Digit Recognition Hardware Accelerator

Ryan Lehrman
Department of Electrical Engineering line
University of Pittsburgh
Pittsburgh, USA
ryl19@pitt.edu

Travis Allabon
Department of Computer Engineering
University of Pittsburgh
Pittsburgh, USA
tma57@pitt.edu

Berfin Bircan
Department of Computer Engineering
University of Pittsburgh
Pittsburgh, USA
beb133@pitt.edu

***Abstract*—This paper introduces a specialized hardware accelerator aimed at enhancing digit recognition performance using the MNIST dataset. It also proposes an optimization to the baseline design and explores the benefits and drawbacks of the optimized design.**

***Keywords*—machine learning, neural networks, handwritten digit recognition, optimization, MNIST**

I. Introduction

Machine Learning (ML) techniques, particularly neural networks (NN), revolutionize various fields by enabling computers to learn from data without explicit programming. This innovation, evident in everyday applications like facial recognition and predictive healthcare analysis, underscores the importance of digit recognition [1]. However, conventional central processing units (CPUs) struggle with the computational demands posed by neural network models due to their sequential processing nature.

The complexity of neural network architectures, especially in digit recognition, surpasses CPU capabilities, necessitating efficient hardware accelerators to offload computation-intensive tasks and enhance system efficiency. CPUs' limitations, such as slower speeds, limited parallelism, and high power consumption, underscore the need for specialized hardware accelerators to optimize digit recognition algorithms.

In justifying the selection of a hardware accelerator approach, it is essential to consider the multi-layer perceptron (MLP) architecture commonly used in digit recognition tasks. MLPs typically consist of an input layer, one or more hidden layers, and an output layer. In our design, we opt for a three-layer MLP configuration, comprising an input layer for image data, a hidden layer with 30 elements, and an output layer for digit classification as shown in Figure 1. Furthermore, we leverage the sigmoid function implemented using a look-up table (LUT) to introduce non-linearity into the network, a crucial element for capturing complex relationships within the data.

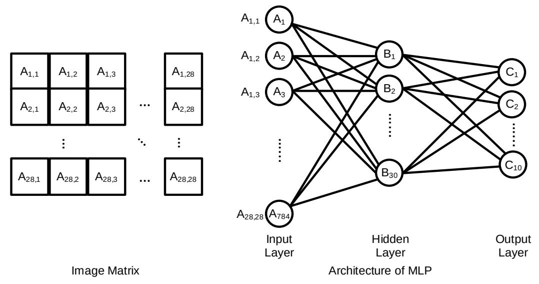


Figure 1: Diagram of Three-Layer MLP

In summary, the decision to design a digit recognition hardware accelerator is motivated by the inadequacies of CPUs in handling ML workloads efficiently. By focusing on a three-layer MLP architecture and leveraging hardware acceleration techniques such as LUT-based sigmoid activation, we aim to address these limitations and pave the way for faster and more power-efficient digit recognition systems. Moreover, by training and testing our hardware accelerator design on the MNIST dataset, a widely adopted dataset for digit recognition tasks in the machine learning community, we ensure its efficacy and accuracy in real-world applications.

II. Baseline Design

The system is composed of three essential components: the external memory, internal SRAM, and the processing module. The external memory, which is situated within the testbench, serves as a simulated storage unit where data can be loaded into it solely for simulation purposes. Similarly, the internal SRAM operates as a non-synthesizable model, employing a handshaking protocol for access, under the control of a practical FSM (Finite State Machine). Unlike the external memory and internal SRAM, the processing module is synthesized and APRed. This module, comprising both FSM and the Datapath, orchestrates tasks such as data loading and MAC operations. An overall diagram of the design is shown in Figure 2, where the light grey is the testbench, which includes the external memory. Then within that is the accelerator, in light blue, which includes the SRAM. And finally highlighted in purple is the processing module, that includes both the FSM as well as the Datapath, which also has its own internal components.

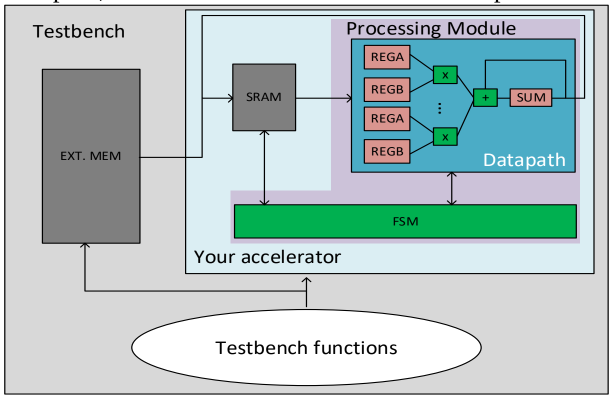


Figure 2: Baseline Design System Block Diagram

This design necessitates integrating memory access within the FSM, which entails the generation of addresses, requesting signals, and read/write controls to facilitate efficient data retrieval and storage. Meanwhile, the Datapath, operating in tandem with the FSM, receives and loads input data such as images, weights, and biases, and performs MAC operations according to the directives dictated by the FSM. This structured approach ensures coordinated processing and allows for seamless execution of tasks within the system.

