EEE6323:LOW-POWER NEURAL NETWORK WITH FINE-GRAIN SUPPLY AND OPERAND GATING

Group 8

Muhammad Ali
muhammad.ali@ufl.edu

Hunter Wickman
hwickman@ufl.edu

Stephen Singh
singhstephen@ufl.com

Deep Tandel
dtandel@ufl.edu

***Abstract*—The purpose of this report is to implement transistor-level low-power techniques on a neural network and analyze the resulting benefits and drawbacks. We employ voltage scaling and operand gating techniques to reduce power consumption. Voltage scaling involves lowering the supply voltage, affecting speed and delay, while operand gating selectively turns off parts of the data path to decrease switching power. We assess the effectiveness of these techniques in reducing both switching and leakage power. Additionally, we describe the construction of convolutional neural network (CNN) architectures and their synthesis in Verilog for power consumption analysis.**

I. Introduction

In modern high-performance computing systems, power consumption within datapath modules due to redundant switching poses a significant challenge. Reducing this redundant switching is crucial not only for conserving power but also for improving overall system efficiency and reliability. However, conventional operand isolation schemes designed to address this issue often incur considerable overhead in terms of delay, power, and area.

This paper addresses these challenges by presenting novel operand isolation techniques based on supply gating. These techniques aim to mitigate the overheads associated with isolating circuitry while effectively reducing redundant switching within the datapath modules. Additionally, the proposed schemes target leakage minimization and introduce additional operand isolation at the internal logic level of the datapath, further contributing to power consumption reduction.

The primary focus of this work is to integrate these novel techniques into a comprehensive synthesis flow for low-power datapath synthesis. By leveraging advanced power and delay models, we aim to develop an efficient methodology for designing low-power datapaths while maintaining optimal performance and functionality.

Through experimental evaluation and validation, we demonstrate the effectiveness and feasibility of the proposed techniques in achieving significant power savings without compromising performance. Our synthesis flow enables designers to systematically explore trade-offs between power, delay, and area, providing valuable insights for designing energy-efficient datapath modules for high-performance applications.

Overall, this paper contributes to the advancement of low-power design methodologies for datapath modules, offering practical solutions to address the growing demand for energy-efficient computing systems in various domains.

II. Background

A. Previous Works

Previous works in the field of low-power neural networks have explored various techniques aimed at reducing power consumption while maintaining performance. One source of inspiration for our research was an open-source GitHub project with the objective of converting convolutional neural network (CNN) layers from Python code into Verilog for simulation.

The GitHub project aimed to bridge the gap between CNN models implemented in high-level programming languages like Python and their hardware realization for simulation in Verilog. Initially, the project focused on converting a simple CNN with five layers from Python code into Verilog. However, to facilitate implementation in Verilog, each convolutional layer had to be split into multiple input blocks and convolution calculation units.

While the GitHub project did not specifically address power efficiency techniques such as voltage scaling or operand gating, it provided a foundational framework for our research. By transforming CNN layers into Verilog code suitable for simulation, the project enabled us to explore power-saving methods tailored to hardware implementation.

Our work built upon the ideas and methodologies presented in the GitHub project, leveraging the Verilog implementation of CNN layers as a starting point. With this foundation in place, we furthered our research by investigating power efficiency techniques, including voltage scaling and operand gating. By integrating these techniques into the Verilog implementation, we aimed to achieve significant reductions in power consumption while preserving the performance of the neural network.

The following viual representation of what the CNN architecture looks like when run through using verilog and simulation can be seen below in Figure 10.

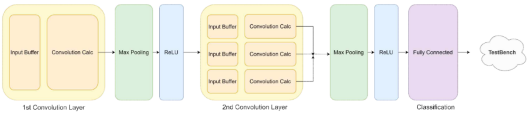


Fig. 1. Visualization of the CNN Verilog Architecture

B. Motivations

The drive for energy-efficient computing systems has become paramount in the era of ubiquitous mobile and IoT (Internet of Things) devices. With the ever-expanding landscape of connected technologies, the demand for neural network hardware capable of operating within stringent power constraints has surged. These constraints arise from a multitude of factors, including battery limitations, thermal considerations, and environmental sustainability goals.

In response to this pressing need, researchers have continued to try and develop innovative low-power techniques tailored specifically for neural network architectures. These techniques aim to not only minimize power consumption but also optimize performance and maintain scalability across diverse application domains.

At the heart of this need lies the recognition of the critical role neural networks play in modern computing paradigms. From edge devices performing real-time inference tasks to data centers crunching massive datasets, neural networks have become indispensable tools driving innovation and progress across various sectors.

However, the inherent complexity and computational intensity of neural networks pose significant challenges in terms of power efficiency. Traditional approaches often fall short in mitigating power consumption without compromising performance, highlighting the need for novel methodologies and design paradigms.

By delving into the realm of low-power neural networks, researchers seek to unlock the potential for sustainable and scalable computing solutions. These solutions hold the promise of extending battery life, reducing energy costs, and minimizing environmental impact, thereby fostering a more efficient and eco-friendly computing ecosystem.

Through the exploration of cutting-edge low-power techniques, the research community aims to not only address the immediate challenges posed by power constraints but also lay the groundwork for future advancements in energy-efficient neural network hardware. By pushing the boundaries of innovation, researchers aspire to usher in a new era of intelligent computing systems that are not only powerful but also energy efficient.

III. Methodology

We employ voltage scaling and operand gating techniques to reduce power consumption in neural network hardware. Voltage scaling involves lowering the supply voltage, while operand gating selectively turns off parts of the data path. These techniques aim to decrease both switching and leakage power.

A. Voltage Scaling

Lowering the supply voltage linearly decreases power consumption, as described by the power equation $P = V_{DD} \times I_D$. However, voltage scaling affects speed and delay.

