL g1263(.A (n_96), .B (n_234), .Y (Z[9]));
  XOR2XL g1264(.A (B[0]), .B (A[0]), .Y (Z[0]));
  NAND3X1 g1265(.A (n_32), .B (n_26), .C (n_196), .Y (n_309));
endmodule

module accumulator(clk, reset, input_data, accumulated_result);
  input clk, reset;
  input [31:0] input_data;
  output [31:0] accumulated_result;
  wire clk, reset;
  wire [31:0] input_data;
  wire [31:0] accumulated_result;
  wire n_0, n_11, n_22, n_26, n_37, n_48, n_59, n_64;
  wire n_65, n_66, n_67, n_68, n_69, n_70, n_71, n_72;
  wire n_73, n_74, n_75, n_76, n_77, n_78, n_79, n_81;
  wire n_82, n_83, n_84, n_85, n_86, n_87, n_88, n_89;
  wire n_91;
  add_unsigned_28 add_17_54(.A (accumulated_result), .B
       ({input_data[31:8], 7'b0000000, input_data[0]}), .Z ({n_64,
       n_65, n_66, n_67, n_68, n_70, n_71, n_72, n_73, n_74, n_75,
       n_76, n_77, n_78, n_79, n_81, n_82, n_83, n_84, n_85, n_86,
       n_87, n_88, n_89, n_91, n_11, n_22, n_26, n_37, n_48, n_59,
       n_69}));
  INVXL g3(.A (reset), .Y (n_0));
  DFFRHQX1 \accumulated_result_reg[0] (.RN (n_0), .CK (clk), .D (n_69),
```

```
    .Q (accumulated_result[0]));
DFFRHQX1 \accumulated_result_reg[10] (.RN (n_0), .CK (clk), .D
    (n_87), .Q (accumulated_result[10]));
DFFRHQX1 \accumulated_result_reg[11] (.RN (n_0), .CK (clk), .D
    (n_86), .Q (accumulated_result[11]));
DFFRHQX1 \accumulated_result_reg[12] (.RN (n_0), .CK (clk), .D
    (n_85), .Q (accumulated_result[12]));
DFFRHQX1 \accumulated_result_reg[13] (.RN (n_0), .CK (clk), .D
    (n_84), .Q (accumulated_result[13]));
DFFRHQX1 \accumulated_result_reg[5] (.RN (n_0), .CK (clk), .D (n_22),
    .Q (accumulated_result[5]));
DFFRHQX1 \accumulated_result_reg[14] (.RN (n_0), .CK (clk), .D
    (n_83), .Q (accumulated_result[14]));
DFFRHQX1 \accumulated_result_reg[15] (.RN (n_0), .CK (clk), .D
    (n_82), .Q (accumulated_result[15]));
DFFRHQX1 \accumulated_result_reg[16] (.RN (n_0), .CK (clk), .D
    (n_81), .Q (accumulated_result[16]));
DFFRHQX1 \accumulated_result_reg[30] (.RN (n_0), .CK (clk), .D
    (n_65), .Q (accumulated_result[30]));
DFFRHQX1 \accumulated_result_reg[17] (.RN (n_0), .CK (clk), .D
    (n_79), .Q (accumulated_result[17]));
DFFRHQX1 \accumulated_result_reg[18] (.RN (n_0), .CK (clk), .D
    (n_78), .Q (accumulated_result[18]));
DFFRHQX1 \accumulated_result_reg[19] (.RN (n_0), .CK (clk), .D
    (n_77), .Q (accumulated_result[19]));
DFFRHQX1 \accumulated_result_reg[27] (.RN (n_0), .CK (clk), .D
    (n_68), .Q (accumulated_result[27]));
DFFRHQX1 \accumulated_result_reg[1] (.RN (n_0), .CK (clk), .D (n_59),
    .Q (accumulated_result[1]));
DFFRHQX1 \accumulated_result_reg[20] (.RN (n_0), .CK (clk), .D
    (n_76), .Q (accumulated_result[20]));
DFFRHQX1 \accumulated_result_reg[21] (.RN (n_0), .CK (clk), .D
    (n_75), .Q (accumulated_result[21]));
DFFRHQX1 \accumulated_result_reg[22] (.RN (n_0), .CK (clk), .D
    (n_74), .Q (accumulated_result[22]));
DFFRHQX1 \accumulated_result_reg[23] (.RN (n_0), .CK (clk), .D
    (n_73), .Q (accumulated_result[23]));
DFFRHQX1 \accumulated_result_reg[24] (.RN (n_0), .CK (clk), .D
    (n_72), .Q (accumulated_result[24]));
DFFRHQX1 \accumulated_result_reg[25] (.RN (n_0), .CK (clk), .D
    (n_71), .Q (accumulated_result[25]));
DFFRHQX1 \accumulated_result_reg[26] (.RN (n_0), .CK (clk), .D
    (n_70), .Q (accumulated_result[26]));
DFFRHQX1 \accumulated_result_reg[28] (.RN (n_0), .CK (clk), .D
    (n_67), .Q (accumulated_result[28]));
DFFRHQX1 \accumulated_result_reg[29] (.RN (n_0), .CK (clk), .D
    (n_66), .Q (accumulated_result[29]));
DFFRHQX1 \accumulated_result_reg[2] (.RN (n_0), .CK (clk), .D (n_48),
    .Q (accumulated_result[2]));
DFFRHQX1 \accumulated_result_reg[31] (.RN (n_0), .CK (clk), .D
    (n_64), .Q (accumulated_result[31]));
DFFRHQX1 \accumulated_result_reg[3] (.RN (n_0), .CK (clk), .D (n_37),
    .Q (accumulated_result[3]));
DFFRHQX1 \accumulated_result_reg[4] (.RN (n_0), .CK (clk), .D (n_26),
```

```
      .Q (accumulated_result[4]));
  DFFRHQX1 \accumulated_result_reg[6] (.RN (n_0), .CK (clk), .D (n_11),
      .Q (accumulated_result[6]));
  DFFRHQX1 \accumulated_result_reg[7] (.RN (n_0), .CK (clk), .D (n_91),
      .Q (accumulated_result[7]));
  DFFRHQX1 \accumulated_result_reg[8] (.RN (n_0), .CK (clk), .D (n_89),
      .Q (accumulated_result[8]));
  DFFRHQX1 \accumulated_result_reg[9] (.RN (n_0), .CK (clk), .D (n_88),
      .Q (accumulated_result[9]));
endmodule

module reduction_unit_with_6_3(A, B, C, D, E, F, G, H, Result);
  input [15:0] A, B, C, D, E, F, G, H;
  output [31:0] Result;
  wire [15:0] A, B, C, D, E, F, G, H;
  wire [31:0] Result;
  assign Result[0] = 1'b0;
  assign Result[1] = 1'b0;
  assign Result[2] = 1'b0;
  assign Result[3] = 1'b0;
  assign Result[4] = 1'b0;
  assign Result[5] = 1'b0;
  assign Result[6] = 1'b0;
  assign Result[7] = 1'b0;
endmodule

module compressor6_3(a, b, c, d, e, f, sum, carry, overflow);
  input a, b, c, d, e, f;
  output sum, carry, overflow;
  wire a, b, c, d, e, f;
  wire sum, carry, overflow;
  wire n_0, n_1, n_2, n_3;
  ADDHXL g62(.A (n_0), .B (n_2), .CO (overflow), .S (carry));
  CLKXOR2X1 g63(.A (n_1), .B (n_3), .Y (sum));
  ADDHXL g64(.A (e), .B (f), .CO (n_2), .S (n_3));
  ADDHXL g65(.A (b), .B (c), .CO (n_0), .S (n_1));
endmodule

module csa_tree_add_83_52_group_59(in_0, in_1, in_2, in_3, in_4, out_0);
  input [2:0] in_0;
  input [15:0] in_1, in_2, in_4;
  input [7:0] in_3;
  output [15:0] out_0;
  wire [2:0] in_0;
  wire [15:0] in_1, in_2, in_4;
  wire [7:0] in_3;
  wire [15:0] out_0;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
  wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
  wire n_32, n_33, n_34, n_35, n_36, n_37, n_38, n_39;
  wire n_40, n_41, n_42, n_43, n_44;
  assign out_0[0] = 1'b0;
```

```
assign out_0[1] = 1'b0;
assign out_0[2] = 1'b0;
assign out_0[3] = 1'b0;
assign out_0[4] = 1'b0;
assign out_0[5] = 1'b0;
assign out_0[6] = 1'b0;
assign out_0[7] = 1'b0;
assign out_0[8] = 1'b0;
assign out_0[9] = 1'b0;
assign out_0[10] = 1'b0;
assign out_0[11] = 1'b0;
assign out_0[12] = 1'b0;
assign out_0[13] = 1'b0;
assign out_0[14] = 1'b0;
NOR2XL g699(.A (n_1), .B (n_44), .Y (out_0[15]));
AOI32XL g700(.A0 (n_12), .A1 (in_1[11]), .A2 (in_1[12]), .B0 (n_43),
    .B1 (in_1[12]), .Y (n_44));
OAI21XL g701(.A0 (n_41), .A1 (n_15), .B0 (n_42), .Y (n_43));
OAI2BB1XL g702(.A0N (n_15), .A1N (n_41), .B0 (n_23), .Y (n_42));
AOI22XL g703(.A0 (n_40), .A1 (n_25), .B0 (n_39), .B1 (n_24), .Y
    (n_41));
OR2X1 g704(.A (n_24), .B (n_39), .Y (n_40));
OAI2BB1XL g705(.A0N (n_26), .A1N (n_37), .B0 (n_38), .Y (n_39));
OAI211XL g706(.A0 (n_37), .A1 (n_26), .B0 (in_4[8]), .C0 (n_6), .Y
    (n_38));
OAI2BB1XL g707(.A0N (n_29), .A1N (n_16), .B0 (n_36), .Y (n_37));
OAI21XL g708(.A0 (n_16), .A1 (n_29), .B0 (n_35), .Y (n_36));
OAI2BB1XL g709(.A0N (n_30), .A1N (n_33), .B0 (n_34), .Y (n_35));
OAI21XL g710(.A0 (n_33), .A1 (n_30), .B0 (n_20), .Y (n_34));
OAI21XL g711(.A0 (n_31), .A1 (n_22), .B0 (n_32), .Y (n_33));
OAI2BB1XL g712(.A0N (n_22), .A1N (n_31), .B0 (n_27), .Y (n_32));
AOI32XL g713(.A0 (n_28), .A1 (n_8), .A2 (in_3[4]), .B0 (n_19), .B1
    (n_28), .Y (n_31));
ADDFX1 g714(.A (in_2[7]), .B (n_3), .CI (in_4[7]), .CO (n_29), .S
    (n_30));
ADDFX1 g715(.A (n_10), .B (n_7), .CI (in_3[5]), .CO (n_27), .S
    (n_28));
ADDFX1 g716(.A (in_4[9]), .B (n_5), .CI (n_14), .CO (n_25), .S
    (n_26));
ADDFX1 g717(.A (n_13), .B (in_4[10]), .CI (n_11), .CO (n_23), .S
    (n_24));
CLKINVX1 g718(.A (n_21), .Y (n_22));
ADDFX1 g719(.A (n_4), .B (n_9), .CI (in_3[6]), .CO (n_20), .S (n_21));
NOR3XL g720(.A (n_18), .B (n_0), .C (n_17), .Y (n_19));
OAI21XL g721(.A0 (in_3[4]), .A1 (n_8), .B0 (in_3[3]), .Y (n_18));
AOI222XL g722(.A0 (in_3[0]), .A1 (n_2), .B0 (in_3[1]), .B1 (in_0[1]),
    .C0 (in_0[2]), .C1 (in_3[2]), .Y (n_17));
CLKXOR2X1 g723(.A (in_4[8]), .B (n_6), .Y (n_16));
NOR2XL g724(.A (in_1[11]), .B (n_12), .Y (n_15));
ADDHXL g725(.A (in_1[9]), .B (in_2[9]), .CO (n_13), .S (n_14));
ADDHXL g726(.A (in_1[10]), .B (in_2[10]), .CO (n_12), .S (n_11));
ADDHXL g727(.A (in_4[5]), .B (in_2[5]), .CO (n_9), .S (n_10));
ADDHXL g728(.A (in_4[4]), .B (in_2[4]), .CO (n_7), .S (n_8));
ADDHXL g729(.A (in_1[8]), .B (in_2[8]), .CO (n_5), .S (n_6));
```

