A Study on Efficient Memory Utilization in Machine Learning and Memory Intensive Systems

JONES EMMANUEL
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

A Study on Efficient Memory Utilization in Machine Learning and Memory Intensive Systems

Jones Emmanuel
jo8026em-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Joachim Rodrigues
Masoud Nouripayam
Arturo Prieto

Examiner: Erik Larsson

October 3, 2021

Printed in Sweden
Tryckeriet i E-huset, Lund

Acknowledgement

I would like to thank my supervisors Joachim Rodrigues, Masoud Nouripayam, and Arturo Prieto for giving me this opportunity and supporting me through this thesis. I am very grateful for the valuable feedback and guidance you provided. A special thanks to my examiner Erik Larsson and my opponent Vignajeth Kuttuva Kishorelal for all the helpful feedback provided during the presentation. Finally, I would like to express my deepest gratitude to my family and friends for the motivation and support you provided.

Abstract

As neural networks find more and more practical applications targeted for edge devices, the implementation of energy-efficient architectures is becoming very crucial. Despite the advancements in process technology, power and performance of memories remain to be a bottleneck for most computing platforms. The aim of this thesis is to study the effect of the breakdown structure of memories on power cost with a focus on a dedicated hardware accelerator in neural network applications. The evaluation test suite of this study consists of a RISC-V based System-on-Chip (SoC), PULPissimo, integrated with an accelerator designed for a convolutional neural network (CNN) application. The memory organization of the CNN hardware accelerator is implemented as a flexible and configurable wrapper for studying different breakdown structures. Moreover, different optimization techniques are also utilized to put area and power costs in the defined design budget. Multiple memory breakdown structures, suitable for the accelerator's memory sub-system, were analyzed for power consumption, as part of the study. The study reveals a non-linear increase in power consumption with the size of the static random-access memory (SRAM) modules and the limits of memory partitioning while using SRAM. It also revealed the power and area limitations of D flip-flop based standard cell memory (SCM) in comparison with SRAM.

Popular Science Summary

The field of artificial intelligence is making fast progress these days. It is finding more practical applications like image recognition, self-driving cars, healthcare, and speech recognition that are relevant in day to day life. New smartphones are the best example of devices that are capable to perform machine learning tasks that find everyday use. In 2017, Google released a new smartphone app called Lens. This app, using the phone's camera and machine learning techniques, can perform many image recognition tasks such as identifying plants or animals, reading and translating text, and scanning QR codes or barcodes. Another example is the voice assistants available with phones or smart speakers. The speech recognition systems for these applications also use machine learning to classify the words spoken by the user. But for all these applications to work, a stable connection to the internet is necessary. The data is sent to powerful data centers located in the cloud to perform the demanding computations. However, with the rising concerns about privacy and due to bandwidth limitations, it is desirable to perform these tasks on the device itself instead of sending it over to a data center. This approach requires devices powerful enough to process the complex machine learning algorithms but highly energy-efficient at the same time so that they can still be powered by batteries.

Rapid draining of batteries on today's laptops, smartphones, smartwatches, and many other portable devices can be attributed to several reasons. One of the reasons is the internal memory of such battery-powered devices. Despite the advancements in technology and design architecture, the memories consume a significant amount of the energy budget. Hence, in order to achieve better battery life on such devices, it is important to have an energy-efficient memory sub-system. This thesis aims to analyze different memory configurations to improve energy efficiency for the memory sub-system in a hardware designed for image classification using machine learning.

List of Acronyms

NN	Neural networks
SRAM	Static random-access memory
SCM	Standard cell memory
SoC	System-on-chip
CNN	Convolutional neural network
PnR	Place and route
ANN	Artificial neural network
MLP	Multi-layer perceptron
ReLU	Rectified linear units
CMOS	Complementary metal-oxide-semiconductor
WL	Word Line
BL	Bit-line
CMAC	Convolution multiply-accumulate
P14	Pixel 14
MAC	Multiply-accumulate
IO	Input/Output
kB	Kilobytes

CS	Chip select
VCD	Value change dump
ROM	Read-only memory

Table of Contents

1 Introduction **1**
- 1.1 Scope of Thesis . . . 1
- 1.2 Related Work . . . 2
- 1.3 Thesis Structure and Organization . . . 3

2 Background **5**
- 2.1 Artificial Neural Networks . . . 5
- 2.2 Memory . . . 8

3 Accelerator Design and Implementation **11**
- 3.1 CNN Model . . . 11
- 3.2 Optimizations . . . 12
- 3.3 Architecture . . . 14
- 3.4 Verification . . . 22