This technique is heavily based on which library was used and what its set up to be used with, since standard libraries are optimized for use at specific Vdd values. For instance we used two different libraries to compare different Vdd values in our project, namely the Sky130nm and NandGate45nm libraries, to compare the impact of voltage scaling on power consumption. The Sky130nm library operates at a nominal voltage of 1.8 volts, while the NandGate45nm library operates at a lower nominal voltage of 1.1 volts. By synthesizing our designs with these libraries, we were able to assess how varying the supply voltage affects power consumption.

Our findings revealed notable differences in power consumption between the two libraries, highlighting the importance of voltage scaling in achieving energy efficiency. The NandGate45nm library, with its lower nominal voltage, demonstrated significantly lower total power consumption compared to the Sky130nm library. This disparity underscores the effectiveness of voltage scaling in reducing power consumption, as lower supply voltages result in lower dynamic and static power consumption.

B. Operand Gating

Operand gating decreases switching power by turning off parts of the data path. This technique reduces the number of active transistors and thereby reduces both switching and leakage power.

At the heart of operand gating is the concept of selectively activating or deactivating transistors within the data path based on the operational requirements of the circuit. When certain operations are not being performed, the corresponding transistors can be turned off, effectively cutting off the flow of current and reducing both dynamic and static power consumption.

One of the key advantages of operand gating is its ability to target both switching and leakage power simultaneously. By reducing the number of active transistors, switching power is minimized as fewer transistors switch states during each clock cycle. Additionally, by gating off unused portions of the data path, leakage power is also reduced as fewer transistors are actively consuming power when the circuit is idle.

1) Convolutional Neural Network Description: The Convolutional Neural Network (CNN) architecture implemented for this project is designed for image classification tasks, specifically using the MNIST dataset. The CNN architecture consists of several layers including input, convolutional, max pooling, and fully connected layers.

a) Input Layer: The input layer of the CNN accepts grayscale images from the MNIST dataset. Each image in the dataset is of size 28x28 pixels, representing handwritten digits ranging from 0 to 9.

b) Convolution Layer 1: The first convolutional layer applies a set of learnable filters to the input image to extract features such as edges, corners, and textures. These filters slide across the input image, performing element-wise multiplications and summations to produce feature maps.

c) ReLU Activation Function: After each convolutional layer, a Rectified Linear Unit (ReLU) activation function is applied element-wise to the feature maps. ReLU introduces non-linearity to the network by thresholding the output at zero, allowing the network to learn complex patterns and representations.

d) Max Pooling Layer 1: Following the first convolutional layer, max pooling is applied to reduce the spatial dimensions of the feature maps while retaining the most important information. Max pooling operates by partitioning the input into non-overlapping regions and retaining only the maximum value from each region.

e) Convolution Layer 2: The second convolutional layer further extracts higher-level features from the output of the first max pooling layer. Similar to the first convolutional layer, this layer applies a set of filters to the feature maps produced by the previous layer.

f) ReLU Activation Function: After the second convolutional layer, another ReLU activation function is applied to introduce non-linearity to the network and enhance its representational power.

g) Max Pooling Layer 2: Another max pooling layer is applied after the second convolutional layer to further reduce the spatial dimensions of the feature maps while preserving important information.

h) Flatten Layer: After the second max pooling layer, the feature maps are flattened into a one-dimensional vector. This flattening process prepares the data for input into the fully connected layer.

i) Fully Connected Layer: The flattened feature vector is connected to a fully connected layer, also known as a dense layer. This layer performs a series of matrix multiplications and activation functions to produce output logits representing the predicted class probabilities.

j) Output Layer: Finally, the output layer of the CNN uses a softmax activation function to convert the logits into probability scores for each class (0 to 9). The class with the highest probability score is predicted as the output label for the input image.

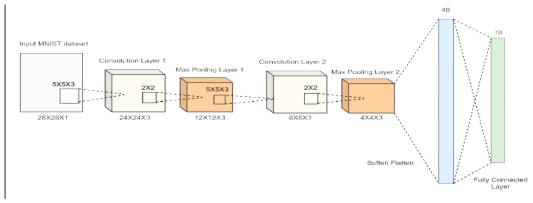


Fig. 2. Visualization of the CNN Architecture

C. Design 2: 2-Input CNN

Design 2 of the CNN architecture is a more complex network that takes two inputs: a 128x128 image of the letter A and an 8x8 pattern test image. The architecture consists of convolutional layers followed by aggregation, max pooling, and post-processing layers.

- **Input Layer:** The input layer of the CNN accepts two inputs: a 128x128 image of the letter A and an 8x8 pattern test image.
- **Convolution Layer 1:** Each input is passed through a separate convolutional layer to extract features. The convolutional layers operate independently and produce feature maps for each input.
- **Aggregation Layer:** The feature maps from the two convolutional layers are aggregated into a single feature map of size 121x121. This layer combines the information from both inputs to create a unified representation.
- **Max Pooling Layer:** Following the aggregation layer, max pooling is applied to reduce the spatial dimensions of the feature map while retaining important information. The max pooling layer outputs a feature map of size 60x60.
- **Thresholding and Sorting:** The output of the max pooling layer is thresholded to remove noise and enhance features. Subsequently, the feature map is sorted to identify significant patterns and structures.

This design aims to process two different types of inputs and extract meaningful information from each input individually before combining them into a unified representation for further analysis.

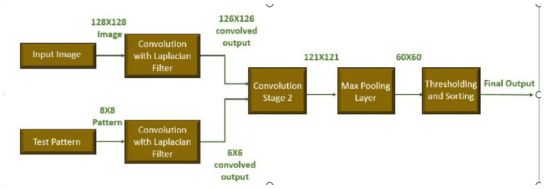


Fig. 3. Visualization of the CNN Architecture

D. EDA Tool Flow

In our project implementation, we utilized two key EDA (Electronic Design Automation) tools: ModelSim and Genus by Cadence.

ModelSim served as our primary tool for RTL simulation and verification. It provided us with a robust environment for simulating our hardware designs at the register-transfer level (RTL) and verifying their functionality before synthesis. ModelSim enabled us to perform behavioral simulation, functional verification, timing analysis, and advanced debugging, ensuring that our designs met the desired specifications.