The system's intricate architecture incorporates distinct elements, each playing a vital role in its operation. While the external memory and internal SRAM facilitate data simulation and access, respectively, the processing module, comprising the FSM and Datapath, orchestrates the execution of tasks, including memory access and MAC operations. This approach ensures efficient data processing and manipulation, laying the foundation for seamless functionality within the system.

Utilizing a dummy-memory model for memory implementation is integral to the system. Key input signals include clk, req, r0w1, addr, and wdata, while outputs include rdata and ack.

Upon initiation by the FSM, req being high signals the start of a memory access, preceded by the readiness of addr and r0w1 signals as shown in Figure 3. With r0w1 denoting read (0) or write (1), addr serves as the memory cell index, incrementing by 1 for each subsequent row. The memory's width and depth are defined by parameters, with memory cells indexed from 0 to depth-1, facilitating seamless data access and manipulation within the system. Additionally, the assignment of mul_temp and mul variables ensures efficient multiplication operations, adhering to specified formats for compatibility and data integrity.

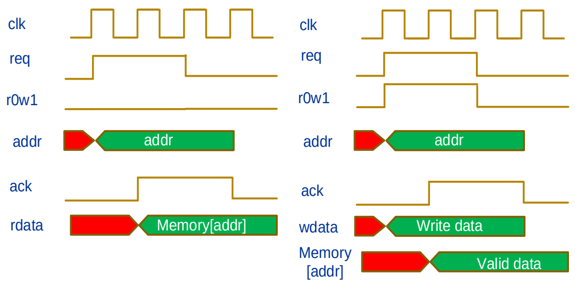


Figure 3: Memory operations

The input undergoes preprocessing to convert 8-bit integers into signed fixed-point format, while the weights and bias, also 8-bit, adhere to a specific format, comprising [sign], [2-bit integer], [5-bit decimal]. To manage multiplication results, truncation is necessary to control data width expansion. Multiplying two numbers in the format [sign], [2-bit integer], [5-bit decimal] yields a result of [sign*sign], [4-bit integer], [25-bit decimal], necessitating truncation of the sign, integer, and decimal components in the output. This process ensures compatibility within the system, maintaining efficient data handling and preventing overflow.

As mentioned in the introduction, the sigmoid function is implemented using a look-up table (LUT) to introduce non-linearity into the network, a crucial element for capturing complex relationships within the data. It does this by applying the equation y = 1/(1+exp(-x)), which limits the values between [0,1] as shown below in Figure 4 [2].

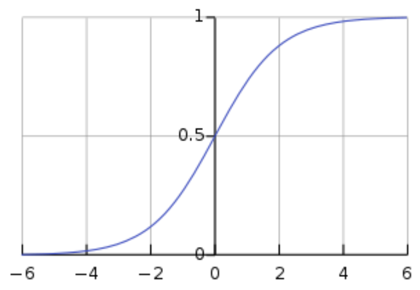


Figure 4: Sigmoid Function

In conclusion, the baseline design, comprising of the external memory, internal SRAM, and the processing module, exemplifies integration of components aimed at facilitating efficient data processing and manipulation. Through the incorporation of memory access within the FSM and a structured datapath, the system ensures seamless execution of tasks, including data loading and MAC operations. Utilization of a dummy-memory model, adherence to specified formats, and careful management of multiplication results underscore the system's commitment to data integrity and compatibility. Furthermore, the implementation of the sigmoid function via a look-up table adds crucial non-linearity, enhancing the system's capability to model complex relationships within the data. Overall, the system stands as a testament to hardware accelerator design considerations, and is poised to meet contemporary computing demands effectively as opposed to a basic CPU.

III. Proposed Design

Similar to the baseline design, in our proposed approach, we introduce a three-layer neural network tailored for digit recognition tasks. This architecture comprises 784 input neurons corresponding to the pixel values of the input images, 30 neurons in the hidden layer, and 10 neurons in the output layer, each representing a digit class from 0 to 9, as shown earlier in Figure 1. However, what distinguishes our design is the strategic variation in the number of multipliers and adders feeding into the Multiply-Accumulate (MAC) units within the network, as shown in the top level diagram of our design in Figure 5.