4 Memory Analysis and Results **23**
- 4.1 Memory Configurations . . . 23
- 4.2 Primary Analysis . . . 24
- 4.3 Post-Layout Power Analysis . . . 27
- 4.4 Additional Analysis . . . 28

5 PULPissimo Integration **29**
- 5.1 Accelerator Integration and Functional Verification . . . 29

6 Conclusion **33**
- 6.1 Future Work . . . 34

List of Figures

2.1 Basic architecture of a single layer neural network [22]. 6
2.2 Basic architecture of a multi-layer neural network [22]. 6
2.3 SRAM architecture [31]. 9

3.1 Architecture of the CNN implemented in the hardware accelerator. . . 12
3.2 Simplified block diagram of the accelerator. 14
3.3 Simplified architecture of the convolution unit. 16
3.4 Visualization of data reuse using the pixel registers. 17
3.5 Visualization of weights and pixels used in each cycle to compute one output pixel during convolution. 18
3.6 Sequence of movement of filter along the image and corresponding output pixels during convolution. 19
3.7 Simplified architecture of the dense unit. 21

4.1 Visualization of address space division in some of the configurations. 25
4.2 Power per word access gain for the SRAM modules compared to a 0.0625kB SRAM module. 27

5.1 Simplified architecture of PULPissimo with the CNN accelerator attached [39]. 30

List of Tables

4.1 Details of the memory configurations analyzed. 24
4.2 Post-synthesis results. 25
4.3 Post-layout results. 28
4.4 SRAM vs SCM comparison. 28

Chapter 1
Introduction

In recent years, there has been immense growth in the field of machine learning and neural networks (NN). The wide utilization of neural networks in a variety of applications such as speech recognition [1], sign language translation [2], face detection [3], healthcare [4], self-driving [5], and voice assistants [6] demands looking for flexible approaches capable of fulfilling current efficiency requirement trends.

NNs are highly complex and expensive in terms of computation, power, and memory. For many NN applications on edge devices [7] the common approach is to transfer the data to the cloud to perform the neural network related computations. However, this approach comes with its own drawbacks such as the requirement of a stable internet connection, privacy concerns over sensitive data, and the bandwidth limitation of the internet [8][9].

Due to these concerns, for many NN applications, processing at the edge devices would be more efficient [7]. But edge devices are resource-constrained with limited memory and power budget. As NNs find more and more applications, the need for efficient low-power devices capable of implementing complex NNs is on the rise [10].

One of the big contributors to power consumption in NN hardware is memory. In fact, the memory can be the bottleneck of the hardware in terms of power, performance, and area [10][11]. Having an efficient memory sub-system is very essential for low-power edge devices. This master's thesis project aims to explore and study efficient memory breakdown structures for low-power dedicated hardware accelerators at the edge.

1.1 Scope of Thesis

The goal of this project is to analyze different memory breakdown structures, to find and propose an efficient solution to be implemented in a NN hardware with an aim to reduce power consumption. These memory breakdown structures can be comprised of conventional static random-access memory (SRAM) and/or standard cell memory (SCM). The NN hardware consists of a hardware accelerator integrated into the open-source RISC-V based System-on-chip (SoC), PULPissimo

[12][13].

To this purpose, a convolutional neural network (CNN)[14] was designed and trained to classify the images of handwritten digits from the MNIST database [15]. MNIST database is a collection of handwritten digits that is commonly used to train various machine learning networks and acts as a base case for new implementations. The CNN model was constructed and optimized for the specified accuracy, performance, and memory requirements.

The next task was to design and implement a hardware accelerator for the inferencing of the CNN. The accelerator uses fixed-point numbers for reduced energy and area consumption. A behavioral model of CNN was developed in MATLAB with both the floating-point and fixed-point numbers. This model was used to analyze the network data and the intermediate results to find the optimum parameters for quantization and the word length for the hardware implementation.

The memory sub-system of the accelerator has a generic design to facilitate the evaluation of other memory techniques in the future. Based on the structure, organization, and size of different memory units within the accelerator, different breakdown structures are selected for the analysis. These breakdown structures are then integrated into the accelerator and synthesized with a 28nm process technology. Later, based on post-synthesis results, selected configurations are pushed into the place and route (PnR) flow for more realistic power analysis.