After successful RTL simulation and verification, we transitioned to synthesis using Genus by Cadence. Genus is a powerful synthesis tool that allowed us to convert our RTL code into gate-level netlists suitable for implementation on

ASIC or FPGA platforms. Genus enabled us to optimize our designs for performance, area, and power consumption, providing valuable insights for further refinement.

The use of both ModelSim and Genus integrated seamlessly into the standard ASIC/FPGA design flow. ModelSim facilitated efficient RTL simulation and verification, while Genus enabled synthesis and optimization of our designs for implementation on hardware platforms.

Overall, the combination of ModelSim and Genus played a crucial role in our project by facilitating efficient RTL simulation, verification, synthesis, and optimization, contributing to the successful implementation and validation of our hardware designs.

IV. Results and Analysis

A. Testing Setup

We conducted power analysis on two distinct convolutional neural network (CNN) designs using ModelSim and Genus software.

1) RTL Setup and Changes: Before synthesizing the CNN designs, we made significant RTL changes to optimize power consumption.

RTL Changes for Design 1:

The most complex block in this simple CNN is the second convolution layer, divided into three parallel blocks of an input buffer followed by a convolution calculator. To reduce dynamic power, we proposed using a 5-bit enable bus to selectively turn off Multiply-Accumulate (MAC) units within the convolutional layers. This approach helped in considerable dynamic power reduction without impacting accuracy. Table 1 summarizes the code changes required for adding clock gating.

Baseline RTL:

```
calc_out_1_1 <= data_out1_0*weight_1[0] + data_out1_1*weight_1[1] +
data_out1_2*weight_1[2] + data_out1_3*weight_1[3] + data_out1_4*weight_1[4];
calc_out_1_2 <= data_out1_5*weight_1[5] + data_out1_6*weight_1[6] +
data_out1_7*weight_1[7] + data_out1_8*weight_1[8] + data_out1_9*weight_1[9];
calc_out_1_3 <= data_out1_10*weight_1[10] + data_out1_11*weight_1[11] +
data_out1_12*weight_1[12] + data_out1_13*weight_1[13] +
data_out1_14*weight_1[14];
calc_out_1_4 <= data_out1_15*weight_1[15] + data_out1_16*weight_1[16] +
data_out1_17*weight_1[17] + data_out1_18*weight_1[18] +
data_out1_19*weight_1[19];
calc_out_1_5 <= data_out1_20*weight_1[20] + data_out1_21*weight_1[21] +
data_out1_22*weight_1[22] + data_out1_23*weight_1[23] +
data_out1_24*weight_1[24];
```

Modified RTL to add enable signals for every 5 MAC units:

```
calc_out_1_1 <= enable[0] ? (data_out1_0*weight_1[0] + data_out1_1*weight_1[1] +
data_out1_2*weight_1[2] + data_out1_3*weight_1[3] + data_out1_4*weight_1[4]) : 0;
calc_out_1_2 <= enable[1] ? (data_out1_5*weight_1[5] + data_out1_6*weight_1[6] +
data_out1_7*weight_1[7] + data_out1_8*weight_1[8] + data_out1_9*weight_1[9]) : 0;
calc_out_1_3 <= enable[2] ? (data_out1_10*weight_1[10] + data_out1_11*weight_1[11]
+ data_out1_12*weight_1[12] + data_out1_13*weight_1[13] +
data_out1_14*weight_1[14]) : 0;
calc_out_1_4 <= enable[3] ? (data_out1_15*weight_1[15] + data_out1_16*weight_1[16]
+ data_out1_17*weight_1[17] + data_out1_18*weight_1[18] +
data_out1_19*weight_1[19]) : 0;
calc_out_1_5 <= enable[4] ? (data_out1_20*weight_1[20] + data_out1_21*weight_1[21]
+ data_out1_22*weight_1[22] + data_out1_23*weight_1[23] +
data_out1_24*weight_1[24]) : 0;
```

Fig. 4. RTL Changes for Design 1

RTL Changes for Design 2:

The second CNN design is bigger and more complex than the first design. To reduce dynamic and leakage power, we optimized the RTL implementation by reducing the bit width of the input and output buses of the convolution stage and subsequent adder stages. This optimization ensured the correctness of results while achieving considerable power savings. Table 2 describes how this was implemented for different stages of the design.

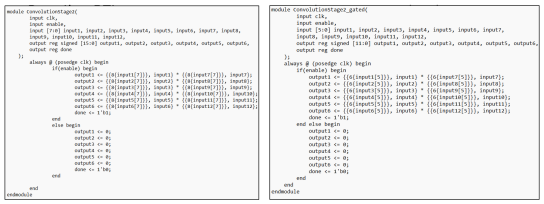


Fig. 5. RTL Changes for Design 2

Reducing the input and output width for Most Significant Bits (MSB) that are unused helps in minimizing unnecessary power consumption. By truncating unused bits, we eliminate unnecessary switching activity in the circuitry, leading to power savings without compromising the accuracy of the results.

2) Testing Image for Design 2: For Design 2, the testing image used was a 128x128 image containing various instances of the letter 'A'. The objective was to develop a CNN that could accurately detect and count the occurrences of the letter 'A' within the image.

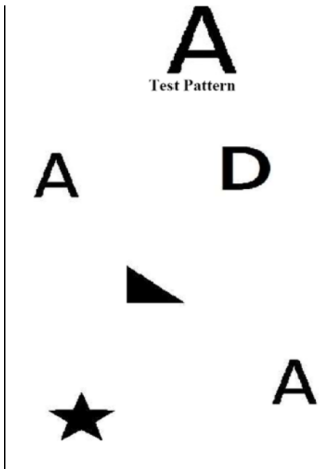


Fig. 6. Testing Image for Design 2

This image served as the basis for evaluating the effectiveness of the complex CNN architecture in detecting and counting specific patterns within larger images and proves ours techniques can be integrated with any large scale CNN.