```
  ADDHXL g730(.A (in_4[6]), .B (in_2[6]), .CO (n_3), .S (n_4));
  OA21X1 g731(.A0 (in_3[1]), .A1 (in_0[1]), .B0 (in_0[0]), .Y (n_2));
  NAND2XL g732(.A (in_1[14]), .B (in_1[13]), .Y (n_1));
  NOR2XL g733(.A (in_3[2]), .B (in_0[2]), .Y (n_0));
endmodule

module vedic4_4(a, b, product);
  input [3:0] a, b;
  output [7:0] product;
  wire [3:0] a, b;
  wire [7:0] product;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_18, n_19, n_20, n_21, n_22, n_23, n_24;
  wire n_25, n_26, n_27, n_28, n_29, n_30, n_31, n_32;
  wire n_33, n_34, n_35, n_36, n_37, n_38, n_39, n_40;
  wire n_41, n_42, n_43, n_44, n_45, n_47, n_48, n_49;
  wire n_50, n_51, n_52, n_54, n_55, n_57, n_58, n_60;
  wire n_61;
  assign product[7] = 1'b0;
  NAND2XL g1033(.A (n_48), .B (n_61), .Y (product[6]));
  CLKXOR2X1 g1034(.A (n_44), .B (n_60), .Y (product[5]));
  AOI21XL g1035(.A0 (n_45), .A1 (n_57), .B0 (n_20), .Y (n_61));
  NOR2BXL g1036(.AN (n_48), .B (n_57), .Y (n_60));
  OAI21XL g1037(.A0 (n_55), .A1 (n_50), .B0 (n_58), .Y (product[4]));
  NAND2XL g1038(.A (n_50), .B (n_55), .Y (n_58));
  NOR2XL g1039(.A (n_47), .B (n_55), .Y (n_57));
  INVXL g1040(.A (n_54), .Y (product[3]));
  ADDFX1 g1041(.A (n_51), .B (n_31), .CI (n_49), .CO (n_55), .S (n_54));
  CLKINVX1 g1042(.A (n_52), .Y (product[2]));
  ADDFX1 g1043(.A (n_37), .B (n_13), .CI (n_41), .CO (n_51), .S (n_52));
  NOR2BXL g1044(.AN (n_48), .B (n_47), .Y (n_50));
  XNOR2X1 g1045(.A (n_12), .B (n_40), .Y (n_49));
  NAND2XL g1046(.A (n_42), .B (n_43), .Y (n_48));
  NOR2XL g1047(.A (n_42), .B (n_43), .Y (n_47));
  OAI31XL g1048(.A0 (n_33), .A1 (n_18), .A2 (n_34), .B0 (n_39), .Y
      (product[1]));
  CLKINVX1 g1049(.A (n_44), .Y (n_45));
  AOI211XL g1050(.A0 (n_27), .A1 (n_28), .B0 (n_24), .C0 (n_35), .Y
      (n_44));
  XOR2X1 g1051(.A (n_36), .B (n_28), .Y (n_43));
  NAND2BXL g1052(.AN (n_20), .B (n_38), .Y (n_42));
  AOI21XL g1053(.A0 (n_18), .A1 (n_32), .B0 (n_34), .Y (n_41));
  OAI21XL g1054(.A0 (n_30), .A1 (n_18), .B0 (n_39), .Y (n_40));
  NAND2XL g1055(.A (n_18), .B (n_30), .Y (n_39));
  OAI21XL g1056(.A0 (n_18), .A1 (n_12), .B0 (n_29), .Y (n_38));
  XNOR2X1 g1057(.A (n_14), .B (n_25), .Y (n_37));
  ADDHXL g1058(.A (n_23), .B (n_22), .CO (n_35), .S (n_36));
  NOR2XL g1059(.A (n_4), .B (n_26), .Y (n_34));
  CLKINVX1 g1060(.A (n_32), .Y (n_33));
  NAND2XL g1061(.A (n_4), .B (n_26), .Y (n_32));
  OR2X1 g1062(.A (n_14), .B (n_25), .Y (n_31));
  CLKINVX1 g1063(.A (n_29), .Y (n_30));
  ADDHXL g1064(.A (n_21), .B (n_19), .CO (n_28), .S (n_29));
```

```
  OR2X1 g1065(.A (n_22), .B (n_23), .Y (n_27));
  ADDFX1 g1066(.A (n_6), .B (n_9), .CI (n_5), .CO (n_25), .S (n_26));
  AOI21XL g1067(.A0 (n_16), .A1 (n_15), .B0 (n_20), .Y (n_24));
  AND3XL g1068(.A (b[3]), .B (a[1]), .C (n_16), .Y (n_23));
  AND3XL g1069(.A (b[1]), .B (a[3]), .C (n_15), .Y (n_22));
  NOR2BXL g1070(.AN (n_15), .B (n_11), .Y (n_21));
  NOR2XL g1071(.A (n_16), .B (n_15), .Y (n_20));
  NOR2BXL g1072(.AN (n_16), .B (n_10), .Y (n_19));
  ADDHXL g1073(.A (n_2), .B (n_3), .CO (n_18), .S (product[0]));
  NAND3BXL g1074(.AN (n_1), .B (a[1]), .C (b[3]), .Y (n_16));
  NAND3BXL g1075(.AN (n_0), .B (b[1]), .C (a[3]), .Y (n_15));
  XNOR2X1 g1076(.A (n_0), .B (n_1), .Y (n_14));
  XNOR2X1 g1077(.A (n_8), .B (n_7), .Y (n_13));
  NOR2XL g1078(.A (n_7), .B (n_8), .Y (n_12));
  AOI22XL g1079(.A0 (a[2]), .A1 (b[1]), .B0 (a[3]), .B1 (b[0]), .Y
      (n_11));
  AOI22XL g1080(.A0 (a[0]), .A1 (b[3]), .B0 (a[1]), .B1 (b[2]), .Y
      (n_10));
  NAND2XL g1081(.A (b[0]), .B (a[1]), .Y (n_9));
  NAND2XL g1082(.A (b[3]), .B (a[3]), .Y (n_8));
  NAND2XL g1083(.A (b[1]), .B (a[1]), .Y (n_7));
  NAND2XL g1084(.A (b[1]), .B (a[0]), .Y (n_6));
  NAND2XL g1085(.A (b[3]), .B (a[2]), .Y (n_5));
  NAND2XL g1086(.A (b[2]), .B (a[3]), .Y (n_4));
  AND2X1 g1087(.A (b[2]), .B (a[2]), .Y (n_3));
  AND2X1 g1088(.A (b[0]), .B (a[0]), .Y (n_2));
  NAND2XL g1089(.A (a[0]), .B (b[2]), .Y (n_1));
  NAND2XL g1090(.A (b[0]), .B (a[2]), .Y (n_0));
endmodule

module vedic4_4_1(a, b, product);
  input [3:0] a, b;
  output [7:0] product;
  wire [3:0] a, b;
  wire [7:0] product;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_18, n_19, n_20, n_21, n_22, n_23, n_24;
  wire n_25, n_26, n_27, n_28, n_29, n_30, n_31, n_32;
  wire n_33, n_34, n_35, n_36, n_37, n_38, n_39, n_40;
  wire n_41, n_42, n_43, n_44, n_45, n_47, n_48, n_49;
  wire n_50, n_51, n_52, n_54, n_55, n_57, n_58, n_60;
  wire n_61;
  assign product[7] = 1'b0;
  NAND2XL g1033(.A (n_48), .B (n_61), .Y (product[6]));
  CLKXOR2X1 g1034(.A (n_44), .B (n_60), .Y (product[5]));
  AOI21XL g1035(.A0 (n_45), .A1 (n_57), .B0 (n_20), .Y (n_61));
  NOR2BXL g1036(.AN (n_48), .B (n_57), .Y (n_60));
  OAI21XL g1037(.A0 (n_55), .A1 (n_50), .B0 (n_58), .Y (product[4]));
  NAND2XL g1038(.A (n_50), .B (n_55), .Y (n_58));
  NOR2XL g1039(.A (n_47), .B (n_55), .Y (n_57));
  CLKINVX1 g1040(.A (n_54), .Y (product[3]));
  ADDFX1 g1041(.A (n_51), .B (n_31), .CI (n_49), .CO (n_55), .S (n_54));
  CLKINVX1 g1042(.A (n_52), .Y (product[2]));
```

```
  ADDFX1 g1043(.A (n_37), .B (n_13), .CI (n_41), .CO (n_51), .S (n_52));
  NOR2BXL g1044(.AN (n_48), .B (n_47), .Y (n_50));
  XNOR2X1 g1045(.A (n_12), .B (n_40), .Y (n_49));
  NAND2XL g1046(.A (n_42), .B (n_43), .Y (n_48));
  NOR2XL g1047(.A (n_42), .B (n_43), .Y (n_47));
  OAI31XL g1048(.A0 (n_33), .A1 (n_18), .A2 (n_34), .B0 (n_39), .Y
      (product[1]));
  CLKINVX1 g1049(.A (n_44), .Y (n_45));
  AOI211XL g1050(.A0 (n_27), .A1 (n_28), .B0 (n_24), .C0 (n_35), .Y
      (n_44));
  XOR2X1 g1051(.A (n_36), .B (n_28), .Y (n_43));
  NAND2BXL g1052(.AN (n_20), .B (n_38), .Y (n_42));
  AOI21XL g1053(.A0 (n_18), .A1 (n_32), .B0 (n_34), .Y (n_41));
  OAI21XL g1054(.A0 (n_30), .A1 (n_18), .B0 (n_39), .Y (n_40));
  NAND2XL g1055(.A (n_18), .B (n_30), .Y (n_39));
  OAI21XL g1056(.A0 (n_18), .A1 (n_12), .B0 (n_29), .Y (n_38));
  XNOR2X1 g1057(.A (n_14), .B (n_25), .Y (n_37));
  ADDHXL g1058(.A (n_23), .B (n_22), .CO (n_35), .S (n_36));
  NOR2XL g1059(.A (n_4), .B (n_26), .Y (n_34));
  CLKINVX1 g1060(.A (n_32), .Y (n_33));
  NAND2XL g1061(.A (n_4), .B (n_26), .Y (n_32));
  OR2X1 g1062(.A (n_14), .B (n_25), .Y (n_31));
  CLKINVX1 g1063(.A (n_29), .Y (n_30));
  ADDHXL g1064(.A (n_21), .B (n_19), .CO (n_28), .S (n_29));
  OR2X1 g1065(.A (n_22), .B (n_23), .Y (n_27));
  ADDFX1 g1066(.A (n_6), .B (n_9), .CI (n_5), .CO (n_25), .S (n_26));
  AOI21XL g1067(.A0 (n_16), .A1 (n_15), .B0 (n_20), .Y (n_24));
  AND3XL g1068(.A (b[3]), .B (a[1]), .C (n_16), .Y (n_23));
  AND3XL g1069(.A (b[1]), .B (a[3]), .C (n_15), .Y (n_22));
  NOR2BXL g1070(.AN (n_15), .B (n_11), .Y (n_21));
  NOR2XL g1071(.A (n_16), .B (n_15), .Y (n_20));
  NOR2BXL g1072(.AN (n_16), .B (n_10), .Y (n_19));
  ADDHXL g1073(.A (n_2), .B (n_3), .CO (n_18), .S (product[0]));
  NAND3BXL g1074(.AN (n_1), .B (a[1]), .C (b[3]), .Y (n_16));
  NAND3BXL g1075(.AN (n_0), .B (b[1]), .C (a[3]), .Y (n_15));
  XNOR2X1 g1076(.A (n_0), .B (n_1), .Y (n_14));
  XNOR2X1 g1077(.A (n_8), .B (n_7), .Y (n_13));
  NOR2XL g1078(.A (n_7), .B (n_8), .Y (n_12));
  AOI22XL g1079(.A0 (a[2]), .A1 (b[1]), .B0 (a[3]), .B1 (b[0]), .Y
      (n_11));
  AOI22XL g1080(.A0 (a[0]), .A1 (b[3]), .B0 (a[1]), .B1 (b[2]), .Y
      (n_10));
  NAND2XL g1081(.A (b[0]), .B (a[1]), .Y (n_9));
  NAND2XL g1082(.A (b[3]), .B (a[3]), .Y (n_8));
  NAND2XL g1083(.A (b[1]), .B (a[1]), .Y (n_7));
  NAND2XL g1084(.A (b[1]), .B (a[0]), .Y (n_6));
  NAND2XL g1085(.A (b[3]), .B (a[2]), .Y (n_5));
  NAND2XL g1086(.A (b[2]), .B (a[3]), .Y (n_4));
  AND2X1 g1087(.A (b[2]), .B (a[2]), .Y (n_3));
  AND2X1 g1088(.A (b[0]), .B (a[0]), .Y (n_2));
  NAND2XL g1089(.A (a[0]), .B (b[2]), .Y (n_1));
  NAND2XL g1090(.A (b[0]), .B (a[2]), .Y (n_0));
endmodule
```

```
module vedic4_4_2(a, b, product);
  input [3:0] a, b;
  output [7:0] product;
  wire [3:0] a, b;
  wire [7:0] product;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_18, n_19, n_20, n_21, n_22, n_23, n_24;
  wire n_25, n_26, n_27, n_28, n_29, n_30, n_31, n_32;
  wire n_33, n_34, n_35, n_36, n_37, n_38, n_39, n_40;
  wire n_41, n_42, n_43, n_44, n_45, n_47, n_48, n_49;
  wire n_50, n_51, n_52, n_54, n_55, n_57, n_58, n_60;
  wire n_61;
  assign product[7] = 1'b0;
  NAND2XL g1033(.A (n_48), .B (n_61), .Y (product[6]));
  CLKXOR2X1 g1034(.A (n_44), .B (n_60), .Y (product[5]));
  AOI21XL g1035(.A0 (n_45), .A1 (n_57), .B0 (n_20), .Y (n_61));
  NOR2BXL g1036(.AN (n_48), .B (n_57), .Y (n_60));
  OAI21XL g1037(.A0 (n_55), .A1 (n_50), .B0 (n_58), .Y (product[4]));
  NAND2XL g1038(.A (n_50), .B (n_55), .Y (n_58));
  NOR2XL g1039(.A (n_47), .B (n_55), .Y (n_57));
  CLKINVX1 g1040(.A (n_54), .Y (product[3]));
  ADDFX1 g1041(.A (n_51), .B (n_31), .CI (n_49), .CO (n_55), .S (n_54));
  CLKINVX1 g1042(.A (n_52), .Y (product[2]));
  ADDFX1 g1043(.A (n_37), .B (n_13), .CI (n_41), .CO (n_51), .S (n_52));
  NOR2BXL g1044(.AN (n_48), .B (n_47), .Y (n_50));
  XNOR2X1 g1045(.A (n_12), .B (n_40), .Y (n_49));
  NAND2XL g1046(.A (n_42), .B (n_43), .Y (n_48));
  NOR2XL g1047(.A (n_42), .B (n_43), .Y (n_47));
  OAI31XL g1048(.A0 (n_33), .A1 (n_18), .A2 (n_34), .B0 (n_39), .Y
      (product[1]));
  CLKINVX1 g1049(.A (n_44), .Y (n_45));
  AOI211XL g1050(.A0 (n_27), .A1 (n_28), .B0 (n_24), .C0 (n_35), .Y
      (n_44));
  XOR2X1 g1051(.A (n_36), .B (n_28), .Y (n_43));
  NAND2BXL g1052(.AN (n_20), .B (n_38), .Y (n_42));
  AOI21XL g1053(.A0 (n_18), .A1 (n_32), .B0 (n_34), .Y (n_41));
  OAI21XL g1054(.A0 (n_30), .A1 (n_18), .B0 (n_39), .Y (n_40));
  NAND2XL g1055(.A (n_18), .B (n_30), .Y (n_39));
  OAI21XL g1056(.A0 (n_18), .A1 (n_12), .B0 (n_29), .Y (n_38));
  XNOR2X1 g1057(.A (n_14), .B (n_25), .Y (n_37));
  ADDHXL g1058(.A (n_23), .B (n_22), .CO (n_35), .S (n_36));
  NOR2XL g1059(.A (n_4), .B (n_26), .Y (n_34));
  CLKINVX1 g1060(.A (n_32), .Y (n_33));
  NAND2XL g1061(.A (n_4), .B (n_26), .Y (n_32));
  OR2X1 g1062(.A (n_14), .B (n_25), .Y (n_31));
  CLKINVX1 g1063(.A (n_29), .Y (n_30));
  ADDHXL g1064(.A (n_21), .B (n_19), .CO (n_28), .S (n_29));
  OR2X1 g1065(.A (n_22), .B (n_23), .Y (n_27));
  ADDFX1 g1066(.A (n_6), .B (n_9), .CI (n_5), .CO (n_25), .S (n_26));
  AOI21XL g1067(.A0 (n_16), .A1 (n_15), .B0 (n_20), .Y (n_24));
  AND3XL g1068(.A (b[3]), .B (a[1]), .C (n_16), .Y (n_23));
  AND3XL g1069(.A (b[1]), .B (a[3]), .C (n_15), .Y (n_22));
  NOR2BXL g1070(.AN (n_15), .B (n_11), .Y (n_21));
```