1.2 Related Work

There have been studies about using memory hierarchy and partitioning to achieve better performance and power numbers. Su and Despain explored both vertical and horizontal partitioning as low power cache design techniques in [16]. The idea behind the vertical cache partitioning was to create hierarchical levels, whereas horizontal partitioning was to use segmented memory. With both the partitioning combined, and for certain cache parameters, they were able to achieve 90% power savings.

The usage of hierarchical designs and partitioned memories is discussed in a comprehensive survey on technologies, architectures, and optimization techniques for energy-aware embedded memory designs [17]. Partitioned memories within a vertical hierarchical level will help to reduce the power consumption at the expense of area and wiring overhead, and communication power.

Meinerzhagen et al. [18] explore different SCM architectures for area and power compared with a base case scenario of an SRAM implementation. This report shows that SCMs compared to SRAMs can provide up to 37% power savings, however, with 50% area overhead. In this case, the memory blocks were larger than 3 kilobits, and at those sizes, SRAM tends to have a lower area than SCM. The analysis shows that using SCM allows the place and route tool to place the

memories near the logic, thereby reducing the routing length.

1.3 Thesis Structure and Organization

This thesis work is organized into 6 chapters.

Chapter 1: Introduction, provides an insight into the problem, how this study is relevant to the problem, and an outline about the target system.

Chapter 2: Background, explains the theory behind NN, CNN, embedded memory, and power consumption.

Chapter 3: Accelerator Design and Implementation, discuss the CNN model training and testing, various optimization techniques used for the hardware implementation of the CNN, architecture of the design, and the working of the core units of the accelerator.

Chapter 4: Memory Analysis and Results, explains the study on the memory breakdown structures and the results.

Chapter 5: PULPissimo Integration, discuss the process of integrating the CNN hardware accelerator to PULPissimo.

Chapter 6: Conclusion, presents the final analysis and the future plans regarding the study.

Chapter 2
Background

This chapter discusses the background of NN, CNN, embedded memory, and power consumption considered in this study.

2.1 Artificial Neural Networks

First introduced in the 1940s, artificial neural networks (ANN) or neural networks are one of the earliest examples of a machine learning model. These networks are inspired by the structure and operation of the human nervous system. NNs find wide use in pattern recognition, visual scene interpretation, and speech recognition [19][20][21].

The basic structure of a NN consists of interconnected computational units which are commonly referred to as neurons. Each of these connections is scaled with an intermediate parameter known as weight. NNs use weights to compute a mathematical function at the neuron from its inputs. Figure 2.1 shows the basic architecture of a simple NN. It is by changing the magnitude of the weights, the learning in a NN occurs. The data employed for training provides feedback to the network about the correctness of the weights based on the extent to which the predicted output matches the output label for the corresponding input. Based on the feedback, the weights in the network are changed in a mathematically justified way to modify the computed function, so that errors are reduced and future predictions become more accurate. By repeating this process over multiple iterations with many different input-output pairs of data, and by adjusting the weights between the neurons in the network successively, the neural network is refined over time, to make more accurate predictions [20][22].

Neural networks can be single-layer or multi-layer networks, where a layer is a set of neurons that compute the same output function. A single-layer network, also referred to as a perceptron is shown in Figure 2.1, where x_1 to x_5 represent the feature variables or input nodes, w_1 to w_5 represent the weights, and b represent a bias value. It consists of an input layer and a single node output layer. The input nodes or neurons are directly mapped to the output using a generalized linear function. The input layer does not perform any computation. The edges between the input nodes and the output node have the weights, to which the feature values

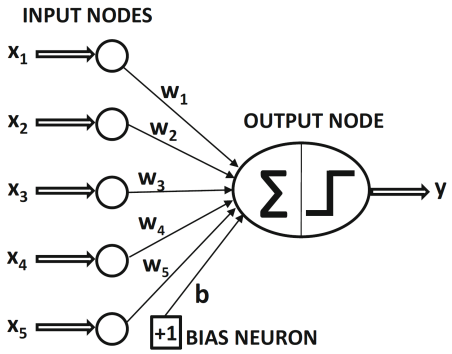

Figure 2.1: Basic architecture of a single layer neural network [22].

from the input layer are multiplied and the products are added at the output node. In order to capture any invariant part of the prediction, an additional bias value might be used. The computed value after the bias addition is then passed through an activation function to the output [22].

Multi-layer networks or multi-layer perceptron (MLP) have one or more intermediate computation layers known as hidden layers in between the input and output layers. These additional layers will help the network to learn more complex functions. Each layer has a separate activation function, which adds nonlinearity to the output of a neuron. Figure 2.2 shows the basic architecture of a multi-layer neural network [22].