B. Testing Results

1) Simulation for Design 1: The input image contains a binary encoded digit '2'. In the expected output of the simulation, we should see the output bus 'decision' set to "0010" when all the *valid_out_** signals have toggled to logic '1'. Figure 16 shows the simulation waveform in ModelSim when all enable signals are set to logic '1', which is equivalent to the Baseline RTL where none of the MAC units are gated. Figure 17 shows the simulation waveform where the LSB bit of the enable bus is set to logic '0', indicating that 3 MAC units, each containing 5 multipliers and 4 adders, are gated and the accuracy of the decision is not impacted. This

demonstrates that even after turning off 15 multipliers and 12 adders, the output remains the same, leading to considerable dynamic power savings.

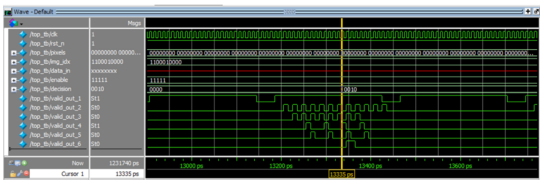


Fig. 7. Simulation Waveform for Design 1: All MAC Units Enabled

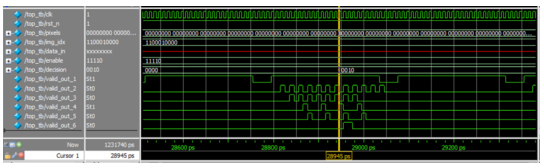


Fig. 8. Simulation Waveform for Design 2

2) Simulation for Design 2: Figure 15 describes the input image and the test pattern 'A'. The aim of this CNN is to determine the number of occurrences of test pattern 'A' within the input image. Figure 18 describes the simulation waveforms from ModelSim depicting that the test pattern 'A' was detected twice in the input image.

Fig. 9. ModelSim Waveform for Design 2

3) Power Analysis:

V. Power Analysis

In our project, we meticulously analyzed the power consumption of our convolutional neural network (CNN) designs using both commercial and open-source tools. We began by implementing the CNN architecture in ModelSim, a popular RTL simulator, and later synthesized it in the Genus software using two different libraries: Sky130nm and NandGate45nm.

Upon comparing the synthesis results for the Sky130nm and NandGate45nm libraries, we observed notable differences in power consumption. The Sky130nm library, operating at a nominal voltage of 1.8 volts, exhibited a total power consumption of 149.74 mW for our baseline CNN model. In contrast, the NandGate45nm library, with a nominal voltage of 1.1 volts, yielded a substantially lower total power consumption of 61.36 mW. This disparity highlighted the effectiveness of voltage scaling in reducing power consumption, with the NandGate45nm library demonstrating a significant reduction in total power consumption compared to the Sky130nm library.

Furthermore, we explored the impact of low-power techniques, particularly operand gating, on power consumption. After implementing operand gating on our CNN architecture, we conducted another synthesis run to evaluate its effect on power consumption.

Our findings revealed substantial power savings achieved through low-power techniques. For instance, Table I provides a detailed breakdown of power consumption for our baseline CNN model synthesized using different libraries. We observed that the NandGate45nm library resulted in lower leakage power, internal power, switching power, and consequently, total power consumption compared to the Sky130nm library.

Table II further illustrates the impact of low-power techniques on power consumption for Design 1. With the implementation of low-power techniques, including voltage scaling and operand gating, we achieved significant reductions in power consumption across various power domains. For example, the total power consumption for Design 1 with low-power techniques decreased from 0.14974 W to 0.06136 W in the NandGate45nm library, showcasing a remarkable 44% reduction in power consumption.

Additionally, we evaluated the power consumption of Design 2 using both Nangate libraries and the Sky library. Table III presents the power consumption results for Design 2 synthesized with Nangate libraries, highlighting the differences in power consumption between the baseline CNN and the CNN with operand gating. Similarly, Table IV provides insights into the power consumption of Design 2 synthesized with the Sky library.

In summary, our power analysis demonstrates the efficacy of voltage scaling and low-power techniques, such as operand gating, in reducing power consumption in CNN designs. By leveraging these techniques, we achieved substantial power savings, paving the way for more energy-efficient neural network implementations in hardware.

TABLE I
Power Consumption Results for Baseline CNN Model

Library	NAND Gate	Sky Library
Nominal Voltage (V)	1.1	1.8
Leakage Power (W)	3.09075×10^{-3}	2.89119×10^{-7}
Internal Power (W)	3.25866×10^{-2}	9.65589×10^{-2}
Switching Power (W)	2.56389×10^{-2}	5.31801×10^{-2}
Total Power (W)	6.13612×10^{-2}	1.49739×10^{-1}

TABLE II
Power Consumption Results for Design 1 with Low-Power Techniques

Library	NAND Gate	Sky Library
Nominal Voltage (V)	1.1	1.8
Leakage Power (W)	3.08476×10^{-2}	4.16858×10^{-7}
Internal Power (W)	3.18092×10^{-2}	1.24431×10^{-1}
Switching Power (W)	2.52959×10^{-2}	8.30875×10^{-2}
Total Power (W)	6.01898×10^{-2}	2.07519×10^{-1}

TABLE III
POWER CONSUMPTION RESULTS FOR DESIGN 2 (NANGATE LIBRARIES)

Library	Nangate 45nm		Nangate 15nm	
Design	**Baseline CNN**	**CNN + Operand Gating**	**Baseline CNN**	**CNN + Operand Gating**
Leakage (mW)	1.39E-01	1.17E-01	9.41E-02	6.92E-02
Internal (mW)	7.43E-01	7.24E-01	3.19E-01	2.60E-01
Switching (mW)	2.70E-01	3.42E-01	7.74E-02	6.12E-02
Total (mW)	1.15E+00	1.18E+00	4.91E-01	3.91E-01

TABLE IV
POWER CONSUMPTION RESULTS FOR DESIGN 2 (SKY LIBRARY)

Library	Baseline CNN	CNN + Operand Gating
Leakage (mW)	NA	1.74E-05
Internal (mW)	NA	4.88E+00
Switching (mW)	NA	8.47E-01
Total (mW)	NA	5.73E+00

VI. DISCUSSION

A. Bottleneck(s)/Challenge(s) Encountered

During the course of the project, several challenges and bottlenecks were encountered, including issues related to synthesis constraints, optimization trade-offs, and resource limitations. Overcoming these challenges required careful analysis and iterative refinement of the design methodologies.