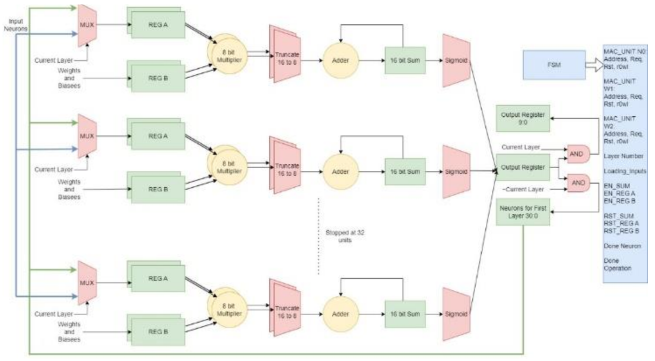


Figure 5: Proposed Top Level Diagram

By dynamically adjusting the resources allocated to each MAC operation, we can optimize the computational efficiency and resource utilization based on the specific requirements of the neural network layers. This approach allows us to efficiently distribute computational resources according to the complexity of the network's operations, thereby enhancing both performance and energy efficiency. Through this innovative adaptation, we aim to achieve a balance between computational power and resource utilization, ultimately leading to improved accuracy and efficiency in digit recognition tasks.

For the FSM of the proposed design the diagram is shown below in Figure 6. Initially the system is idle until the start operation where the neurons are first loaded into the system. Following this, calculations are performed on these neurons. Then once the calculations are complete, the neurons and their states are saved into the memory. Then the neurons transition between the input layer and layer one. Finally, as the process concludes, the output neurons are prepared for exportation to the testbench, where their efficacy and performance can be evaluated and analyzed.

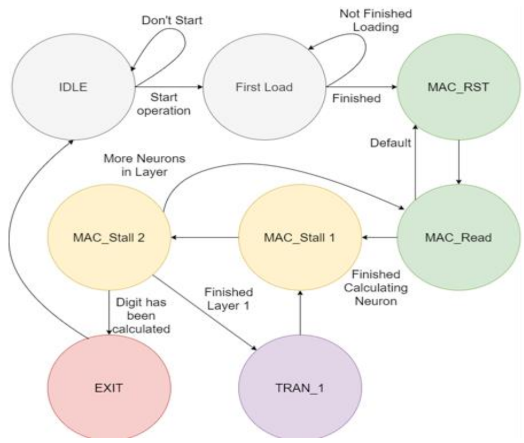


Figure 6: Proposed Finite State Machine

It is important to notice that the operation of the accelerator has two distinct functions. The first is the load phase, and then the operational phase. The load phase is when all of the SRAM are loaded with weights, biases, and input neurons, setting the stage for computation. This phase occurs within every calculation of a digit, but the length of this phases reduces significantly after computing the first digit. This is because after computing the first digit, the weights and biases for each layer are not reloaded, only the input neurons change, and they are far fewer in number than the number of weights and biases. Therefore, the time it takes for the first load phase to complete will not be accounted for in the calculation of the latency of the digital accelerator. As the operation progresses, the top-level module reads the output once computation is complete, concluding the task. This sequence, within the accelerator ensures streamlined execution, maximizing the efficiency and effectiveness of the its operations, which is illustrated in Figure 7 below.