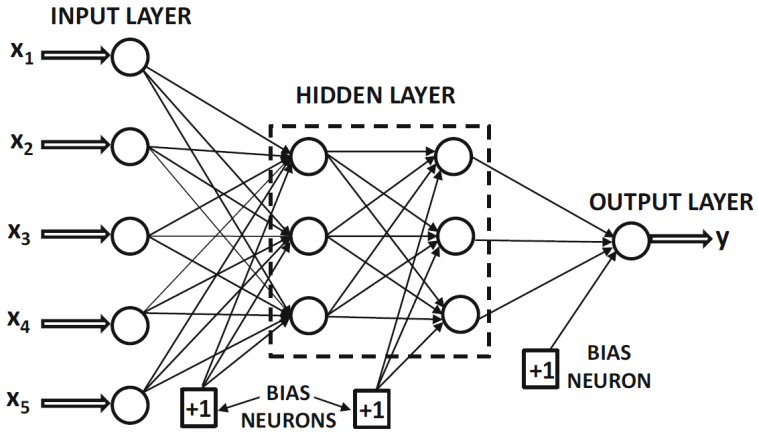

Figure 2.2: Basic architecture of a multi-layer neural network [22].

One critical part of a neural network design is the selection of the activation functions. The use of nonlinear activation functions is significant for multi-layer networks as they help to increase the modeling power of a network. Depending

on the target function of the network, different activation functions such as sigmoid, hyperbolic tangents, rectified linear units (ReLU), or softmax can be used in different layers. These are generally part of the neurons. The output node in Figure 2.1 has two parts. The first one shows the summation of the products from multiplying the weights and input feature values and the bias value. The second one denotes the nonlinear activation function [22].

2.1.1 Convolutional Neural Networks

CNN use the mathematical operation convolution in at least one layer, to process grid-structured inputs with strong spatial dependencies. The time-series data which can be considered as a 1-dimensional grid with samples at regular intervals, or an image which can be considered as a 2-dimensional grid of pixels are some examples for such grid-structured data [22][23].

In a convolutional neural network, there are three types of layers: convolutional, ReLU and pooling. In addition, there is a final set of fully connected layers which are mapped to the output nodes [22].

Convolutional Layer

The convolutional layer performs the convolution operation between weights and inputs. In the convolutional layer, weights are organized as sets of 3-D arrays known as filters or kernels. In spatial dimensions, the filters are generally square-shaped and have much smaller dimensions compared to the input on which they are applied. The convolution operation is a dot product between the weights and the matching input pixels to generate a 2-dimensional output, known as feature maps or activation maps. Every filter will generate an activation map arranged along the depth dimension to generate the full output of a convolutional layer. The depth of the output feature map for a layer will be defined by the number of the filters used in that layer [22][24][25].

ReLU

Generally, the convolution is followed by an activation function that adds nonlinearity to the network. One popular choice of the activation function in the hidden layers is the ReLU function represented by equation 2.1. This function is applied to all the output feature values before passing them to the next layer and is generally considered to be a part of the convolutional layer [22][26].

$$f(x) = max(0, x) \tag{2.1}$$

Pooling

The function of the pooling layer in a convolutional neural network is to merge reasonably similar features into one output feature. The pooling operation reduces the spatial dimensions of the feature maps while keeping the same depth. The pooling operation works on small patches of a feature map. The size of these

patches is decided by the kernel. In max-pooling, the type of pooling used in the CNN for this study, for each of those small patches, the maximum value of the neurons is forwarded to the output.The most common setting for pooling is max-pooling with a kernel size of 2x2 and a stride of 2 [22][23][24].

Fully Connected Layer

The fully connected layers are stacked between the convolutional layers or the pooling layers and the output node. The first fully connected layer is formed by connecting each neuron from the last spatial layer to each neuron in the fully connected layer. It is possible to use more than one fully connected layer. Since the fully connected layers connect all the neurons of one layer to the next layer, these layers will dominate the number of weights used in the entire network [22].

Generally, the last fully connected layer uses the softmax activation when the final output is meant to be a k-way classification, where k denotes the number of classes or the number of nodes in the output layer. It is used to predict the probabilities of each class. It converts a vector of k real values into a probability distribution. The softmax function is very complex for implementation in hardware. To reduce the complexity, many alternatives and approximations have been proposed in different works.