Additionally, we faced significant issues with the Skynet library for Design 2, which hindered our ability to obtain satisfactory baseline results. Due to these challenges, we had to switch to comparing the Nangate 45nm library and a Nangate 15nm library to ensure meaningful comparisons and accurate power consumption analysis.

Additionally, the synthesis process was often lengthy, sometimes taking many hours to complete. This time-consuming aspect of the project made time a crucial factor, limiting our ability to test as many combinations and iterations as we would have liked. Despite this challenge, we prioritized efficiency and focused on optimizing the synthesis workflow to make the most of the available time.

B. What Did We Learn?

1) Specific Insights:

- **Low-Power Techniques Application**: Through rigorous experimentation, we successfully applied low-power techniques at the register-transfer level (RTL), specifically targeting FPGA and ASIC designs. Our focus on clock gating, memory splitting, and multi-voltage scaling yielded substantial reductions in power consumption without compromising computational performance.
  Implement power gating techniques to shut down power to the convolution layer when it is unused based on the availability of correct input data for that layer. It helps in reducing both static and dynamic power consumption. HVT transistors can be used to implement the logic under convolution layer which requires maximum multipliers and adders. This can give considerable leakage power savings. The trade-off is that this can make timing paths through the multiply-add units as the critical path. To combat this, more pipeline stages can be added to control the worst negative slack. Power intent can be read during synthesis which can define power domains, which are groups of logic or circuitry that can be powered on or off independently to manage power consumption. Dynamic voltage and frequency scaling (DVFS) can also be used to adjust operating voltage and clock frequencies based on workload requirements and power constraints.
- **Low power adders and multipliers**
  - **Conditional Sum Adder:** This adder reduces the number of carry propagation stages by selectively generating and propagating carries based on the input data pattern. It minimizes unnecessary transitions in the carry path leading to a reduction in power consumption. Compared to ripple carry adders, it has fewer stages for carry propagation, resulting in lower dynamic power consumption due to reduced switching activity. It typically requires fewer transistors compared to carry-select adders or carry-lookahead adders, which results in reduced area and power consumption.
  - **Wallace Tree Multiplier:** A high-speed, low-power multiplier that uses carry-save addition and parallel reduction techniques to reduce the number of partial products and the critical path delay. It utilizes a carry-save addition method to add partial products without generating the final sum. The partial products and carry bits are then combined in a tree structure to produce the result. This tree structure enables efficient sharing of resources and minimizes the number of full adders required in the critical path, making it area-efficient and easily pipelined, ideal for low-power applications requiring high-speed multiplication.

2) Scalability Insights:

- **Modular Design Adaptability**: We observed that modular design principles are crucial for scalability in neural network hardware. Our CNN accelerator architecture exemplifies this adaptability, allowing for the integration of low-power techniques without necessitating significant redesign efforts. This modularity ensures scalability for future iterations and variations of the CNN-based object detection system.

3) Real-World Applications:

- **Impact on Autonomous Vehicles**: Our research has direct implications for real-world applications such as autonomous vehicles. By reducing power consumption in CNN accelerators, we contribute to the development of energy-efficient hardware platforms suitable for deployment in vehicles with limited power resources.

4) Further Research Directions:

- **Integration into ASIC Fabrication**: Looking ahead, our focus shifts towards integrating these low-power tech-

niques into ASIC fabrication. This next phase of research aims to leverage our findings to develop low-power CNN accelerators tailored for ASIC implementation, thereby advancing the power efficiency of autonomous vehicle systems.

By combining specific insights with considerations for scalability and real-world applications, our research lays the groundwork for the development of sustainable and scalable neural network hardware solutions for critical applications like object detection in autonomous vehicles.

C. Implementation Timeline

The implementation timeline for the project spanned several weeks and involved multiple stages, including research, design exploration, synthesis, and analysis. Each stage required meticulous planning and coordination to ensure timely progress and successful completion of the project objectives.

The implementation process began with selecting an appropriate baseline CNN model that was simple and widely used for image classification tasks. Once chosen, we focused on finding examples of this baseline model implemented in Verilog to serve as a reference for our hardware implementation. This phase involved studying existing literature and code repositories to gather insights and code snippets.

Following this, we proceeded to convert the baseline CNN model into Verilog code and conducted RTL simulations in ModelSim to verify the functionality and correctness of our hardware implementation. This step was crucial in ensuring that our Verilog code accurately represented the desired behavior of the CNN model.

After confirming the successful RTL simulation, we moved on to synthesizing the Verilog code using synthesis tools such as Genus to generate gate-level netlists. This allowed us to analyze the power consumption and performance metrics of our baseline CNN design.

Once we obtained baseline results, we explored various techniques to enhance power efficiency, including operand gating, voltage scaling, and test gating. We experimented with different libraries to assess the impact of these techniques on power consumption and performance.

As part of our exploration, we introduced a more complex CNN architecture, referred to as Design 2, which incorporated two input streams: a 128x128 image of the letter A and an 8x8 pattern test image. We investigated whether the same power-saving techniques, such as operand gating, could be applied to Design 2 without compromising its performance.

Overall, the implementation timeline involved iterative refinement and validation of our hardware designs, culminating in the successful integration of power-saving techniques into both the baseline CNN model and the more complex Design 2 architecture. Each week the team dedicated approximately 36 to 40 hours to testing.