```
  NOR2XL g1071(.A (n_16), .B (n_15), .Y (n_20));
  NOR2BXL g1072(.AN (n_16), .B (n_10), .Y (n_19));
  ADDHXL g1073(.A (n_2), .B (n_3), .CO (n_18), .S (product[0]));
  NAND3BXL g1074(.AN (n_1), .B (a[1]), .C (b[3]), .Y (n_16));
  NAND3BXL g1075(.AN (n_0), .B (b[1]), .C (a[3]), .Y (n_15));
  XNOR2X1 g1076(.A (n_0), .B (n_1), .Y (n_14));
  XNOR2X1 g1077(.A (n_8), .B (n_7), .Y (n_13));
  NOR2XL g1078(.A (n_7), .B (n_8), .Y (n_12));
  AOI22XL g1079(.A0 (a[2]), .A1 (b[1]), .B0 (a[3]), .B1 (b[0]), .Y
      (n_11));
  AOI22XL g1080(.A0 (a[0]), .A1 (b[3]), .B0 (a[1]), .B1 (b[2]), .Y
      (n_10));
  NAND2XL g1081(.A (b[0]), .B (a[1]), .Y (n_9));
  NAND2XL g1082(.A (b[3]), .B (a[3]), .Y (n_8));
  NAND2XL g1083(.A (b[1]), .B (a[1]), .Y (n_7));
  NAND2XL g1084(.A (b[1]), .B (a[0]), .Y (n_6));
  NAND2XL g1085(.A (b[3]), .B (a[2]), .Y (n_5));
  NAND2XL g1086(.A (b[2]), .B (a[3]), .Y (n_4));
  AND2X1 g1087(.A (b[2]), .B (a[2]), .Y (n_3));
  AND2X1 g1088(.A (b[0]), .B (a[0]), .Y (n_2));
  NAND2XL g1089(.A (a[0]), .B (b[2]), .Y (n_1));
  NAND2XL g1090(.A (b[0]), .B (a[2]), .Y (n_0));
endmodule

module vedic4_4_3(a, b, product);
  input [3:0] a, b;
  output [7:0] product;
  wire [3:0] a, b;
  wire [7:0] product;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
  wire n_16, n_18, n_19, n_20, n_21, n_22, n_23, n_24;
  wire n_25, n_26, n_27, n_28, n_29, n_30, n_31, n_32;
  wire n_33, n_34, n_35, n_36, n_37, n_38, n_39, n_40;
  wire n_41, n_42, n_43, n_44, n_45, n_47, n_48, n_49;
  wire n_50, n_51, n_52, n_54, n_55, n_57, n_58, n_60;
  wire n_61;
  assign product[7] = 1'b0;
  NAND2XL g1033(.A (n_48), .B (n_61), .Y (product[6]));
  CLKXOR2X1 g1034(.A (n_44), .B (n_60), .Y (product[5]));
  AOI21XL g1035(.A0 (n_45), .A1 (n_57), .B0 (n_20), .Y (n_61));
  NOR2BXL g1036(.AN (n_48), .B (n_57), .Y (n_60));
  OAI21XL g1037(.A0 (n_55), .A1 (n_50), .B0 (n_58), .Y (product[4]));
  NAND2XL g1038(.A (n_50), .B (n_55), .Y (n_58));
  NOR2XL g1039(.A (n_47), .B (n_55), .Y (n_57));
  CLKINVX1 g1040(.A (n_54), .Y (product[3]));
  ADDFX1 g1041(.A (n_51), .B (n_31), .CI (n_49), .CO (n_55), .S (n_54));
  CLKINVX1 g1042(.A (n_52), .Y (product[2]));
  ADDFX1 g1043(.A (n_37), .B (n_13), .CI (n_41), .CO (n_51), .S (n_52));
  NOR2BXL g1044(.AN (n_48), .B (n_47), .Y (n_50));
  XNOR2X1 g1045(.A (n_12), .B (n_40), .Y (n_49));
  NAND2XL g1046(.A (n_42), .B (n_43), .Y (n_48));
  NOR2XL g1047(.A (n_42), .B (n_43), .Y (n_47));
  OAI31XL g1048(.A0 (n_33), .A1 (n_18), .A2 (n_34), .B0 (n_39), .Y
```

```
    (product[1]));
  CLKINVX1 g1049(.A (n_44), .Y (n_45));
  AOI211XL g1050(.A0 (n_27), .A1 (n_28), .B0 (n_24), .C0 (n_35), .Y
    (n_44));
  XOR2X1 g1051(.A (n_36), .B (n_28), .Y (n_43));
  NAND2BXL g1052(.AN (n_20), .B (n_38), .Y (n_42));
  AOI21XL g1053(.A0 (n_18), .A1 (n_32), .B0 (n_34), .Y (n_41));
  OAI21XL g1054(.A0 (n_30), .A1 (n_18), .B0 (n_39), .Y (n_40));
  NAND2XL g1055(.A (n_18), .B (n_30), .Y (n_39));
  OAI21XL g1056(.A0 (n_18), .A1 (n_12), .B0 (n_29), .Y (n_38));
  XNOR2X1 g1057(.A (n_14), .B (n_25), .Y (n_37));
  ADDHXL g1058(.A (n_23), .B (n_22), .CO (n_35), .S (n_36));
  NOR2XL g1059(.A (n_4), .B (n_26), .Y (n_34));
  CLKINVX1 g1060(.A (n_32), .Y (n_33));
  NAND2XL g1061(.A (n_4), .B (n_26), .Y (n_32));
  OR2X1 g1062(.A (n_14), .B (n_25), .Y (n_31));
  CLKINVX1 g1063(.A (n_29), .Y (n_30));
  ADDHXL g1064(.A (n_21), .B (n_19), .CO (n_28), .S (n_29));
  OR2X1 g1065(.A (n_22), .B (n_23), .Y (n_27));
  ADDFX1 g1066(.A (n_6), .B (n_9), .CI (n_5), .CO (n_25), .S (n_26));
  AOI21XL g1067(.A0 (n_16), .A1 (n_15), .B0 (n_20), .Y (n_24));
  AND3XL g1068(.A (b[3]), .B (a[1]), .C (n_16), .Y (n_23));
  AND3XL g1069(.A (b[1]), .B (a[3]), .C (n_15), .Y (n_22));
  NOR2BXL g1070(.AN (n_15), .B (n_11), .Y (n_21));
  NOR2XL g1071(.A (n_16), .B (n_15), .Y (n_20));
  NOR2BXL g1072(.AN (n_16), .B (n_10), .Y (n_19));
  ADDHXL g1073(.A (n_2), .B (n_3), .CO (n_18), .S (product[0]));
  NAND3BXL g1074(.AN (n_1), .B (a[1]), .C (b[3]), .Y (n_16));
  NAND3BXL g1075(.AN (n_0), .B (b[1]), .C (a[3]), .Y (n_15));
  XNOR2X1 g1076(.A (n_0), .B (n_1), .Y (n_14));
  XNOR2X1 g1077(.A (n_8), .B (n_7), .Y (n_13));
  NOR2XL g1078(.A (n_7), .B (n_8), .Y (n_12));
  AOI22XL g1079(.A0 (a[2]), .A1 (b[1]), .B0 (a[3]), .B1 (b[0]), .Y
    (n_11));
  AOI22XL g1080(.A0 (a[0]), .A1 (b[3]), .B0 (a[1]), .B1 (b[2]), .Y
    (n_10));
  NAND2XL g1081(.A (b[0]), .B (a[1]), .Y (n_9));
  NAND2XL g1082(.A (b[3]), .B (a[3]), .Y (n_8));
  NAND2XL g1083(.A (b[1]), .B (a[1]), .Y (n_7));
  NAND2XL g1084(.A (b[1]), .B (a[0]), .Y (n_6));
  NAND2XL g1085(.A (b[3]), .B (a[2]), .Y (n_5));
  NAND2XL g1086(.A (b[2]), .B (a[3]), .Y (n_4));
  AND2X1 g1087(.A (b[2]), .B (a[2]), .Y (n_3));
  AND2X1 g1088(.A (b[0]), .B (a[0]), .Y (n_2));
  NAND2XL g1089(.A (a[0]), .B (b[2]), .Y (n_1));
  NAND2XL g1090(.A (b[0]), .B (a[2]), .Y (n_0));
endmodule

module vedic8_8(a, b, product);
  input [7:0] a, b;
  output [15:0] product;
  wire [7:0] a, b;
  wire [15:0] product;
  wire [7:0] p1;
```

```
  wire [7:0] p2;
  wire [7:0] p3;
  wire [15:0] temp1;
  wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
      UNCONNECTED3, UNCONNECTED4, UNCONNECTED5, UNCONNECTED6;
  wire UNCONNECTED7, UNCONNECTED8, UNCONNECTED9, UNCONNECTED10,
      UNCONNECTED11, UNCONNECTED12, UNCONNECTED13, UNCONNECTED14;
  wire UNCONNECTED15, UNCONNECTED16, UNCONNECTED17, carry, overflow,
      sum;
  assign product[0] = 1'b0;
  assign product[1] = 1'b0;
  assign product[2] = 1'b0;
  assign product[3] = 1'b0;
  assign product[4] = 1'b0;
  assign product[5] = 1'b0;
  assign product[6] = 1'b0;
  assign product[7] = 1'b0;
  assign product[8] = 1'b0;
  assign product[9] = 1'b0;
  assign product[10] = 1'b0;
  assign product[11] = 1'b0;
  assign product[12] = 1'b0;
  assign product[13] = 1'b0;
  assign product[14] = 1'b0;
  compressor6_3 c1(1'b0, p1[3], p2[3], 1'b0, p1[4], p2[4], sum, carry,
      overflow);
  csa_tree_add_83_52_group_59 csa_tree_add_83_52_groupi(.in_0
      ({overflow, carry, sum}), .in_1 ({1'b0, p3[6:0], 8'b00000000}),
      .in_2 ({5'b00000, p2[6:0], 4'b0000}), .in_3 ({1'b0,
      temp1[6:0]}), .in_4 ({5'b00000, p1[6:0], 4'b0000}), .out_0
      ({product[15], UNCONNECTED, UNCONNECTED0, UNCONNECTED1,
      UNCONNECTED2, UNCONNECTED3, UNCONNECTED4, UNCONNECTED5,
      UNCONNECTED6, UNCONNECTED7, UNCONNECTED8, UNCONNECTED9,
      UNCONNECTED10, UNCONNECTED11, UNCONNECTED12, UNCONNECTED13}));
  vedic4_4 vm1(a[3:0], b[3:0], {UNCONNECTED14, temp1[6:0]});
  vedic4_4_1 vm2(a[3:0], b[7:4], {UNCONNECTED15, p1[6:0]});
  vedic4_4_2 vm3(a[7:4], b[3:0], {UNCONNECTED16, p2[6:0]});
  vedic4_4_3 vm4(a[7:4], b[7:4], {UNCONNECTED17, p3[6:0]});
endmodule

module hybrid_mac(activation, weight, clk, reset, mac_result);
  input [7:0] activation, weight;
  input clk, reset;
  output [31:0] mac_result;
  wire [7:0] activation, weight;
  wire clk, reset;
  wire [31:0] mac_result;
  wire [31:0] reduction_result;
  wire [15:0] product;
  wire UNCONNECTED18, UNCONNECTED19, UNCONNECTED20, UNCONNECTED21,
      UNCONNECTED22, UNCONNECTED23, UNCONNECTED24, UNCONNECTED25;
  wire UNCONNECTED26, UNCONNECTED27, UNCONNECTED28, UNCONNECTED29,
      UNCONNECTED30, UNCONNECTED31, UNCONNECTED32, UNCONNECTED33;
  wire UNCONNECTED34, UNCONNECTED35, UNCONNECTED36, UNCONNECTED37,
```