2.2 Memory

Despite the improvements in memory density and interface speed, the new multiprocessing systems experience a performance bottleneck due to memory bandwidth, known as the "Memory Wall" [27]. In many advanced processing systems, the memory sub-system and the clocks are the most power-consuming domains. As far as edge devices are concerned, energy is one of their crucial requirements. With the constraints becoming stringent and performance requirements getting stronger driven by more sophisticated applications, the design of low-power systems will be a challenge to the SoC architects and designers. Reducing memory power with the use of new technologies and clever architectures will play a crucial role in achieving the target constraints.

In complementary metal-oxide-semiconductor (CMOS) technology, the power consumed by its gates can be broken down into three components. Dynamic power, short-circuit power, and static power. Dynamic power consumption is of special importance for this study. It is caused by the charging and discharging of the capacitances present at the output of the gate when it switches from logic '0' to '1' and vice-versa. These capacitances are comprised of the parasitic capacitances internal to the transistors, the load capacitance, and the capacitance of the wire connecting the load to the gate output. Dynamic power is also dependent on the number of transitions per second and the supply voltage, and can be represented by equation 2.2 [28].

$$P_{\mathrm{dyn}} = \alpha C {V_{\mathrm{DD}}}^2 f \tag{2.2}$$

Where C represents the total capacitance, i. e. $C_{internal} + C_{wire} + C_{load}$, V_{DD} represents the supply voltage, f represents the frequency at which the circuit is working and α is a constant representing the switching factor for the circuit.

Any memory that is integrated within the chip along with the logic can be termed as embedded memory. High performance and efficient embedded memories are a critical element in the modern SoC and ASIC designs as it helps to reduce the off-chip data transfers. Among the many different embedded memory techniques, SRAM is the most commonly used one. The main advantage of SRAM over other options like Flash or DRAM is its process compatibility with logic and faster operation. The basic building blocks of the memory are the storage elements, also known as memory cells or bitcells. These storage elements are capable of storing a single bit of data. Compared to a SCM which uses a flip-flop or latch as the storage element, SRAM has a smaller bitcell leading to a better density. This helps to pack more storage in a given real estate and when it comes to modern memory-hungry architectures and applications, having a faster and larger on-chip memory is critical [10][17][18][29][30].

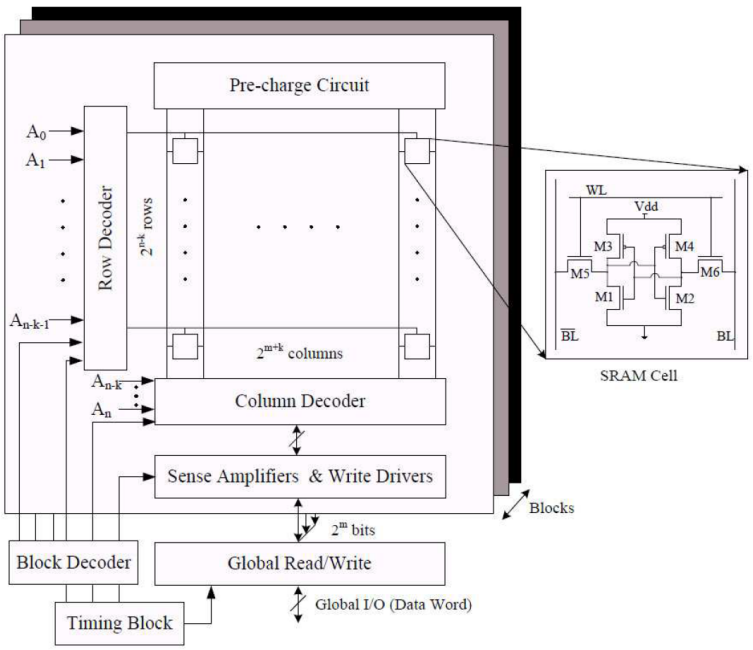

Figure 2.3: SRAM architecture [31].