D. Division of Labour

The division of labor within the project team was based on individual strengths and expertise. Tasks were assigned and distributed according to team members' skill sets and interests, ensuring efficient utilization of resources and effective collaboration. Regular communication and coordination were maintained to track progress, address challenges, and ensure alignment with project goals. Deep, Hunter, and Muhammad worked giving many hours towards testing each design and committing changes as necessary in Modelsim. Stephen worked on the proving insight to the general CNN architecture and providing Python code and diagrams to assist with this as well as putting results together in the report.

VII. Conclusion

In this project, we explored and implemented transistor-level low-power techniques on neural networks, focusing on voltage scaling and operand gating. By employing these techniques, we aimed to reduce power consumption while maintaining performance in hardware implementations of convolutional neural networks (CNNs).

Our methodology involved implementing these techniques in Verilog and conducting power analysis using commercial and open-source tools such as ModelSim and Genus. Through rigorous simulation and synthesis, we evaluated the effectiveness of voltage scaling and operand gating in reducing both switching and leakage power.

The results of our power analysis highlighted significant power savings achieved through low-power techniques. Specifically, we observed notable differences in power consumption between different libraries and synthesis runs, demonstrating the impact of voltage scaling and operand gating on reducing total power consumption.

Furthermore, we successfully applied these techniques to more complex CNN architectures, showcasing their versatility and effectiveness in various neural network designs. Our findings underscore the importance of integrating low-power techniques into hardware designs to address the growing demand for energy-efficient computing systems.

Moving forward, future work will focus on further optimizing these techniques to achieve even greater power savings without compromising performance. Additionally, ongoing research will explore new approaches to low-power neural network design, leveraging advances in technology and methodology to push the boundaries of energy efficiency in hardware implementations.

In conclusion, this project contributes to the advancement of low-power design methodologies for neural networks, offering practical solutions to address the challenges of power consumption in hardware implementations. By leveraging voltage scaling and operand gating techniques, we pave the way for more energy-efficient and sustainable computing systems in various domains.

Acknowledgment

We would like to acknowledge the support of Herbert Wertheim College of Engineering, University of Florida, Gainesville, USA. We would also like to acknowledge the support of our professor and the teacher assistants that helped us throughout this journey.

References

1 boaaaang, "boaaaang/CNN-Implementation-in-Verilog," GitHub, Apr. 26, 2024. https://github.com/boaaaang/CNN-Implementation-in-Verilog/tree/master (accessed May 01, 2024).

2 A. Badhan, "AniketBadhan/Convolutional-Neural-Network," GitHub, May 01, 2024. https://github.com/AniketBadhan/Convolutional-Neural-Network (accessed May 01, 2024).