```
    UNCONNECTED38, UNCONNECTED39, UNCONNECTED40;
  accumulator acc(.clk (clk), .reset (reset), .input_data
    ({reduction_result[31:8], 7'b0000000, product[15]}),
    .accumulated_result (mac_result));
  reduction_unit_with_6_3 hreduce(.A (16'b0000000000000000), .B
    (16'b0000000000000000), .C (16'b0000000000000000), .D
    (16'b0000000000000000), .E (16'b0000000000000000), .F
    (16'b0000000000000000), .G (16'b0000000000000000), .H
    (16'b0000000000000000), .Result ({reduction_result[31:8],
    UNCONNECTED18, UNCONNECTED19, UNCONNECTED20, UNCONNECTED21,
    UNCONNECTED22, UNCONNECTED23, UNCONNECTED24, UNCONNECTED25}));
  vedic8_8 vedic_mult(.a (activation), .b (weight), .product
    ({product[15], UNCONNECTED26, UNCONNECTED27, UNCONNECTED28,
    UNCONNECTED29, UNCONNECTED30, UNCONNECTED31, UNCONNECTED32,
    UNCONNECTED33, UNCONNECTED34, UNCONNECTED35, UNCONNECTED36,
    UNCONNECTED37, UNCONNECTED38, UNCONNECTED39, UNCONNECTED40}));
endmodule
```

APPENDIX-D

TIMING REPORT APPROXIMATE MAC: -

```
==============================================================
  Generated by:           Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:           Dec 09 2024  11:53:53 am
  Module:                 hybrid_mac
  Technology library:     slow
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
==============================================================

          Pin                Type       Fanout Load Slew Delay Arrival
                                        (fF) (ps)  (ps)   (ps)
------------------------------------------------------------------------------
(clock clk)                  launch                          0 R
acc
  accumulated_result_reg[0]/CK                        100          0 R
  accumulated_result_reg[0]/Q    DFFRHQX1               3  5.3   61  +371     371 F
  add_17_54/A[0]
    g1237/A                                                +0     371
    g1237/Y                    NAND2XL                 3  7.1  204  +112     483 R
    g1068/B                                                +0     483
    g1068/Y                    NAND2X1                 4  7.8  150  +142     625 F
    g1058/B                                                +0     625
    g1058/Y                    NOR2X1                  4 10.4  192  +181     806 R
    g1037/B                                                +0     806
    g1037/Y                    CLKAND2X2               2  6.6   59  +148     955 R
    g1267/C                                                +0     955
    g1267/Y                    NAND3BX1                4  7.6  211  +159    1114 F
    g1009/B                                                +0    1114
    g1009/Y                    NAND2XL                 1  2.2   90   +93    1207 R
    g1004/B                                                +0    1207
    g1004/Y                    CLKAND2X2              16 42.1  191  +238    1445 R
    g3/A1                                                  +0    1445
    g3/Y                       OA21XL                  1  2.7   68  +155    1600 R
    g1266/A1                                               +0    1600
    g1266/Y                    OAI21X1                 1  3.7  106   +81    1681 F
    g1248/B                                                +0    1681
    g1248/Y                    XOR2XL                  1  2.3   66  +171    1852 R
  add_17_54/Z[19]
  accumulated_result_reg[19]/D   DFFRHQX1                     +0    1852
  accumulated_result_reg[19]/CK  setup                    100  +134    1986 R
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(clock clk)                  capture                        2000 R
                           uncertainty                    -10    1990 R
------------------------------------------------------------------------------
Cost Group   : 'clk' (path_group 'clk')
Timing slack :       4ps
Start-point  : acc/accumulated_result_reg[0]/CK
End-point    : acc/accumulated_result_reg[19]/D
```

AREA REPORT APPROXIMATE MAC: -

```
==========================================================
  Generated by:          Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:          Dec 09 2024  11:53:53 am
  Module:                hybrid_mac
  Technology library:    slow
  Operating conditions:  slow (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
==========================================================

          Instance          Cells  Cell Area  Net Area  Total Area  Wireload
-------------------------------------------------------------------------------
hybrid_mac                    489       2834         0        2834    <none> (D)
  acc                         282       1728         0        1728    <none> (D)
    add_17_54                 249       1072         0        1072    <none> (D)
  vedic_mult                  207       1106         0        1106    <none> (D)
    csa_tree_add_85_52_groupi  35        292         0         292    <none> (D)
    vm4                         42        192         0         192    <none> (D)
    vm3                         42        192         0         192    <none> (D)
    vm2                         42        192         0         192    <none> (D)
    vm1                         42        192         0         192    <none> (D)
    c1                           4         45         0          45    <none> (D)

  (D) = wireload is default in technology library
```

POWER REPORT APPROXIMATE MAC: -

```
==========================================================
  Generated by:          Encounter(R) RTL Compiler v12.10-s012_1
  Generated on:          Dec 09 2024  11:53:53 am
  Module:                hybrid_mac
  Technology library:    slow
  Operating conditions:  slow (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
==========================================================

                          Leakage    Dynamic     Total
          Instance          Cells Power(nW)  Power(nW)  Power(nW)
-------------------------------------------------------------------
hybrid_mac                    489 13036.449 459399.536 472435.985
  acc                         282  8339.132 398451.696 406790.828
    add_17_54                 249  5115.911  60942.721  66058.632
  vedic_mult                  207  4697.316  37441.910  42139.227
    csa_tree_add_85_52_groupi  35  1364.032  12597.733  13961.765
    vm1                         42   749.964   5574.687   6324.651
    vm2                         42   749.964   5415.987   6165.951
    vm3                         42   749.964   5702.958   6452.922
    vm4                         42   749.964   5106.633   5856.597
    c1                           4   333.429   3043.912   3377.341
  hreduce                        0     0.000    630.180    630.180
```

CONNECTION REPORT APPROXIMATE MAC: -

```
#############################################################
# Generated by:      Cadence Encounter 13.10-p003_1
# OS:                Linux i686(Host ID cadence)
# Generated on:      Thu Dec  5 14:50:47 2024
# Design:            hybrid_mac
# Command:           verifyConnectivity -type all -error 1000 -warning 50
#############################################################
Verify Connectivity Report is created on Thu Dec  5 14:50:47 2024

Begin Summary
   Found no problems or warnings.
End Summary
```

CHECK PLACE REPORT APPROXIMATE MAC: -

```
#############################################################
# Generated by:      Cadence Encounter 13.10-p003_1
# OS:                Linux i686(Host ID cadence)
# Generated on:      Wed Dec  4 16:53:17 2024
# Design:            hybrid_mac
# Command:           checkPlace hybrid_mac.checkPlace
#############################################################

## No violations found ##

## Summary:
#############################################################
## Number of Placed Instances = 555
## Number of Unplaced Instances = 0
## Placement Density:70.21%(3371/4802)
```

GEOMETRY REPORT APPROXIMATE MAC: -

```
#############################################################
# Generated by:      Cadence Encounter 13.10-p003_1
# OS:                Linux i686(Host ID cadence)
# Generated on:      Thu Dec  5 14:50:29 2024
# Design:            hybrid_mac
# Command:           verifyGeometry
#############################################################

SPACING: Regular Via of Net reduction_result[24] & Regular Wire of Net reduction_result[24]  (
Metal2 )
Bounds : ( 18.925, 24.285 ) ( 19.070, 24.375 )
Actual:     0.09     Min:     0.14     Type: SameNet  SameNetGap

SPACING: Regular Via of Net reduction_result[25] & Regular Wire of Net reduction_result[25]  (
Metal2 )
Bounds : ( 11.095, 22.025 ) ( 11.240, 22.115 )
Actual:     0.09     Min:     0.14     Type: SameNet  SameNetGap

SPACING: Regular Via of Net reduction_result[26] & Regular Via of Net reduction_result[26]  (
Metal2 )
Bounds : ( 19.505, 11.525 ) ( 19.650, 11.615 )
Actual:     0.09     Min:     0.14     Type: SameNet  SameNetGap
```

SPACING: Regular Via of Net reduction_result[27] & Regular Wire of Net reduction_result[27] (Metal2)
Bounds : (20.375, 13.905) (20.520, 13.995)
Actual: 0.09 Min: 0.14 Type: SameNet SameNetGap

Begin Summary ...
Cells : 0
SameNet : 4
Wiring : 0
Antenna : 0
Short : 0
Overlap : 0
End Summary

Total Violations : 4 Viols.

NETLIST REPORT APPROXIMATE MAC: -

```
// Generated by Cadence Encounter(R) RTL Compiler v12.10-s012_1

// Verification Directory fv/hybrid_mac

module add_unsigned_24(A, B, Z);
  input [31:0] A, B;
  output [31:0] Z;
  wire [31:0] A, B;
  wire [31:0] Z;
  wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
  wire n_32, n_33, n_34, n_35, n_36, n_37, n_38, n_39;
  wire n_40, n_41, n_42, n_43, n_44, n_45, n_46, n_47;
  wire n_48, n_49, n_50, n_51, n_52, n_53, n_54, n_55;
  wire n_56, n_57, n_58, n_59, n_60, n_61, n_62, n_63;
  wire n_64, n_65, n_66, n_67, n_68, n_69, n_70, n_71;
  wire n_72, n_73, n_74, n_75, n_76, n_77, n_78, n_79;
  wire n_80, n_81, n_82, n_83, n_84, n_85, n_86, n_87;
  wire n_88, n_89, n_90, n_91, n_92, n_93, n_94, n_95;
  wire n_96, n_97, n_98, n_99, n_100, n_101, n_102, n_103;
  wire n_104, n_105, n_106, n_107, n_108, n_109, n_110, n_111;
  wire n_112, n_113, n_114, n_115, n_116, n_117, n_118, n_119;
  wire n_120, n_121, n_122, n_123, n_124, n_125, n_126, n_127;
  wire n_128, n_129, n_130, n_131, n_132, n_133, n_134, n_135;
  wire n_136, n_137, n_138, n_139, n_140, n_141, n_142, n_143;
  wire n_144, n_145, n_146, n_147, n_148, n_149, n_150, n_151;
  wire n_153, n_154, n_155, n_156, n_157, n_159, n_160, n_161;
  wire n_162, n_163, n_164, n_165, n_166, n_167, n_168, n_169;
  wire n_170, n_172, n_173, n_174, n_175, n_176, n_177, n_178;
  wire n_179, n_180, n_182, n_183, n_184, n_185, n_186, n_187;
  wire n_188, n_189, n_190, n_192, n_193, n_194, n_195, n_196;
  wire n_197, n_198, n_199, n_200, n_203, n_204, n_205, n_207;
  wire n_208, n_209, n_210, n_211, n_212, n_213, n_214, n_217;
  wire n_218, n_220, n_221, n_222, n_223, n_224, n_225, n_226;
```

```
wire n_227, n_228, n_229, n_230, n_231, n_232, n_233, n_234;
wire n_235, n_237, n_238, n_239, n_240, n_242, n_243, n_244;
wire n_245, n_246, n_247, n_248, n_249, n_250, n_251, n_309;
wire n_310, n_311;
OAI21X1 g968(.A0 (n_101), .A1 (n_222), .B0 (n_140), .Y (n_251));
OAI21X1 g969(.A0 (n_68), .A1 (n_222), .B0 (n_59), .Y (n_250));
OAI21X1 g970(.A0 (n_188), .A1 (n_222), .B0 (n_220), .Y (n_249));
OAI21X1 g971(.A0 (n_187), .A1 (n_222), .B0 (n_232), .Y (n_248));
OAI21X1 g972(.A0 (n_165), .A1 (n_222), .B0 (n_212), .Y (n_247));
OAI21X1 g973(.A0 (n_182), .A1 (n_222), .B0 (n_218), .Y (n_246));
OAI21X1 g974(.A0 (n_184), .A1 (n_222), .B0 (n_221), .Y (n_245));
OAI21X1 g975(.A0 (n_186), .A1 (n_222), .B0 (n_231), .Y (n_244));
OAI21X1 g976(.A0 (n_166), .A1 (n_222), .B0 (n_213), .Y (n_243));
OAI21X1 g977(.A0 (n_157), .A1 (n_222), .B0 (n_199), .Y (n_242));
OAI21X1 g979(.A0 (n_146), .A1 (n_222), .B0 (n_177), .Y (n_240));
OAI21X1 g980(.A0 (n_160), .A1 (n_222), .B0 (n_203), .Y (n_239));
OAI21X1 g981(.A0 (n_185), .A1 (n_222), .B0 (n_225), .Y (n_238));
OAI21X1 g989(.A0 (n_183), .A1 (n_222), .B0 (n_228), .Y (n_237));
OAI21X1 g990(.A0 (n_113), .A1 (n_223), .B0 (n_235), .Y (Z[16]));
NAND2XL g991(.A (n_113), .B (n_223), .Y (n_235));
AO21X1 g992(.A0 (n_111), .A1 (n_311), .B0 (n_141), .Y (n_234));
OAI21X1 g993(.A0 (n_74), .A1 (n_207), .B0 (n_84), .Y (n_233));
AOI21XL g994(.A0 (n_122), .A1 (n_204), .B0 (n_138), .Y (n_232));
OA21XL g995(.A0 (n_39), .A1 (n_203), .B0 (n_66), .Y (n_231));
OAI31XL g996(.A0 (n_154), .A1 (n_70), .A2 (n_207), .B0 (n_210), .Y
    (n_230));
OAI31XL g997(.A0 (n_154), .A1 (n_117), .A2 (n_207), .B0 (n_211), .Y
    (n_229));
AOI21XL g998(.A0 (n_147), .A1 (n_204), .B0 (n_174), .Y (n_228));
OAI31XL g999(.A0 (n_154), .A1 (n_149), .A2 (n_207), .B0 (n_198), .Y
    (n_227));
OAI31XL g1000(.A0 (n_110), .A1 (n_76), .A2 (n_207), .B0 (n_180), .Y
    (n_226));
AOI21XL g1001(.A0 (n_156), .A1 (n_204), .B0 (n_200), .Y (n_225));
OAI21X1 g1002(.A0 (n_154), .A1 (n_207), .B0 (n_175), .Y (n_224));
CLKINVX1 g1003(.A (n_222), .Y (n_223));
CLKAND2X2 g1004(.A (n_205), .B (n_217), .Y (n_222));
AOI221X1 g1005(.A0 (n_92), .A1 (n_174), .B0 (n_168), .B1 (n_204), .C0
    (n_143), .Y (n_221));
AOI21XL g1006(.A0 (n_161), .A1 (n_204), .B0 (n_209), .Y (n_220));
MXI2XL g1007(.A (n_207), .B (n_311), .S0 (n_124), .Y (Z[8]));
AOI21XL g1008(.A0 (n_144), .A1 (n_204), .B0 (n_179), .Y (n_218));
NAND2XL g1009(.A (n_167), .B (n_311), .Y (n_217));
OAI21XL g1010(.A0 (n_196), .A1 (n_85), .B0 (n_214), .Y (Z[5]));
OAI21XL g1011(.A0 (n_192), .A1 (n_131), .B0 (n_208), .Y (Z[7]));
NAND2XL g1013(.A (n_85), .B (n_196), .Y (n_214));
AOI21X1 g1014(.A0 (n_95), .A1 (n_176), .B0 (n_142), .Y (n_213));
OA21XL g1015(.A0 (n_65), .A1 (n_177), .B0 (n_37), .Y (n_212));
AOI2BB1XL g1016(.A0N (n_117), .A1N (n_175), .B0 (n_135), .Y (n_211));
OA21XL g1017(.A0 (n_70), .A1 (n_175), .B0 (n_64), .Y (n_210));
OAI21X1 g1018(.A0 (n_58), .A1 (n_173), .B0 (n_50), .Y (n_209));
NAND2XL g1019(.A (n_131), .B (n_192), .Y (n_208));
CLKINVX1 g1024(.A (n_311), .Y (n_207));
AOI211X1 g1026(.A0 (n_99), .A1 (n_135), .B0 (n_137), .C0 (n_197), .Y
```