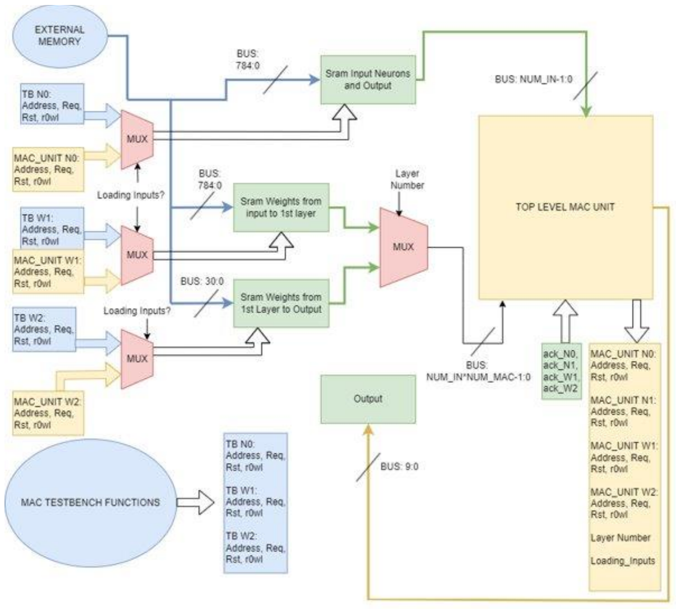

EXTERNAL MEMORY
BUS: 784:0
Sram Input Neurons and Output
BUS: NUM_IN-1:0
TB N0: Address, Req, Rst, r0wl
MUX
MAC_UNIT N0: Address, Req, Rst, r0wl
Loading Inputs?
BUS: 784:0
Sram Weights from input to 1st layer
Layer Number
TOP LEVEL MAC UNIT
TB W1: Address, Req, Rst, r0wl
MUX
MAC_UNIT W1: Address, Req, Rst, r0wl
MUX
BUS: 30:0
Sram Weights from 1st Layer to Output
Loading Inputs?
TB W2: Address, Req, Rst, r0wl
MUX
MAC_UNIT W2: Address, Req, Rst, r0wl
BUS: NUM_IN*NUM_MAC-1:0
ack_N0, ack_N1, ack_W1, ack_W2
MAC_UNIT N0: Address, Req, Rst, r0wl
MAC_UNIT N1: Address, Req, Rst, r0wl
MAC_UNIT W1: Address, Req, Rst, r0wl
MAC_UNIT W2: Address, Req, Rst, r0wl
Layer Number
Loading_Inputs
Output
BUS: 9:0
MAC TESTBENCH FUNCTIONS
TB N0: Address, Req, Rst, r0wl
TB W1: Address, Req, Rst, r0wl
TB W2: Address, Req, Rst, r0wl

Figure 7: Accelerator Diagram

It's important to consider tradeoffs between time, power, and area. Increasing computational efficiency may lead to reduced processing time, or it could potentially require more area on the hardware due to additional resources allocated for optimization. Moreover, while optimizing energy efficiency can result in reduced power consumption, it might also introduce complexities that increase design time and area overhead. Striking a balance between these factors is crucial, as excessive optimization in one area could negatively impact performance in others.

In summary, our proposed approach to a digit recognition hardware accelerator utilizes a three-layer neural network architecture with a proposed strategic variation in the allocation of computational resources, including multipliers and adders, to the Multiply-Accumulate (MAC) units. By dynamically adjusting these resources, we optimize both computational efficiency and resource utilization, tailoring them to the specific requirements of each neural network layer. This innovative approach aims to strike a balance between computational power and resource consumption, ultimately enhancing both accuracy and energy efficiency. The Finite State Machine (FSM) diagram showcases the systematic transition of neurons through the layers, ensuring a methodical flow when processing. Additionally, the accelerator process efficiently manages the loading of parameters into memory and the computation process, maximizing processing speed and resource usage. Through these optimizations, our approach offers a streamlined and effective solution for digit recognition tasks.

IV. Experiment Results

Our digit recognition hardware accelerator employs a three-layer neural network architecture with strategic resource allocation to Multiply-Accumulate (MAC) units. By dynamically adjusting these resources, we optimize computational efficiency and resource utilization tailored to each neural network layer.

Our findings reveal that the number of multipliers in the MAC has minimal impact on area and power consumption but significantly affects latency and accuracy. Specifically, latency decreases notably from one to two multipliers and further with the addition of two more, totaling four multipliers, as shown in Figure 8. However, transitioning from two to four multipliers results in a substantial drop in accuracy from 85% to approximately 52%, as depicted in Figure 9. Thus, while increasing to two multipliers proves beneficial for latency, further increments negatively impact accuracy.