A complete memory unit consists of an array of storage elements along with some peripheral circuits. In the case of an SRAM, these peripheral circuits include address decoders, sense amplifiers, and pre-charge circuits which assist the read/write operation of the memory array. Figure 2.3 shows the classic SRAM architecture with a standard 6-Transistor SRAM cell. The bitcell consists of a

pair of cross-coupled inverters (M3, M1 and M4, M2) and two access transistors (M5 and M6). The cross-coupled inverter pair holds the single bit of data and its complement between them, whereas the access transistors control the read/write operations into the cell. The word line (WL) signal, connected to the gates of the access transistors, selects the cell for an operation and the data transfer happens through the bit-lines (BL and $\overline{\text{BL}}$). Pre-charge circuits are used to pre-charge the bit-lines to a voltage level before the read operation. Sense amplifiers, on the other hand, amplify the signal from the bitcell into a voltage level as required by the IO. Due to design limitations on the capacitance and resistance of the bit-lines and word lines, the bitcells are arranged in multiple blocks of 2-dimensional arrays to achieve the required size. The address will then be split into 3 parts, one to select the block, one to select the row, and one to select the column. Decoders are used to reduce the number of the address lines [28][31].

The bit-line capacitance is a crucial factor in determining the power consumption and the speed of operation of the SRAM. Dynamic power consumption is directly proportional to the capacitance (equation 2.2) and hence an increase in bit-line capacitance will result in an increase in dynamic power, and thereby the total power of the SRAM. Similarly, the higher capacitance will lead to a larger propagation delay affecting the performance of the SRAM adversely. Due to these factors, in some cases, it will be beneficial to break down a larger memory into many smaller memories in a single and continuous address space. This thesis evaluates such breakdown structures, which can help to reduce the power consumption.

SCM, as another type of memory, uses a flip-flop or latch as the storage element. The memory has an array of flip-flops/latches and a set of peripheral circuitries to handle the read/write operations. When compared with a typical SRAM bitcell, the SCM bitcell is larger in size. Despite the bigger storage element, the SCM has an advantage when it comes to memories with smaller storage capacity. The SRAM memories require a lot more peripheral circuits compared to SCM, like the sense amplifiers and logic for pre-charge. Due to these additional circuitries, SRAM memories tend to consume more area than SCM when the required storage capacity is small. However, when the storage size increases, SRAM has a clear advantage [18][32].

Compared to SRAM, SCM has other advantages too. Since it uses the standard cells from the library and can be described using a hardware description language, it is more flexible to design compared to SRAM. The next advantage for SCM comes during the place and route phase. SRAM blocks need to be manually placed by the designer. The SCMs on the other hand can go through the standard placement process and can be placed very close to the computation blocks. By placing the SCM near the computation blocks, the routing length from memory to logic blocks can be reduced, which in turn can help to reduce the energy consumption [18][32].

Chapter 3

Accelerator Design and Implementation

This chapter presents the details about the CNN model selected for implementation, the architecture and design of the accelerator, and various optimizations used.

3.1 CNN Model

The first major task in the implementation of the hardware accelerator is to find a hardware-friendly CNN with reasonably high accuracy. By increasing the number of layers in the network or by using more kernels per convolutional layer, very high accuracy can be achieved. But this requires more computations and memory accesses. Hence, it is very important to balance the computations, memory accesses, area, and accuracy in the accelerator aimed for low-power hardware.

To find a suitable network for this study, different networks with different number of layers, or different kernel parameters, were trained and analyzed for the storage requirement, computations and resulting throughput of the system, and the accuracy. The initial training and analysis of the networks were performed using Python and Keras. Based on the results, four networks were selected for further analysis. In the next stage, a behavioral model was developed in MATLAB with floating-point numbers, and each of the networks was analyzed for accuracy and the range of values of the intermediate results. These networks were further analyzed using fixed-point numbers.

Based on the results after the fixed-point behavioral model analysis, the network for the implementation in the accelerator was chosen. The selected network, shown in Figure 3.1, has only one convolutional layer and 3x3x8 kernels. The convolutional layer accepts an input of size 28x28 pixels and produces an output of size 26x26x8. It is followed by a ReLU layer which is then followed by a max-pooling layer with a filter size of 2x2 which reduces the size of the feature map into 13x13x8 pixels. The output from the max-pooling layer is then flattened into a 1352x1 matrix and is fed into the fully connected layer, which converts the 1352

neurons into 10 neurons. The softmax function in the last layer converts the values of 10 neurons into a vector of 10 values, with sum 1. This set of final values can be interpreted as the probability for each digit.