```
    (n_205));
CLKINVX1 g1027(.A (n_204), .Y (n_203));
OAI21X1 g1028(.A0 (n_151), .A1 (n_177), .B0 (n_172), .Y (n_204));
OAI21X1 g1029(.A0 (n_87), .A1 (n_190), .B0 (n_194), .Y (Z[3]));
XOR2XL g1030(.A (n_90), .B (n_178), .Y (Z[4]));
OAI211X1 g1031(.A0 (n_150), .A1 (n_173), .B0 (n_34), .C0 (n_162), .Y
    (n_200));
AOI221X1 g1032(.A0 (n_31), .A1 (n_142), .B0 (n_155), .B1 (n_176), .C0
    (n_52), .Y (n_199));
AOI211X1 g1033(.A0 (n_56), .A1 (n_135), .B0 (n_44), .C0 (n_193), .Y
    (n_198));
NOR2XL g1034(.A (n_153), .B (n_175), .Y (n_197));
NAND2XL g1035(.A (n_40), .B (n_178), .Y (n_196));
CLKAND2X2 g1037(.A (n_24), .B (n_178), .Y (n_195));
NAND2XL g1038(.A (n_87), .B (n_190), .Y (n_194));
NOR2XL g1039(.A (n_149), .B (n_175), .Y (n_193));
NAND3X1 g1040(.A (n_29), .B (n_24), .C (n_178), .Y (n_192));
OAI21XL g1041(.A0 (n_170), .A1 (n_89), .B0 (n_189), .Y (Z[2]));
NAND2BX1 g1042(.AN (n_170), .B (n_71), .Y (n_190));
NAND2XL g1043(.A (n_170), .B (n_89), .Y (n_189));
NAND2XL g1044(.A (n_159), .B (n_161), .Y (n_188));
NAND2XL g1045(.A (n_122), .B (n_159), .Y (n_187));
NAND2BXL g1046(.AN (n_39), .B (n_159), .Y (n_186));
NAND2XL g1047(.A (n_159), .B (n_156), .Y (n_185));
NAND2XL g1048(.A (n_159), .B (n_168), .Y (n_184));
NAND2XL g1049(.A (n_147), .B (n_159), .Y (n_183));
NAND2XL g1050(.A (n_144), .B (n_159), .Y (n_182));
AOI21XL g1052(.A0 (n_75), .A1 (n_141), .B0 (n_78), .Y (n_180));
OAI2BB1X1 g1053(.A0N (n_49), .A1N (n_138), .B0 (n_60), .Y (n_179));
NOR2X1 g1058(.A (n_169), .B (n_170), .Y (n_178));
CLKINVX1 g1059(.A (n_177), .Y (n_176));
AOI21X1 g1060(.A0 (n_139), .A1 (n_104), .B0 (n_134), .Y (n_177));
AOI21X1 g1061(.A0 (n_141), .A1 (n_121), .B0 (n_136), .Y (n_175));
CLKINVX1 g1062(.A (n_174), .Y (n_173));
OAI211X1 g1063(.A0 (n_60), .A1 (n_38), .B0 (n_42), .C0 (n_164), .Y
    (n_174));
AOI21X1 g1064(.A0 (n_125), .A1 (n_142), .B0 (n_133), .Y (n_172));
OAI21X1 g1065(.A0 (n_132), .A1 (n_88), .B0 (n_163), .Y (Z[1]));
NAND2X1 g1068(.A (n_27), .B (n_132), .Y (n_170));
NAND2XL g1069(.A (n_26), .B (n_71), .Y (n_169));
AND2X1 g1070(.A (n_92), .B (n_147), .Y (n_168));
NOR2XL g1071(.A (n_153), .B (n_154), .Y (n_167));
NAND2XL g1072(.A (n_95), .B (n_145), .Y (n_166));
NAND2BXL g1073(.AN (n_65), .B (n_145), .Y (n_165));
NAND2XL g1074(.A (n_138), .B (n_112), .Y (n_164));
NAND2XL g1075(.A (n_132), .B (n_88), .Y (n_163));
NAND2BX1 g1076(.AN (n_82), .B (n_143), .Y (n_162));
NOR2XL g1077(.A (n_58), .B (n_148), .Y (n_161));
INVXL g1078(.A (n_159), .Y (n_160));
NOR2XL g1079(.A (n_151), .B (n_146), .Y (n_159));
NAND2XL g1083(.A (n_155), .B (n_145), .Y (n_157));
NOR2XL g1084(.A (n_150), .B (n_148), .Y (n_156));
AND2X1 g1085(.A (n_31), .B (n_95), .Y (n_155));
NAND2XL g1086(.A (n_111), .B (n_121), .Y (n_154));
```

```
NAND2XL g1087(.A (n_118), .B (n_99), .Y (n_153));
NAND2XL g1089(.A (n_95), .B (n_125), .Y (n_151));
NAND2BX1 g1090(.AN (n_82), .B (n_92), .Y (n_150));
NAND2XL g1091(.A (n_56), .B (n_118), .Y (n_149));
CLKINVX1 g1092(.A (n_148), .Y (n_147));
NAND2XL g1093(.A (n_122), .B (n_112), .Y (n_148));
CLKINVX1 g1094(.A (n_146), .Y (n_145));
NAND2XL g1095(.A (n_100), .B (n_104), .Y (n_146));
AND2X1 g1096(.A (n_49), .B (n_122), .Y (n_144));
OAI21X1 g1097(.A0 (n_50), .A1 (n_73), .B0 (n_67), .Y (n_143));
OAI21X1 g1098(.A0 (n_37), .A1 (n_63), .B0 (n_36), .Y (n_142));
OAI21X1 g1099(.A0 (n_84), .A1 (n_61), .B0 (n_32), .Y (n_141));
CLKINVX1 g1100(.A (n_139), .Y (n_140));
OAI21X1 g1101(.A0 (n_59), .A1 (n_81), .B0 (n_33), .Y (n_139));
OAI21X1 g1102(.A0 (n_66), .A1 (n_83), .B0 (n_57), .Y (n_138));
OAI21X1 g1103(.A0 (n_45), .A1 (n_77), .B0 (n_62), .Y (n_137));
OAI21X1 g1104(.A0 (n_79), .A1 (n_72), .B0 (n_47), .Y (n_136));
OAI21X1 g1105(.A0 (n_64), .A1 (n_51), .B0 (n_46), .Y (n_135));
OAI21X1 g1106(.A0 (n_35), .A1 (n_54), .B0 (n_69), .Y (n_134));
OAI21X1 g1107(.A0 (n_53), .A1 (n_80), .B0 (n_28), .Y (n_133));
CLKINVX1 g1110(.A (n_130), .Y (n_131));
ADDHX1 g1111(.A (A[7]), .B (B[0]), .CO (n_129), .S (n_130));
NAND2BX1 g1113(.AN (n_39), .B (n_66), .Y (n_128));
NAND2BX1 g1115(.AN (n_82), .B (n_34), .Y (n_127));
NAND2XL g1117(.A (n_53), .B (n_31), .Y (n_126));
NOR2XL g1118(.A (n_30), .B (n_80), .Y (n_125));
NOR2BX1 g1119(.AN (n_84), .B (n_74), .Y (n_124));
NAND2BX1 g1121(.AN (n_63), .B (n_36), .Y (n_123));
NOR2XL g1122(.A (n_39), .B (n_83), .Y (n_122));
NOR2XL g1123(.A (n_76), .B (n_72), .Y (n_121));
NAND2BX1 g1125(.AN (n_65), .B (n_37), .Y (n_120));
NOR2BX1 g1127(.AN (n_69), .B (n_54), .Y (n_119));
CLKINVX1 g1128(.A (n_118), .Y (n_117));
NOR2XL g1129(.A (n_70), .B (n_51), .Y (n_118));
NAND2BX1 g1131(.AN (n_41), .B (n_35), .Y (n_116));
NAND2BX1 g1133(.AN (n_81), .B (n_33), .Y (n_115));
NOR2BX1 g1135(.AN (n_47), .B (n_72), .Y (n_114));
NAND2BX1 g1136(.AN (n_68), .B (n_59), .Y (n_113));
NOR2XL g1137(.A (n_48), .B (n_38), .Y (n_112));
INVXL g1138(.A (n_111), .Y (n_110));
NOR2XL g1139(.A (n_74), .B (n_61), .Y (n_111));
NAND2BX1 g1141(.AN (n_58), .B (n_50), .Y (n_109));
NAND2BX1 g1143(.AN (n_51), .B (n_46), .Y (n_108));
NAND2XL g1145(.A (n_79), .B (n_75), .Y (n_107));
NAND2BX1 g1147(.AN (n_70), .B (n_64), .Y (n_106));
NAND2BX1 g1149(.AN (n_77), .B (n_62), .Y (n_105));
NOR2XL g1150(.A (n_41), .B (n_54), .Y (n_104));
NOR2BX1 g1152(.AN (n_67), .B (n_73), .Y (n_103));
NAND2BX1 g1154(.AN (n_80), .B (n_28), .Y (n_102));
CLKINVX1 g1155(.A (n_100), .Y (n_101));
NOR2XL g1156(.A (n_68), .B (n_81), .Y (n_100));
NOR2XL g1157(.A (n_55), .B (n_77), .Y (n_99));
NOR2BX1 g1159(.AN (n_42), .B (n_38), .Y (n_98));
NAND2XL g1161(.A (n_45), .B (n_56), .Y (n_97));
```

```
NAND2XL g1163(.A (n_60), .B (n_49), .Y (n_96));
NOR2XL g1164(.A (n_65), .B (n_63), .Y (n_95));
NAND2BX1 g1166(.AN (n_83), .B (n_57), .Y (n_94));
NAND2BX1 g1168(.AN (n_61), .B (n_32), .Y (n_93));
NOR2XL g1169(.A (n_58), .B (n_73), .Y (n_92));
OAI21X1 g1170(.A0 (A[6]), .A1 (B[0]), .B0 (n_29), .Y (n_91));
OAI21X1 g1171(.A0 (A[4]), .A1 (B[0]), .B0 (n_40), .Y (n_90));
OAI21X1 g1172(.A0 (A[2]), .A1 (B[0]), .B0 (n_71), .Y (n_89));
OA21X1 g1173(.A0 (A[1]), .A1 (B[0]), .B0 (n_27), .Y (n_88));
OAI21X1 g1174(.A0 (A[3]), .A1 (B[0]), .B0 (n_26), .Y (n_87));
AOI21X1 g1176(.A0 (B[31]), .A1 (A[31]), .B0 (n_43), .Y (n_86));
OAI21X1 g1177(.A0 (A[5]), .A1 (B[0]), .B0 (n_25), .Y (n_85));
NAND2XL g1179(.A (B[8]), .B (A[8]), .Y (n_84));
NOR2XL g1180(.A (B[25]), .B (A[25]), .Y (n_83));
NOR2XL g1181(.A (B[30]), .B (A[30]), .Y (n_82));
NOR2XL g1182(.A (B[17]), .B (A[17]), .Y (n_81));
NOR2XL g1183(.A (B[23]), .B (A[23]), .Y (n_80));
INVXL g1184(.A (n_79), .Y (n_78));
NAND2XL g1185(.A (B[10]), .B (A[10]), .Y (n_79));
NOR2XL g1186(.A (B[15]), .B (A[15]), .Y (n_77));
CLKINVX1 g1187(.A (n_76), .Y (n_75));
NOR2XL g1188(.A (B[10]), .B (A[10]), .Y (n_76));
NOR2XL g1189(.A (B[8]), .B (A[8]), .Y (n_74));
NOR2XL g1190(.A (B[29]), .B (A[29]), .Y (n_73));
NOR2XL g1191(.A (B[11]), .B (A[11]), .Y (n_72));
NAND2XL g1192(.A (A[2]), .B (B[0]), .Y (n_71));
NOR2XL g1193(.A (B[12]), .B (A[12]), .Y (n_70));
NAND2XL g1194(.A (B[19]), .B (A[19]), .Y (n_69));
NOR2XL g1195(.A (B[16]), .B (A[16]), .Y (n_68));
NAND2XL g1196(.A (B[29]), .B (A[29]), .Y (n_67));
NAND2XL g1197(.A (B[24]), .B (A[24]), .Y (n_66));
NOR2XL g1198(.A (B[20]), .B (A[20]), .Y (n_65));
NAND2XL g1199(.A (B[12]), .B (A[12]), .Y (n_64));
NOR2XL g1200(.A (B[21]), .B (A[21]), .Y (n_63));
NAND2XL g1201(.A (B[15]), .B (A[15]), .Y (n_62));
NOR2XL g1202(.A (B[9]), .B (A[9]), .Y (n_61));
NAND2XL g1203(.A (B[26]), .B (A[26]), .Y (n_60));
NAND2XL g1204(.A (B[16]), .B (A[16]), .Y (n_59));
NOR2XL g1205(.A (B[28]), .B (A[28]), .Y (n_58));
NAND2XL g1206(.A (B[25]), .B (A[25]), .Y (n_57));
CLKINVX1 g1207(.A (n_55), .Y (n_56));
NOR2XL g1208(.A (B[14]), .B (A[14]), .Y (n_55));
NOR2XL g1209(.A (B[19]), .B (A[19]), .Y (n_54));
INVXL g1210(.A (n_53), .Y (n_52));
NAND2XL g1211(.A (B[22]), .B (A[22]), .Y (n_53));
NOR2XL g1212(.A (B[13]), .B (A[13]), .Y (n_51));
NAND2XL g1213(.A (B[28]), .B (A[28]), .Y (n_50));
CLKINVX1 g1214(.A (n_48), .Y (n_49));
NOR2XL g1215(.A (B[26]), .B (A[26]), .Y (n_48));
NAND2XL g1216(.A (B[11]), .B (A[11]), .Y (n_47));
NAND2XL g1217(.A (B[13]), .B (A[13]), .Y (n_46));
INVXL g1218(.A (n_45), .Y (n_44));
NAND2XL g1219(.A (B[14]), .B (A[14]), .Y (n_45));
NOR2XL g1220(.A (B[31]), .B (A[31]), .Y (n_43));
```