Appendix A
Figures and Tables

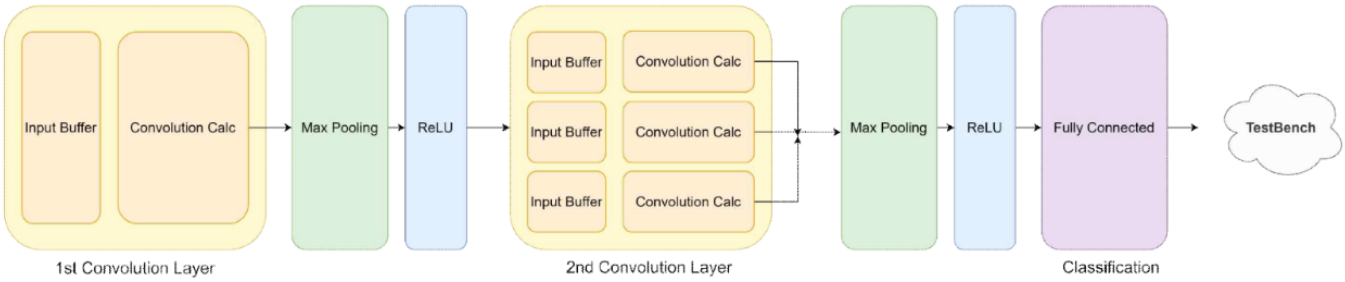


Fig. 10. Visualization of the CNN Verilog Architecture

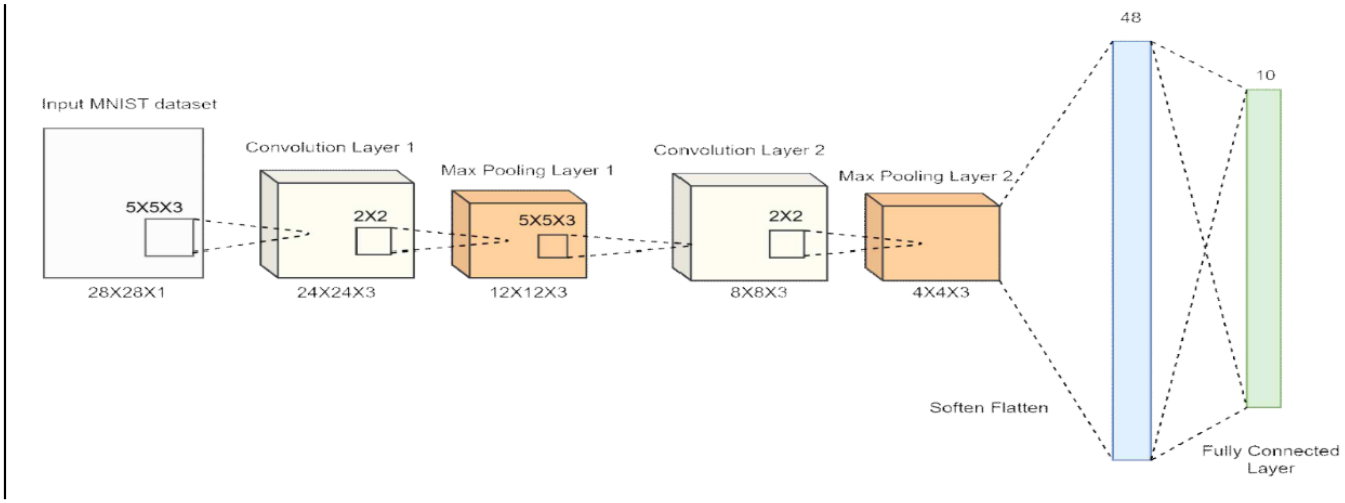


Fig. 11. Visualization of the CNN Architecture

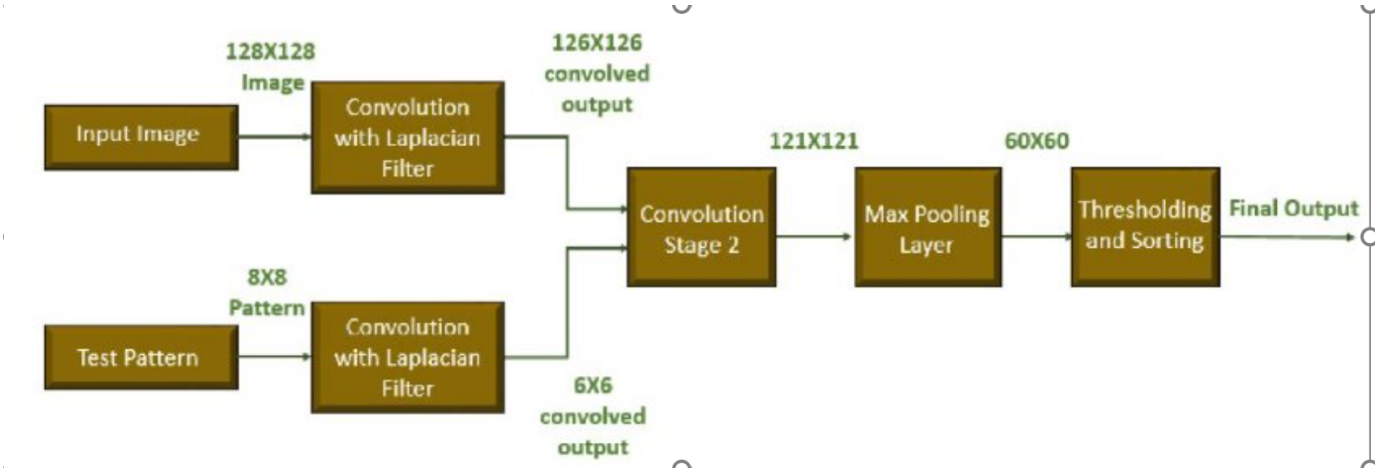


Fig. 12. Visualization of the CNN Architecture for Design 2

Baseline RTL:

```
calc_out_1_1 <= data_out1_0*weight_1[0] + data_out1_1*weight_1[1] +
data_out1_2*weight_1[2] + data_out1_3*weight_1[3] + data_out1_4*weight_1[4];
calc_out_1_2 <= data_out1_5*weight_1[5] + data_out1_6*weight_1[6] +
data_out1_7*weight_1[7] + data_out1_8*weight_1[8] + data_out1_9*weight_1[9];
calc_out_1_3 <= data_out1_10*weight_1[10] + data_out1_11*weight_1[11] +
data_out1_12*weight_1[12] + data_out1_13*weight_1[13] +
data_out1_14*weight_1[14];
calc_out_1_4 <= data_out1_15*weight_1[15] + data_out1_16*weight_1[16] +
data_out1_17*weight_1[17] + data_out1_18*weight_1[18] +
data_out1_19*weight_1[19];
calc_out_1_5 <= data_out1_20*weight_1[20] + data_out1_21*weight_1[21] +
data_out1_22*weight_1[22] + data_out1_23*weight_1[23] +
data_out1_24*weight_1[24];
```

Modified RTL to add enable signals for every 5 MAC units:

```
calc_out_1_1 <= enable[0] ? (data_out1_0*weight_1[0] + data_out1_1*weight_1[1] +
data_out1_2*weight_1[2] + data_out1_3*weight_1[3] + data_out1_4*weight_1[4]) : 0;
calc_out_1_2 <= enable[1] ? (data_out1_5*weight_1[5] + data_out1_6*weight_1[6] +
data_out1_7*weight_1[7] + data_out1_8*weight_1[8] + data_out1_9*weight_1[9]) : 0;
calc_out_1_3 <= enable[2] ? (data_out1_10*weight_1[10] + data_out1_11*weight_1[11]
+ data_out1_12*weight_1[12] + data_out1_13*weight_1[13] +
data_out1_14*weight_1[14]) : 0;
calc_out_1_4 <= enable[3] ? (data_out1_15*weight_1[15] + data_out1_16*weight_1[16]
+ data_out1_17*weight_1[17] + data_out1_18*weight_1[18] +
data_out1_19*weight_1[19]) : 0;
calc_out_1_5 <= enable[4] ? (data_out1_20*weight_1[20] + data_out1_21*weight_1[21]
+ data_out1_22*weight_1[22] + data_out1_23*weight_1[23] +
data_out1_24*weight_1[24]) : 0;
```