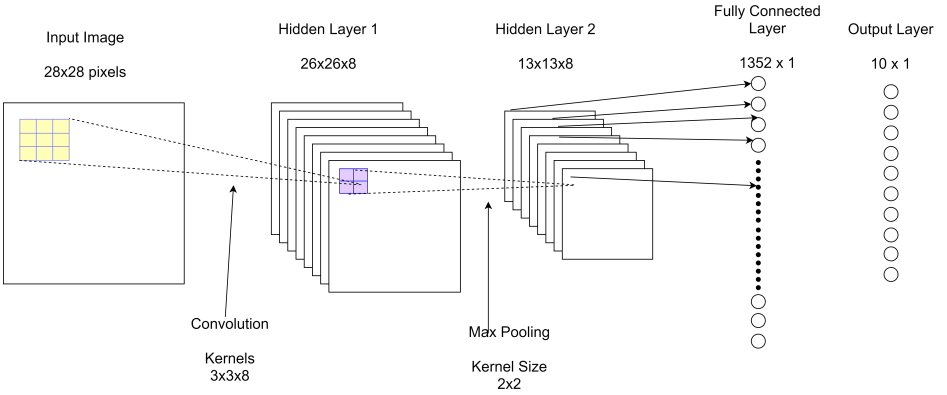

Figure 3.1: Architecture of the CNN implemented in the hardware accelerator.

3.2 Optimizations

To improve the performance of the accelerator, and to conserve area and power, certain optimizations were implemented. Since both the computations and memory accesses consume a significant portion of the power budget of the hardware, it is important to reduce both these processes, but without affecting the accuracy drastically. Similarly, the major chunk of area for any modern-day hardware is consumed by the memory. Reducing the total memory size will thereby help to reduce the area and cost and also the static power consumption of the hardware.

3.2.1 Fixed-Point Computation and Quantization

One method generally used in hardware implementations to reduce the resource requirements is the fixed-point arithmetic system. Compared to a floating-point system, a fixed-point system is less complex to implement, faster in computations, and less power and area consuming. In [33] Hashemi et al show that using an 8-bit fixed-point arithmetic system reduces power by 85%, and area by 80% compared to a floating-point system with no accuracy loss for classification on the MNIST database.

One critical step in implementing the fixed-point system is quantization. This process maps the numbers from a large and continuous set to a smaller and discrete set reducing the precision, and hence the word length and the memory requirement. Quantization is necessary due to the fact that the network was trained using 32-bit floating-point numbers and hence the final weights exported are all represented in 32-bit floating-point numbers. But the process of quantization will

lead to a class of errors commonly known as quantization errors. These errors can adversely affect the accuracy of the network. Hence, the quantization parameters such as word length and the position of the decimal point must be fixed after proper analysis so that the loss in accuracy is minimized [34].

The weights and input pixels for the network are quantized into 8-bit fixed-point numbers using the built-in functions available in MATLAB. The word length for the intermediate layers is set to 16 bits as 8 bits were not sufficient to represent the full range of values producing an overflow. The position of the decimal point changes depending on the layer and was set based on the analysis of the values at each stage.

3.2.2 Removal of Softmax

Softmax function includes exponential and division operations and its implementation in hardware is complex and computationally challenging. Many different approximations have been suggested over the years, but those were with the aim to find the final probability of each class. If the probability is not a requirement, the softmax layer can be removed completely. To find the final classification result, only a simple logic to find the output node with the maximum value in the last fully connected layer is required. The output nodes will have a range of values including negative, zero, and positive values. Among these nodes, the one with the maximum value is likely to be the final class. This implementation is efficient in terms of area, power, and throughput as it can save a lot of hardware and is faster. Since the requirement for this project was only the classification, the softmax function in the software model was removed and a logic to find the maximum values in the final fully-connected layer was added. Prior to the hardware implementation, the design was verified in MATLAB and was ensured there is no accuracy loss.

3.2.3 Memory Optimization

Two main approaches were taken to optimize the power consumption in memory utilization. Reducing the storage requirement, and increasing the data reuse.

To reduce the storage requirement, the accelerator was designed to consume the pixels generated in one stage by the next stage immediately. The pixels generated by one computation stage are sent directly to the next stage which then performs the required processing. This reduces the required storage between the stages significantly. The parallelization of computations, the order in which convolution operation generates the output pixels, and the organization of weights within the memory enabled this optimization.

Data reuse can help to reduce memory transactions significantly. Convolution operation on an image requires pixels to be processed multiple times. Instead of reading the pixels from the main memory every time, reading the pixels from local registers can help to reduce the power consumption. In this accelerator, a set of registers called pixel registers are used to store the pixels temporarily. Pixel

registers combined with the convolution algorithm used in the system significantly improves the data reuse in the system.