```
  NAND2XL g1221(.A (B[27]), .B (A[27]), .Y (n_42));
  NOR2XL g1222(.A (B[18]), .B (A[18]), .Y (n_41));
  NAND2XL g1224(.A (A[4]), .B (B[0]), .Y (n_40));
  NOR2XL g1225(.A (B[24]), .B (A[24]), .Y (n_39));
  NOR2XL g1226(.A (B[27]), .B (A[27]), .Y (n_38));
  NAND2XL g1227(.A (B[20]), .B (A[20]), .Y (n_37));
  NAND2XL g1228(.A (B[21]), .B (A[21]), .Y (n_36));
  NAND2XL g1229(.A (B[18]), .B (A[18]), .Y (n_35));
  NAND2XL g1230(.A (B[30]), .B (A[30]), .Y (n_34));
  NAND2XL g1231(.A (B[17]), .B (A[17]), .Y (n_33));
  NAND2XL g1232(.A (B[9]), .B (A[9]), .Y (n_32));
  CLKINVX1 g1233(.A (n_30), .Y (n_31));
  NOR2XL g1234(.A (B[22]), .B (A[22]), .Y (n_30));
  NAND2XL g1235(.A (A[6]), .B (B[0]), .Y (n_29));
  NAND2XL g1236(.A (B[23]), .B (A[23]), .Y (n_28));
  NAND2XL g1237(.A (A[0]), .B (B[0]), .Y (n_132));
  NAND2XL g1238(.A (A[1]), .B (B[0]), .Y (n_27));
  NAND2XL g1239(.A (A[3]), .B (B[0]), .Y (n_26));
  NAND2XL g1240(.A (A[5]), .B (B[0]), .Y (n_25));
  AND2X1 g2(.A (n_40), .B (n_25), .Y (n_24));
  XOR2XL g1242(.A (n_91), .B (n_195), .Y (Z[6]));
  XNOR2XL g1243(.A (n_128), .B (n_239), .Y (Z[24]));
  XNOR2XL g1244(.A (n_127), .B (n_245), .Y (Z[30]));
  XNOR2XL g1245(.A (n_126), .B (n_243), .Y (Z[22]));
  XNOR2XL g1246(.A (n_123), .B (n_247), .Y (Z[21]));
  XNOR2XL g1247(.A (n_120), .B (n_240), .Y (Z[20]));
  XOR2XL g1248(.A (n_119), .B (n_310), .Y (Z[19]));
  XNOR2XL g1249(.A (n_116), .B (n_251), .Y (Z[18]));
  XNOR2XL g1250(.A (n_115), .B (n_250), .Y (Z[17]));
  XOR2XL g1251(.A (n_114), .B (n_226), .Y (Z[11]));
  XNOR2XL g1252(.A (n_109), .B (n_237), .Y (Z[28]));
  XNOR2XL g1253(.A (n_108), .B (n_230), .Y (Z[13]));
  XNOR2XL g1254(.A (n_107), .B (n_234), .Y (Z[10]));
  XNOR2XL g1255(.A (n_106), .B (n_224), .Y (Z[12]));
  XNOR2XL g1256(.A (n_105), .B (n_227), .Y (Z[15]));
  XOR2XL g1257(.A (n_103), .B (n_249), .Y (Z[29]));
  XNOR2XL g1258(.A (n_102), .B (n_242), .Y (Z[23]));
  XOR2XL g1259(.A (n_98), .B (n_246), .Y (Z[27]));
  XNOR2XL g1260(.A (n_97), .B (n_229), .Y (Z[14]));
  XNOR2XL g1261(.A (n_96), .B (n_248), .Y (Z[26]));
  XNOR2XL g1262(.A (n_94), .B (n_244), .Y (Z[25]));
  XNOR2XL g1263(.A (n_93), .B (n_233), .Y (Z[9]));
  XOR2XL g1264(.A (n_86), .B (n_238), .Y (Z[31]));
  XOR2XL g1265(.A (B[0]), .B (A[0]), .Y (Z[0]));
  OAI21X1 g1266(.A0 (n_41), .A1 (n_309), .B0 (n_35), .Y (n_310));
  OA21XL g3(.A0 (n_101), .A1 (n_222), .B0 (n_140), .Y (n_309));
  NAND3BX1 g1267(.AN (n_129), .B (n_29), .C (n_195), .Y (n_311));
endmodule

module accumulator(clk, reset, input_data, accumulated_result);
  input clk, reset;
  input [31:0] input_data;
  output [31:0] accumulated_result;
  wire clk, reset;
```

```
wire [31:0] input_data;
wire [31:0] accumulated_result;
wire n_0, n_11, n_22, n_26, n_37, n_48, n_59, n_64;
wire n_65, n_66, n_67, n_68, n_69, n_70, n_71, n_72;
wire n_73, n_74, n_75, n_76, n_77, n_78, n_79, n_81;
wire n_82, n_83, n_84, n_85, n_86, n_87, n_88, n_89;
wire n_91;
add_unsigned_24 add_17_54(.A (accumulated_result), .B
    ({input_data[31:8], 7'b0000000, input_data[0]}), .Z ({n_64,
    n_65, n_66, n_67, n_68, n_70, n_71, n_72, n_73, n_74, n_75,
    n_76, n_77, n_78, n_79, n_81, n_82, n_83, n_84, n_85, n_86,
    n_87, n_88, n_89, n_91, n_11, n_22, n_26, n_37, n_48, n_59,
    n_69}));
INVXL g3(.A (reset), .Y (n_0));
DFFRHQX1 \accumulated_result_reg[0] (.RN (n_0), .CK (clk), .D (n_69),
    .Q (accumulated_result[0]));
DFFRHQX1 \accumulated_result_reg[10] (.RN (n_0), .CK (clk), .D
    (n_87), .Q (accumulated_result[10]));
DFFRHQX1 \accumulated_result_reg[11] (.RN (n_0), .CK (clk), .D
    (n_86), .Q (accumulated_result[11]));
DFFRHQX1 \accumulated_result_reg[12] (.RN (n_0), .CK (clk), .D
    (n_85), .Q (accumulated_result[12]));
DFFRHQX1 \accumulated_result_reg[13] (.RN (n_0), .CK (clk), .D
    (n_84), .Q (accumulated_result[13]));
DFFRHQX1 \accumulated_result_reg[5] (.RN (n_0), .CK (clk), .D (n_22),
    .Q (accumulated_result[5]));
DFFRHQX1 \accumulated_result_reg[14] (.RN (n_0), .CK (clk), .D
    (n_83), .Q (accumulated_result[14]));
DFFRHQX1 \accumulated_result_reg[15] (.RN (n_0), .CK (clk), .D
    (n_82), .Q (accumulated_result[15]));
DFFRHQX1 \accumulated_result_reg[16] (.RN (n_0), .CK (clk), .D
    (n_81), .Q (accumulated_result[16]));
DFFRHQX1 \accumulated_result_reg[30] (.RN (n_0), .CK (clk), .D
    (n_65), .Q (accumulated_result[30]));
DFFRHQX1 \accumulated_result_reg[17] (.RN (n_0), .CK (clk), .D
    (n_79), .Q (accumulated_result[17]));
DFFRHQX1 \accumulated_result_reg[18] (.RN (n_0), .CK (clk), .D
    (n_78), .Q (accumulated_result[18]));
DFFRHQX1 \accumulated_result_reg[19] (.RN (n_0), .CK (clk), .D
    (n_77), .Q (accumulated_result[19]));
DFFRHQX1 \accumulated_result_reg[27] (.RN (n_0), .CK (clk), .D
    (n_68), .Q (accumulated_result[27]));
DFFRHQX1 \accumulated_result_reg[1] (.RN (n_0), .CK (clk), .D (n_59),
    .Q (accumulated_result[1]));
DFFRHQX1 \accumulated_result_reg[20] (.RN (n_0), .CK (clk), .D
    (n_76), .Q (accumulated_result[20]));
DFFRHQX1 \accumulated_result_reg[21] (.RN (n_0), .CK (clk), .D
    (n_75), .Q (accumulated_result[21]));
DFFRHQX1 \accumulated_result_reg[22] (.RN (n_0), .CK (clk), .D
    (n_74), .Q (accumulated_result[22]));
DFFRHQX1 \accumulated_result_reg[23] (.RN (n_0), .CK (clk), .D
    (n_73), .Q (accumulated_result[23]));
DFFRHQX1 \accumulated_result_reg[24] (.RN (n_0), .CK (clk), .D
    (n_72), .Q (accumulated_result[24]));
```

```
  DFFRHQX1 \accumulated_result_reg[25] (.RN (n_0), .CK (clk), .D
      (n_71), .Q (accumulated_result[25]));
  DFFRHQX1 \accumulated_result_reg[26] (.RN (n_0), .CK (clk), .D
      (n_70), .Q (accumulated_result[26]));
  DFFRHQX1 \accumulated_result_reg[28] (.RN (n_0), .CK (clk), .D
      (n_67), .Q (accumulated_result[28]));
  DFFRHQX1 \accumulated_result_reg[29] (.RN (n_0), .CK (clk), .D
      (n_66), .Q (accumulated_result[29]));
  DFFRHQX1 \accumulated_result_reg[2] (.RN (n_0), .CK (clk), .D (n_48),
      .Q (accumulated_result[2]));
  DFFRHQX1 \accumulated_result_reg[31] (.RN (n_0), .CK (clk), .D
      (n_64), .Q (accumulated_result[31]));
  DFFRHQX1 \accumulated_result_reg[3] (.RN (n_0), .CK (clk), .D (n_37),
      .Q (accumulated_result[3]));
  DFFRHQX1 \accumulated_result_reg[4] (.RN (n_0), .CK (clk), .D (n_26),
      .Q (accumulated_result[4]));
  DFFRHQX1 \accumulated_result_reg[6] (.RN (n_0), .CK (clk), .D (n_11),
      .Q (accumulated_result[6]));
  DFFRHQX1 \accumulated_result_reg[7] (.RN (n_0), .CK (clk), .D (n_91),
      .Q (accumulated_result[7]));
  DFFRHQX1 \accumulated_result_reg[8] (.RN (n_0), .CK (clk), .D (n_89),
      .Q (accumulated_result[8]));
  DFFRHQX1 \accumulated_result_reg[9] (.RN (n_0), .CK (clk), .D (n_88),
      .Q (accumulated_result[9]));
endmodule

module reduction_unit_with_6_3(A, B, C, D, E, F, G, H, Result);
  input [15:0] A, B, C, D, E, F, G, H;
  output [31:0] Result;
  wire [15:0] A, B, C, D, E, F, G, H;
  wire [31:0] Result;
  assign Result[0] = 1'b0;
  assign Result[1] = 1'b0;
  assign Result[2] = 1'b0;
  assign Result[3] = 1'b0;
  assign Result[4] = 1'b0;
  assign Result[5] = 1'b0;
  assign Result[6] = 1'b0;
  assign Result[7] = 1'b0;
endmodule

module compressor6_3(a, b, c, d, e, f, sum, carry, overflow);
  input a, b, c, d, e, f;
  output sum, carry, overflow;
  wire a, b, c, d, e, f;
  wire sum, carry, overflow;
  wire n_0, n_1, n_2, n_3;
  ADDHXL g62(.A (n_0), .B (n_2), .CO (overflow), .S (carry));
  CLKXOR2X1 g63(.A (n_1), .B (n_3), .Y (sum));
  ADDHXL g64(.A (e), .B (f), .CO (n_2), .S (n_3));
  ADDHXL g65(.A (b), .B (c), .CO (n_0), .S (n_1));
endmodule