Fig. 13. RTL Changes for Design 1

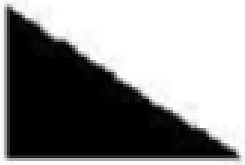
```
module ConvolutionStage2(
        input clk,
        input enable,
        input [7:0] input1, input2, input3, input4, input5, input6, input7, input8,
        input9, input10, input11, input12,
        output reg signed [15:0] output1, output2, output3, output4, output5, output6,
        output reg done
    );
        always @ (posedge clk) begin
                if(enable) begin
                        output1 <= {{8{input1[7]}}, input1} * {{8{input7[7]}}, input7};
                        output2 <= {{8{input2[7]}}, input2} * {{8{input8[7]}}, input8};
                        output3 <= {{8{input3[7]}}, input3} * {{8{input9[7]}}, input9};
                        output4 <= {{8{input4[7]}}, input4} * {{8{input10[7]}}, input10};
                        output5 <= {{8{input5[7]}}, input5} * {{8{input11[7]}}, input11};
                        output6 <= {{8{input6[7]}}, input6} * {{8{input12[7]}}, input12};
                        done <= 1'b1;
                end
                else begin
                        output1 <= 0;
                        output2 <= 0;
                        output3 <= 0;
                        output4 <= 0;
                        output5 <= 0;
                        output6 <= 0;
                        done <= 1'b0;
                end

        end
endmodule
```

```
module ConvolutionStage2_gated(
        input clk,
        input enable,
        input [5:0] input1, input2, input3, input4, input5, input6, input7,
        input8, input9, input10, input11, input12,
        output reg signed [11:0] output1, output2, output3, output4, output5, output6,
        output reg done
    );
    always @ (posedge clk) begin
        if(enable) begin
            output1 <= {{6{input1[5]}}, input1} * {{6{input7[5]}}, input7};
            output2 <= {{6{input2[5]}}, input2} * {{6{input8[5]}}, input8};
            output3 <= {{6{input3[5]}}, input3} * {{6{input9[5]}}, input9};
            output4 <= {{6{input4[5]}}, input4} * {{6{input10[5]}}, input10};
            output5 <= {{6{input5[5]}}, input5} * {{6{input11[5]}}, input11};
            output6 <= {{6{input6[5]}}, input6} * {{6{input12[5]}}, input12};
            done <= 1'b1;
        end else begin
            output1 <= 0;
            output2 <= 0;
            output3 <= 0;
            output4 <= 0;
            output5 <= 0;
            output6 <= 0;
            done <= 1'b0;
        end
    end
endmodule
```

Fig. 14. RTL Changes for Design 2

Test Pattern



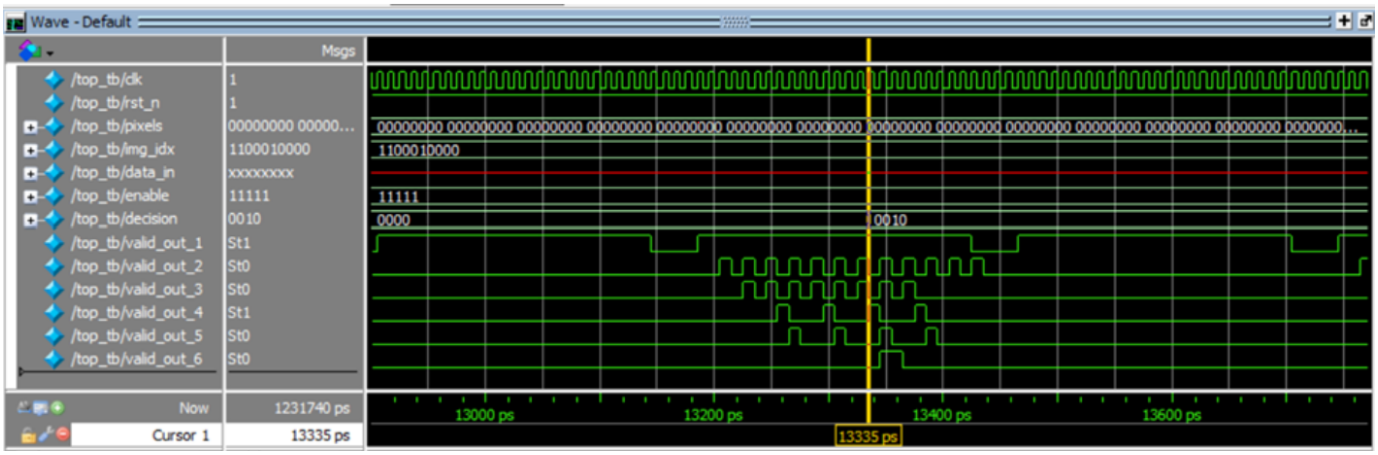


Fig. 16. Simulation Waveform for Design 1: All MAC Units Enabled

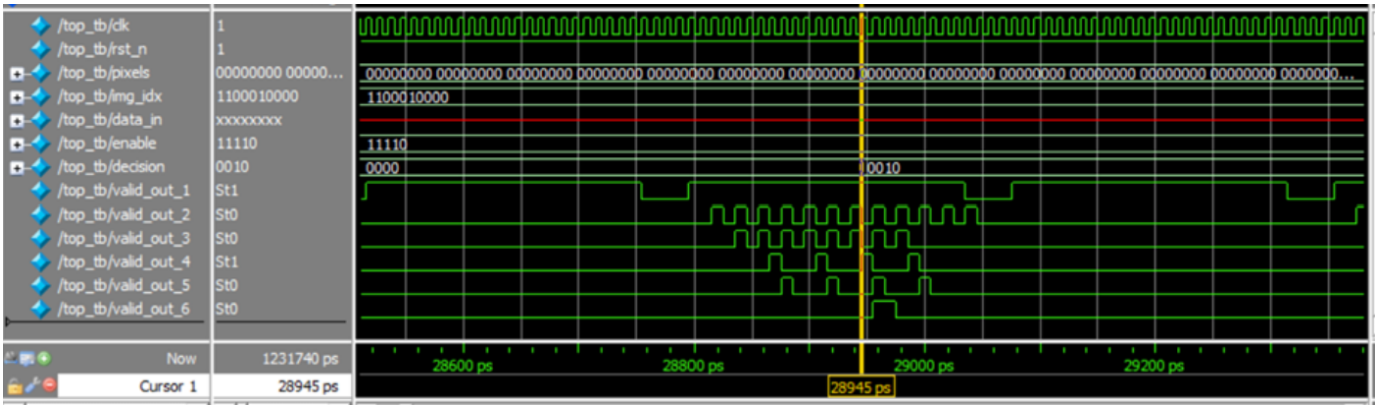


Fig. 17. Simulation Waveform for Design 2

Fig. 18. ModelSim Waveform for Design 2