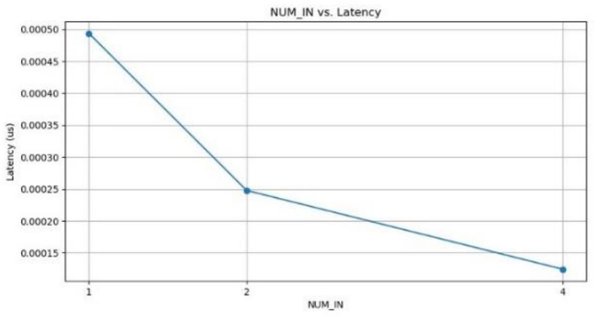


Figure 8: NUM_IN vs Latency

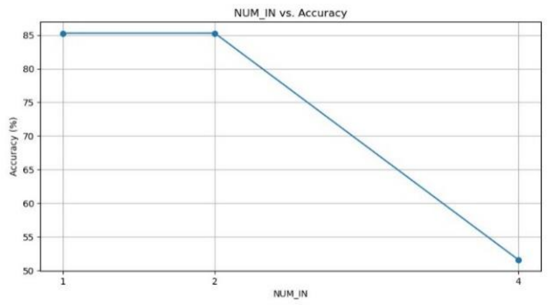
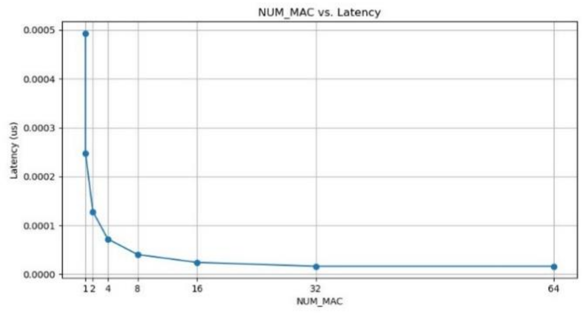


Figure 9: NUM_IN vs Accuracy

Similarly, when increasing the number of adders, we found that there were no significant impacts on accuracy, area, or power. The main impact of increasing the adders was the significant decrease in latency. However, it is important to note that the decrease in latency between 32 and 64 adders was extremely minimal compared to the decrease in latency between 1 to 32 adders.



We were able to run APR on our optimized design as seen below. This was done through trial and error, specifically in altering the density, and placement of I/Os in the TCL script. We also spent considerable time making sure that Makefiles and TCL scripts were in the proper directory so that they could run.

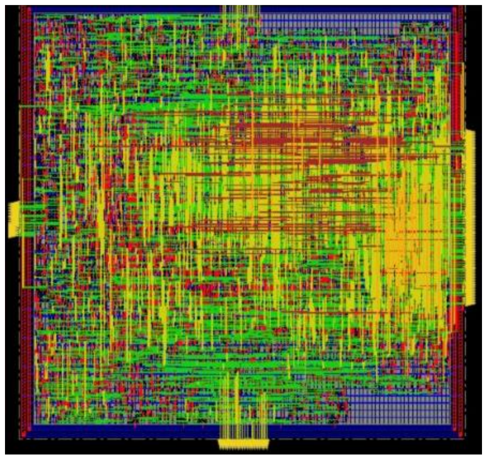


Figure 11: APR Results

When trying to complete both DRC and LVS we experienced issues with VIAs. This is due to the library that is used not being able to handle the level of detail required for our design. Unfortunately, LVS builds off DRC and we could not pass LVS either.

V. Conclusion

In conclusion, this paper introduces a dedicated hardware accelerator aimed at enhancing digit recognition performance using the MNIST dataset. It explores both a baseline design and a proposed optimized design, which strategically varies the number of multipliers and adders in the Multiply-Accumulate (MAC) units. The study discusses the advantages and limitations of this enhanced approach, contributing to ongoing efforts to optimize hardware solutions for efficient and accurate digit recognition tasks.

Acknowledgment

We would like to express our gratitude everyone who contributed to this paper. Our heartfelt appreciation goes to our professor, Dr. Inhee Lee, and TA, Yuyang Li, for their invaluable guidance, support, and encouragement. Furthermore, we extend our thanks to the University of Pittsburgh for providing the necessary resources and facilities, and our peers for their constructive comments and suggestions.

Work Distribution

In the beginning, Mr. Lehrman handled the data path, accelerator, and finite state machine files; Mr. Allabon handled the testbench; and Ms. Bircan handled the sigmoid and top-level files. Over time Mr. Lehrman decided to also write the top-level file, so that he could easily make changes to it without having to go through another person. When Mr. Allabon was having trouble with the test bench, Mr. Lehrman assisted Mr. Allabon in writing a testbench that could interface with the accelerator file. When testing the MAC, it was discovered that the sigmoid file was causing some problems. Mr. Lehrman, Mr. Allabon, and Ms. Bircan spent time debugging and creating a sigmoid function that would work. After considerable testing, Mr.

Allabon took care of the synthesis and the routing of the top-level file. Ms. Bircan took care of the report and presentation.

References

[1] N. Shahid, T. Rappon, and W. Berta, “Applications of artificial neural networks in health care organizational decision-making: A scoping review,” PLOS ONE, vol. 14, no. 2, p. e0212356, Feb. 2019, doi: https://doi.org/10.1371/journal.pone.0212356.

[2] I. Lee, Advanced VLSI Design, “Project 1: Digit Recognition Hardware Accelerator” University of Pittsburgh, Pittsburgh, Feb. 2024