module csa_tree_add_85_52_group_59(in_0, in_1, in_2, in_3, in_4, out_0);
```

```
input [2:0] in_0;
input [15:0] in_1, in_2, in_4;
input [7:0] in_3;
output [15:0] out_0;
wire [2:0] in_0;
wire [15:0] in_1, in_2, in_4;
wire [7:0] in_3;
wire [15:0] out_0;
wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
wire n_32, n_33, n_34, n_35, n_36, n_37, n_38, n_39;
wire n_40, n_41, n_42, n_43, n_44;
assign out_0[0] = 1'b0;
assign out_0[1] = 1'b0;
assign out_0[2] = 1'b0;
assign out_0[3] = 1'b0;
assign out_0[4] = 1'b0;
assign out_0[5] = 1'b0;
assign out_0[6] = 1'b0;
assign out_0[7] = 1'b0;
assign out_0[8] = 1'b0;
assign out_0[9] = 1'b0;
assign out_0[10] = 1'b0;
assign out_0[11] = 1'b0;
assign out_0[12] = 1'b0;
assign out_0[13] = 1'b0;
assign out_0[14] = 1'b0;
NOR2XL g699(.A (n_1), .B (n_44), .Y (out_0[15]));
AOI32XL g700(.A0 (n_12), .A1 (in_1[11]), .A2 (in_1[12]), .B0 (n_43),
    .B1 (in_1[12]), .Y (n_44));
OAI21XL g701(.A0 (n_41), .A1 (n_15), .B0 (n_42), .Y (n_43));
OAI2BB1XL g702(.A0N (n_15), .A1N (n_41), .B0 (n_23), .Y (n_42));
AOI22XL g703(.A0 (n_40), .A1 (n_25), .B0 (n_39), .B1 (n_24), .Y
    (n_41));
OR2X1 g704(.A (n_24), .B (n_39), .Y (n_40));
OAI2BB1XL g705(.A0N (n_26), .A1N (n_37), .B0 (n_38), .Y (n_39));
OAI211XL g706(.A0 (n_37), .A1 (n_26), .B0 (in_4[8]), .C0 (n_6), .Y
    (n_38));
OAI2BB1XL g707(.A0N (n_29), .A1N (n_16), .B0 (n_36), .Y (n_37));
OAI21XL g708(.A0 (n_16), .A1 (n_29), .B0 (n_35), .Y (n_36));
OAI2BB1XL g709(.A0N (n_30), .A1N (n_33), .B0 (n_34), .Y (n_35));
OAI21XL g710(.A0 (n_33), .A1 (n_30), .B0 (n_20), .Y (n_34));
OAI21XL g711(.A0 (n_31), .A1 (n_22), .B0 (n_32), .Y (n_33));
OAI2BB1XL g712(.A0N (n_22), .A1N (n_31), .B0 (n_27), .Y (n_32));
AOI32XL g713(.A0 (n_28), .A1 (n_8), .A2 (in_3[4]), .B0 (n_19), .B1
    (n_28), .Y (n_31));
ADDFX1 g714(.A (in_2[7]), .B (n_3), .CI (in_4[7]), .CO (n_29), .S
    (n_30));
ADDFX1 g715(.A (n_10), .B (n_7), .CI (in_3[5]), .CO (n_27), .S
    (n_28));
ADDFX1 g716(.A (in_4[9]), .B (n_5), .CI (n_14), .CO (n_25), .S
    (n_26));
```

```
  ADDFX1 g717(.A (n_11), .B (in_4[10]), .CI (n_13), .CO (n_23), .S
      (n_24));
  CLKINVX1 g718(.A (n_21), .Y (n_22));
  ADDFX1 g719(.A (n_4), .B (n_9), .CI (in_3[6]), .CO (n_20), .S (n_21));
  NOR3XL g720(.A (n_18), .B (n_0), .C (n_17), .Y (n_19));
  OAI21XL g721(.A0 (in_3[4]), .A1 (n_8), .B0 (in_3[3]), .Y (n_18));
  AOI222XL g722(.A0 (in_3[0]), .A1 (n_2), .B0 (in_3[1]), .B1 (in_0[1]),
      .C0 (in_0[2]), .C1 (in_3[2]), .Y (n_17));
  CLKXOR2X1 g723(.A (in_4[8]), .B (n_6), .Y (n_16));
  NOR2XL g724(.A (in_1[11]), .B (n_12), .Y (n_15));
  ADDHXL g725(.A (in_1[9]), .B (in_2[9]), .CO (n_13), .S (n_14));
  ADDHXL g726(.A (in_1[10]), .B (in_2[10]), .CO (n_12), .S (n_11));
  ADDHXL g727(.A (in_4[5]), .B (in_2[5]), .CO (n_9), .S (n_10));
  ADDHXL g728(.A (in_4[4]), .B (in_2[4]), .CO (n_7), .S (n_8));
  ADDHXL g729(.A (in_1[8]), .B (in_2[8]), .CO (n_5), .S (n_6));
  ADDHXL g730(.A (in_4[6]), .B (in_2[6]), .CO (n_3), .S (n_4));
  OA21X1 g731(.A0 (in_3[1]), .A1 (in_0[1]), .B0 (in_0[0]), .Y (n_2));
  NAND2XL g732(.A (in_1[14]), .B (in_1[13]), .Y (n_1));
  NOR2XL g733(.A (in_3[2]), .B (in_0[2]), .Y (n_0));
endmodule

module vedic4_4(A, B, result);
  input [3:0] A, B;
  output [7:0] result;
  wire [3:0] A, B;
  wire [7:0] result;
  wire n_0, n_1, n_2, n_3, n_5, n_6, n_7, n_8;
  wire n_9, n_10, n_11, n_13, n_14, n_15, n_16, n_17;
  wire n_18, n_19, n_20, n_21, n_22, n_23, n_24, n_25;
  wire n_27, n_28, n_29, n_31, n_33, n_34, n_35, n_36;
  wire n_37, n_38, n_39, n_42, n_54;
  assign result[7] = 1'b0;
  OAI21XL g848(.A0 (n_38), .A1 (n_29), .B0 (n_42), .Y (result[5]));
  NAND2XL g849(.A (n_29), .B (n_38), .Y (n_42));
  OAI21XL g850(.A0 (n_37), .A1 (n_28), .B0 (n_27), .Y (result[6]));
  OAI21XL g851(.A0 (n_36), .A1 (n_33), .B0 (n_39), .Y (result[4]));
  NAND2XL g852(.A (n_33), .B (n_36), .Y (n_39));
  NAND2XL g853(.A (n_34), .B (n_37), .Y (n_38));
  NAND2XL g854(.A (n_35), .B (n_33), .Y (n_37));
  NAND2XL g855(.A (n_35), .B (n_34), .Y (n_36));
  NAND2BXL g856(.AN (n_31), .B (n_20), .Y (n_35));
  NAND2BXL g857(.AN (n_20), .B (n_31), .Y (n_34));
  ADDFX1 g858(.A (n_23), .B (n_11), .CI (n_24), .CO (n_33), .S
      (result[3]));
  NAND2XL g859(.A (n_18), .B (n_54), .Y (n_31));
  NAND2BXL g861(.AN (n_28), .B (n_27), .Y (n_29));
  NOR2XL g862(.A (n_8), .B (n_25), .Y (n_28));
  NAND2XL g863(.A (n_8), .B (n_25), .Y (n_27));
  AOI21XL g864(.A0 (n_2), .A1 (n_19), .B0 (n_23), .Y (result[2]));
  OAI21XL g865(.A0 (n_17), .A1 (n_16), .B0 (n_10), .Y (n_25));
  OAI21XL g866(.A0 (n_21), .A1 (n_16), .B0 (n_22), .Y (n_24));
  NOR2XL g867(.A (n_2), .B (n_19), .Y (n_23));
  NAND2XL g868(.A (n_16), .B (n_21), .Y (n_22));
  NOR2BXL g869(.AN (n_18), .B (n_17), .Y (n_21));
```

```
  NAND3BXL g870(.AN (n_8), .B (B[3]), .C (A[1]), .Y (n_20));
  OAI21XL g871(.A0 (n_14), .A1 (n_9), .B0 (n_16), .Y (n_19));
  NAND2XL g872(.A (n_13), .B (n_15), .Y (n_18));
  NOR2XL g873(.A (n_13), .B (n_15), .Y (n_17));
  NAND2XL g874(.A (n_9), .B (n_14), .Y (n_16));
  NOR2BXL g875(.AN (n_10), .B (n_7), .Y (n_15));
  ADDHXL g876(.A (n_1), .B (n_3), .CO (n_13), .S (n_14));
  NOR2XL g877(.A (n_6), .B (n_9), .Y (result[1]));
  AOI21XL g878(.A0 (n_0), .A1 (n_5), .B0 (n_8), .Y (n_11));
  NAND3XL g879(.A (A[3]), .B (B[1]), .C (n_1), .Y (n_10));
  AND2X1 g880(.A (result[0]), .B (n_3), .Y (n_9));
  NOR2XL g881(.A (n_0), .B (n_5), .Y (n_8));
  AOI22XL g882(.A0 (A[2]), .A1 (B[1]), .B0 (A[3]), .B1 (B[0]), .Y
       (n_7));
  AOI22XL g883(.A0 (A[1]), .A1 (B[0]), .B0 (A[0]), .B1 (B[1]), .Y
       (n_6));
  NAND2XL g884(.A (B[3]), .B (A[0]), .Y (n_5));
  AND2X1 g885(.A (B[0]), .B (A[0]), .Y (result[0]));
  AND2X1 g886(.A (B[1]), .B (A[1]), .Y (n_3));
  NAND2XL g887(.A (B[2]), .B (A[0]), .Y (n_2));
  AND2X1 g888(.A (A[2]), .B (B[0]), .Y (n_1));
  NAND2XL g889(.A (B[2]), .B (A[1]), .Y (n_0));
  NAND3BXL g2(.AN (n_25), .B (A[3]), .C (B[1]), .Y (n_54));
endmodule

module vedic4_4_1(A, B, result);
  input [3:0] A, B;
  output [7:0] result;
  wire [3:0] A, B;
  wire [7:0] result;
  wire n_0, n_1, n_2, n_3, n_5, n_6, n_7, n_8;
  wire n_9, n_10, n_11, n_13, n_14, n_15, n_16, n_17;
  wire n_18, n_19, n_20, n_21, n_22, n_23, n_24, n_25;
  wire n_27, n_28, n_29, n_31, n_33, n_34, n_35, n_36;
  wire n_37, n_38, n_39, n_42, n_54;
  assign result[7] = 1'b0;
  OAI21XL g848(.A0 (n_38), .A1 (n_29), .B0 (n_42), .Y (result[5]));
  NAND2XL g849(.A (n_29), .B (n_38), .Y (n_42));
  OAI21XL g850(.A0 (n_37), .A1 (n_28), .B0 (n_27), .Y (result[6]));
  OAI21XL g851(.A0 (n_36), .A1 (n_33), .B0 (n_39), .Y (result[4]));
  NAND2XL g852(.A (n_33), .B (n_36), .Y (n_39));
  NAND2XL g853(.A (n_34), .B (n_37), .Y (n_38));
  NAND2XL g854(.A (n_35), .B (n_33), .Y (n_37));
  NAND2XL g855(.A (n_35), .B (n_34), .Y (n_36));
  NAND2BXL g856(.AN (n_31), .B (n_20), .Y (n_35));
  NAND2BXL g857(.AN (n_20), .B (n_31), .Y (n_34));
  ADDFX1 g858(.A (n_23), .B (n_11), .CI (n_24), .CO (n_33), .S
       (result[3]));
  NAND2XL g859(.A (n_18), .B (n_54), .Y (n_31));
  NAND2BXL g861(.AN (n_28), .B (n_27), .Y (n_29));
  NOR2XL g862(.A (n_8), .B (n_25), .Y (n_28));
  NAND2XL g863(.A (n_8), .B (n_25), .Y (n_27));
  AOI21XL g864(.A0 (n_2), .A1 (n_19), .B0 (n_23), .Y (result[2]));
  OAI21XL g865(.A0 (n_17), .A1 (n_16), .B0 (n_10), .Y (n_25));
```

```
  OAI21XL g866(.A0 (n_21), .A1 (n_16), .B0 (n_22), .Y (n_24));
  NOR2XL g867(.A (n_2), .B (n_19), .Y (n_23));
  NAND2XL g868(.A (n_16), .B (n_21), .Y (n_22));
  NOR2BXL g869(.AN (n_18), .B (n_17), .Y (n_21));
  NAND3BXL g870(.AN (n_8), .B (B[3]), .C (A[1]), .Y (n_20));
  OAI21XL g871(.A0 (n_14), .A1 (n_9), .B0 (n_16), .Y (n_19));
  NAND2XL g872(.A (n_13), .B (n_15), .Y (n_18));
  NOR2XL g873(.A (n_13), .B (n_15), .Y (n_17));
  NAND2XL g874(.A (n_9), .B (n_14), .Y (n_16));
  NOR2BXL g875(.AN (n_10), .B (n_7), .Y (n_15));
  ADDHXL g876(.A (n_1), .B (n_3), .CO (n_13), .S (n_14));
  NOR2XL g877(.A (n_6), .B (n_9), .Y (result[1]));
  AOI21XL g878(.A0 (n_0), .A1 (n_5), .B0 (n_8), .Y (n_11));
  NAND3XL g879(.A (A[3]), .B (B[1]), .C (n_1), .Y (n_10));
  AND2X1 g880(.A (result[0]), .B (n_3), .Y (n_9));
  NOR2XL g881(.A (n_0), .B (n_5), .Y (n_8));
  AOI22XL g882(.A0 (A[2]), .A1 (B[1]), .B0 (A[3]), .B1 (B[0]), .Y
      (n_7));
  AOI22XL g883(.A0 (A[1]), .A1 (B[0]), .B0 (A[0]), .B1 (B[1]), .Y
      (n_6));
  NAND2XL g884(.A (B[3]), .B (A[0]), .Y (n_5));
  AND2X1 g885(.A (B[0]), .B (A[0]), .Y (result[0]));
  AND2X1 g886(.A (B[1]), .B (A[1]), .Y (n_3));
  NAND2XL g887(.A (B[2]), .B (A[0]), .Y (n_2));
  AND2X1 g888(.A (A[2]), .B (B[0]), .Y (n_1));
  NAND2XL g889(.A (B[2]), .B (A[1]), .Y (n_0));
  NAND3BXL g2(.AN (n_25), .B (A[3]), .C (B[1]), .Y (n_54));
endmodule

module vedic4_4_2(A, B, result);
  input [3:0] A, B;
  output [7:0] result;
  wire [3:0] A, B;
  wire [7:0] result;
  wire n_0, n_1, n_2, n_3, n_5, n_6, n_7, n_8;
  wire n_9, n_10, n_11, n_13, n_14, n_15, n_16, n_17;
  wire n_18, n_19, n_20, n_21, n_22, n_23, n_24, n_25;
  wire n_27, n_28, n_29, n_31, n_33, n_34, n_35, n_36;
  wire n_37, n_38, n_39, n_42, n_54;
  assign result[7] = 1'b0;
  OAI21XL g848(.A0 (n_38), .A1 (n_29), .B0 (n_42), .Y (result[5]));
  NAND2XL g849(.A (n_29), .B (n_38), .Y (n_42));
  OAI21XL g850(.A0 (n_37), .A1 (n_28), .B0 (n_27), .Y (result[6]));
  OAI21XL g851(.A0 (n_36), .A1 (n_33), .B0 (n_39), .Y (result[4]));
  NAND2XL g852(.A (n_33), .B (n_36), .Y (n_39));
  NAND2XL g853(.A (n_34), .B (n_37), .Y (n_38));
  NAND2XL g854(.A (n_35), .B (n_33), .Y (n_37));
  NAND2XL g855(.A (n_35), .B (n_34), .Y (n_36));
  NAND2BXL g856(.AN (n_31), .B (n_20), .Y (n_35));
  NAND2BXL g857(.AN (n_20), .B (n_31), .Y (n_34));
  ADDFX1 g858(.A (n_23), .B (n_11), .CI (n_24), .CO (n_33), .S
      (result[3]));
  NAND2XL g859(.A (n_18), .B (n_54), .Y (n_31));
  NAND2BXL g861(.AN (n_28), .B (n_27), .Y (n_29));
```

```
  NOR2XL g862(.A (n_8), .B (n_25), .Y (n_28));
  NAND2XL g863(.A (n_8), .B (n_25), .Y (n_27));
  AOI21XL g864(.A0 (n_2), .A1 (n_19), .B0 (n_23), .Y (result[2]));
  OAI21XL g865(.A0 (n_17), .A1 (n_16), .B0 (n_10), .Y (n_25));
  OAI21XL g866(.A0 (n_21), .A1 (n_16), .B0 (n_22), .Y (n_24));
  NOR2XL g867(.A (n_2), .B (n_19), .Y (n_23));
  NAND2XL g868(.A (n_16), .B (n_21), .Y (n_22));
  NOR2BXL g869(.AN (n_18), .B (n_17), .Y (n_21));
  NAND3BXL g870(.AN (n_8), .B (B[3]), .C (A[1]), .Y (n_20));
  OAI21XL g871(.A0 (n_14), .A1 (n_9), .B0 (n_16), .Y (n_19));
  NAND2XL g872(.A (n_13), .B (n_15), .Y (n_18));
  NOR2XL g873(.A (n_13), .B (n_15), .Y (n_17));
  NAND2XL g874(.A (n_9), .B (n_14), .Y (n_16));
  NOR2BXL g875(.AN (n_10), .B (n_7), .Y (n_15));
  ADDHXL g876(.A (n_1), .B (n_3), .CO (n_13), .S (n_14));
  NOR2XL g877(.A (n_6), .B (n_9), .Y (result[1]));
  AOI21XL g878(.A0 (n_0), .A1 (n_5), .B0 (n_8), .Y (n_11));
  NAND3XL g879(.A (A[3]), .B (B[1]), .C (n_1), .Y (n_10));
  AND2X1 g880(.A (result[0]), .B (n_3), .Y (n_9));
  NOR2XL g881(.A (n_0), .B (n_5), .Y (n_8));
  AOI22XL g882(.A0 (A[2]), .A1 (B[1]), .B0 (A[3]), .B1 (B[0]), .Y
      (n_7));
  AOI22XL g883(.A0 (A[1]), .A1 (B[0]), .B0 (A[0]), .B1 (B[1]), .Y
      (n_6));
  NAND2XL g884(.A (B[3]), .B (A[0]), .Y (n_5));
  AND2X1 g885(.A (B[0]), .B (A[0]), .Y (result[0]));
  AND2X1 g886(.A (B[1]), .B (A[1]), .Y (n_3));
  NAND2XL g887(.A (B[2]), .B (A[0]), .Y (n_2));
  AND2X1 g888(.A (A[2]), .B (B[0]), .Y (n_1));
  NAND2XL g889(.A (B[2]), .B (A[1]), .Y (n_0));
  NAND3BXL g2(.AN (n_25), .B (A[3]), .C (B[1]), .Y (n_54));
endmodule

module vedic4_4_3(A, B, result);
  input [3:0] A, B;
  output [7:0] result;
  wire [3:0] A, B;
  wire [7:0] result;
  wire n_0, n_1, n_2, n_3, n_5, n_6, n_7, n_8;
  wire n_9, n_10, n_11, n_13, n_14, n_15, n_16, n_17;
  wire n_18, n_19, n_20, n_21, n_22, n_23, n_24, n_25;
  wire n_27, n_28, n_29, n_31, n_33, n_34, n_35, n_36;
  wire n_37, n_38, n_39, n_42, n_54;
  assign result[7] = 1'b0;
  OAI21XL g848(.A0 (n_38), .A1 (n_29), .B0 (n_42), .Y (result[5]));
  NAND2XL g849(.A (n_29), .B (n_38), .Y (n_42));
  OAI21XL g850(.A0 (n_37), .A1 (n_28), .B0 (n_27), .Y (result[6]));
  OAI21XL g851(.A0 (n_36), .A1 (n_33), .B0 (n_39), .Y (result[4]));
  NAND2XL g852(.A (n_33), .B (n_36), .Y (n_39));
  NAND2XL g853(.A (n_34), .B (n_37), .Y (n_38));
  NAND2XL g854(.A (n_35), .B (n_33), .Y (n_37));
  NAND2XL g855(.A (n_35), .B (n_34), .Y (n_36));
  NAND2BXL g856(.AN (n_31), .B (n_20), .Y (n_35));
  NAND2BXL g857(.AN (n_20), .B (n_31), .Y (n_34));
```

```
  ADDFX1 g858(.A (n_23), .B (n_11), .CI (n_24), .CO (n_33), .S
      (result[3]));
  NAND2XL g859(.A (n_18), .B (n_54), .Y (n_31));
  NAND2BXL g861(.AN (n_28), .B (n_27), .Y (n_29));
  NOR2XL g862(.A (n_8), .B (n_25), .Y (n_28));
  NAND2XL g863(.A (n_8), .B (n_25), .Y (n_27));
  AOI21XL g864(.A0 (n_2), .A1 (n_19), .B0 (n_23), .Y (result[2]));
  OAI21XL g865(.A0 (n_17), .A1 (n_16), .B0 (n_10), .Y (n_25));
  OAI21XL g866(.A0 (n_21), .A1 (n_16), .B0 (n_22), .Y (n_24));
  NOR2XL g867(.A (n_2), .B (n_19), .Y (n_23));
  NAND2XL g868(.A (n_16), .B (n_21), .Y (n_22));
  NOR2BXL g869(.AN (n_18), .B (n_17), .Y (n_21));
  NAND3BXL g870(.AN (n_8), .B (B[3]), .C (A[1]), .Y (n_20));
  OAI21XL g871(.A0 (n_14), .A1 (n_9), .B0 (n_16), .Y (n_19));
  NAND2XL g872(.A (n_13), .B (n_15), .Y (n_18));
  NOR2XL g873(.A (n_13), .B (n_15), .Y (n_17));
  NAND2XL g874(.A (n_9), .B (n_14), .Y (n_16));
  NOR2BXL g875(.AN (n_10), .B (n_7), .Y (n_15));
  ADDHXL g876(.A (n_1), .B (n_3), .CO (n_13), .S (n_14));
  NOR2XL g877(.A (n_6), .B (n_9), .Y (result[1]));
  AOI21XL g878(.A0 (n_0), .A1 (n_5), .B0 (n_8), .Y (n_11));
  NAND3XL g879(.A (A[3]), .B (B[1]), .C (n_1), .Y (n_10));
  AND2X1 g880(.A (result[0]), .B (n_3), .Y (n_9));
  NOR2XL g881(.A (n_0), .B (n_5), .Y (n_8));
  AOI22XL g882(.A0 (A[2]), .A1 (B[1]), .B0 (A[3]), .B1 (B[0]), .Y
      (n_7));
  AOI22XL g883(.A0 (A[1]), .A1 (B[0]), .B0 (A[0]), .B1 (B[1]), .Y
      (n_6));
  NAND2XL g884(.A (B[3]), .B (A[0]), .Y (n_5));
  AND2X1 g885(.A (B[0]), .B (A[0]), .Y (result[0]));
  AND2X1 g886(.A (B[1]), .B (A[1]), .Y (n_3));
  NAND2XL g887(.A (B[2]), .B (A[0]), .Y (n_2));
  AND2X1 g888(.A (A[2]), .B (B[0]), .Y (n_1));
  NAND2XL g889(.A (B[2]), .B (A[1]), .Y (n_0));
  NAND3BXL g2(.AN (n_25), .B (A[3]), .C (B[1]), .Y (n_54));
endmodule

module vedic8_8(a, b, product);
  input [7:0] a, b;
  output [15:0] product;
  wire [7:0] a, b;
  wire [15:0] product;
  wire [7:0] p1;
  wire [7:0] p2;
  wire [7:0] p3;
  wire [15:0] temp1;
  wire UNCONNECTED, UNCONNECTED0, UNCONNECTED1, UNCONNECTED2,
      UNCONNECTED3, UNCONNECTED4, UNCONNECTED5, UNCONNECTED6;
  wire UNCONNECTED7, UNCONNECTED8, UNCONNECTED9, UNCONNECTED10,
      UNCONNECTED11, UNCONNECTED12, UNCONNECTED13, UNCONNECTED14;
  wire UNCONNECTED15, UNCONNECTED16, UNCONNECTED17, carry, overflow,
      sum;
  assign product[0] = 1'b0;
  assign product[1] = 1'b0;
```

```
  assign product[2] = 1'b0;
  assign product[3] = 1'b0;
  assign product[4] = 1'b0;
  assign product[5] = 1'b0;
  assign product[6] = 1'b0;
  assign product[7] = 1'b0;
  assign product[8] = 1'b0;
  assign product[9] = 1'b0;
  assign product[10] = 1'b0;
  assign product[11] = 1'b0;
  assign product[12] = 1'b0;
  assign product[13] = 1'b0;
  assign product[14] = 1'b0;
  compressor6_3 c1(1'b0, p1[3], p2[3], 1'b0, p1[4], p2[4], sum, carry,
      overflow);
  csa_tree_add_85_52_group_59 csa_tree_add_85_52_groupi(.in_0
      ({overflow, carry, sum}), .in_1 ({1'b0, p3[6:0], 8'b00000000}),
      .in_2 ({5'b00000, p2[6:0], 4'b0000}), .in_3 ({1'b0,
      temp1[6:0]}), .in_4 ({5'b00000, p1[6:0], 4'b0000}), .out_0
      ({product[15], UNCONNECTED, UNCONNECTED0, UNCONNECTED1,
      UNCONNECTED2, UNCONNECTED3, UNCONNECTED4, UNCONNECTED5,
      UNCONNECTED6, UNCONNECTED7, UNCONNECTED8, UNCONNECTED9,
      UNCONNECTED10, UNCONNECTED11, UNCONNECTED12, UNCONNECTED13}));
  vedic4_4 vm1(a[3:0], b[3:0], {UNCONNECTED14, temp1[6:0]});
  vedic4_4_1 vm2(a[3:0], b[7:4], {UNCONNECTED15, p1[6:0]});
  vedic4_4_2 vm3(a[7:4], b[3:0], {UNCONNECTED16, p2[6:0]});
  vedic4_4_3 vm4(a[7:4], b[7:4], {UNCONNECTED17, p3[6:0]});
endmodule

module hybrid_mac(activation, weight, clk, reset, mac_result);
  input [7:0] activation, weight;
  input clk, reset;
  output [31:0] mac_result;
  wire [7:0] activation, weight;
  wire clk, reset;
  wire [31:0] mac_result;
  wire [31:0] reduction_result;
  wire [15:0] product;
  wire UNCONNECTED18, UNCONNECTED19, UNCONNECTED20, UNCONNECTED21,
      UNCONNECTED22, UNCONNECTED23, UNCONNECTED24, UNCONNECTED25;
  wire UNCONNECTED26, UNCONNECTED27, UNCONNECTED28, UNCONNECTED29,
      UNCONNECTED30, UNCONNECTED31, UNCONNECTED32, UNCONNECTED33;
  wire UNCONNECTED34, UNCONNECTED35, UNCONNECTED36, UNCONNECTED37,
      UNCONNECTED38, UNCONNECTED39, UNCONNECTED40;
  accumulator acc(.clk (clk), .reset (reset), .input_data
      ({reduction_result[31:8], 7'b0000000, product[15]}),
      .accumulated_result (mac_result));
  reduction_unit_with_6_3 hreduce(.A (16'b0000000000000000), .B
      (16'b0000000000000000), .C (16'b0000000000000000), .D
      (16'b0000000000000000), .E (16'b0000000000000000), .F
      (16'b0000000000000000), .G (16'b0000000000000000), .H
      (16'b0000000000000000), .Result ({reduction_result[31:8],
      UNCONNECTED18, UNCONNECTED19, UNCONNECTED20, UNCONNECTED21,
      UNCONNECTED22, UNCONNECTED23, UNCONNECTED24, UNCONNECTED25}));
```

```
  vedic8_8 vedic_mult(.a (activation), .b (weight), .product
      ({product[15], UNCONNECTED26, UNCONNECTED27, UNCONNECTED28,
      UNCONNECTED29, UNCONNECTED30, UNCONNECTED31, UNCONNECTED32,
      UNCONNECTED33, UNCONNECTED34, UNCONNECTED35, UNCONNECTED36,
      UNCONNECTED37, UNCONNECTED38, UNCONNECTED39, UNCONNECTED40}));
endmodule
```

ASSESSMENT

Internal:

SL NO	RUBRICS	FULL MARK	MARKS OBTAINED	REMARKS
1	Understanding the relevance, scope and dimension of the project	10		
2	Methodology	10		
3	Quality of Analysis and Results	10		
4	Interpretations and Conclusions	10		
5	Report	10		
	Total	**50**		

Date: **Signature of the Faculty**