3.3 Architecture

The hardware accelerator was designed to be integrated within PULPissimo and to perform the classification of a 28x28 pixel image into one of the 10 classes assigned for the digits 0 to 9. It was designed with a dedicated memory system and a set of computational units for the efficient implementation of the selected convolutional neural network. The memory sub-system was designed with an aim to improve data reuse and to reduce the routing distance between the memory and the computational units. Special care was taken to design the memory sub-system with some flexibility to try different memory breakdown structures. Computation units and the data flow through different layers were designed to reduce memory usage for storing results of the intermediate layers.

The general architecture of the accelerator is shown in Figure 3.2. The convolution unit performs the convolution operation on the input image whereas the dense unit handles the operations of the fully connected layer. The max-pooling unit is responsible to perform max-pooling on the output from the convolution unit. Finally, the max-finder unit finds the final class from the values of the 10 neurons of the output layer. The convolution unit and the dense unit have specific memory sub-units to store the input image and the weights. The weight loader unit is responsible to direct the weights coming from the CPU into the respective memories. Each of the blocks will be discussed in detail in the coming subsections.

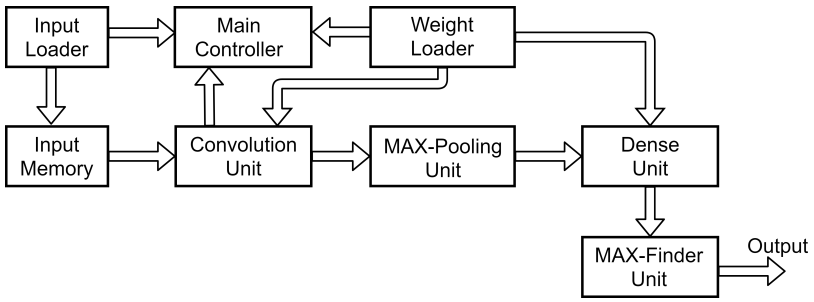

Figure 3.2: Simplified block diagram of the accelerator.

3.3.1 Weight Loader and Input Loader

The weight loader unit is responsible to load the weights received from the CPU into the right memory sub-units, keeps a track of the weights received, and informs the main controller when all the weights have been loaded into the memory.

Input loader is a simple module to handle the input pixels received. It redirects the pixels into the input memory and keeps a track of the number of pixels it

received. It also informs the main controller and the convolution controller once all the pixels have been sent to the input memory.

3.3.2 Main Controller

This module handles the busy status signal of the accelerator based on the status of loading weights, input pixels, and the convolution operation. The CPU monitors this signal to decide when to send an input image into the accelerator for processing. During initialization, the main controller asserts the busy signal until all the weights are loaded into the memories. This is to ensure that the input pixels are not sent for processing before the accelerator has all the weights required for the computation.

Once all the weights are loaded, the busy signal is de-asserted, so that the CPU can send in the pixels of the input image one by one. When all the pixels are loaded, the busy signal is asserted until the convolution operation is completed.

3.3.3 Convolution Unit

The convolution unit consists of two different memory sub-systems, one for input and the other for weights. For computation, it has 8 convolution multiply-accumulate (CMAC) units, along with 8 registers to store the intermediate results. Each of these CMAC units consists of 3 multipliers and 2 adders. In addition, the convolution unit has 8 adders for bias addition, 8 ReLU units to perform the non-linear ReLU activation function, and a controller for tracking and managing the convolution operations. The architecture of the convolution unit is shown in Figure 3.3. For simplicity, only datapaths are shown and the control modules and control paths are omitted.

Since the input image has 784 pixels with each pixel of size 8 bits, the input memory is designed with a word length of 8 bits and has a total capacity of 784 words. A memory controller module controls the read and write operations to the memory and is responsible for the address translation during the read operation. The input memory sub-system is designed in such a way to enable using both SCM or SRAM as the storage unit with minimum modifications. Additionally, while using SRAM as the storage unit, the memory can use either a single SRAM or a combination of smaller SRAMs.

The convolution weight memory sub-system has two parts. One for storing the main weights used for convolution and another used to store the bias weights used in the bias adder. Main weight memory consists of 8 banks of size 3x24 bits each. The convolutional layer has 8 filters of size 3x3, and each weight within the filter is 8 bits long. Each bank has a word length of 24 bits (3 weights) and is connected directly to only one CMAC unit. This ensures that each computation unit is associated with a separate memory bank and always accesses the same bank. This arrangement helps to place the computation unit and the associated memory bank as close as possible. All the banks are accessed in parallel so that in

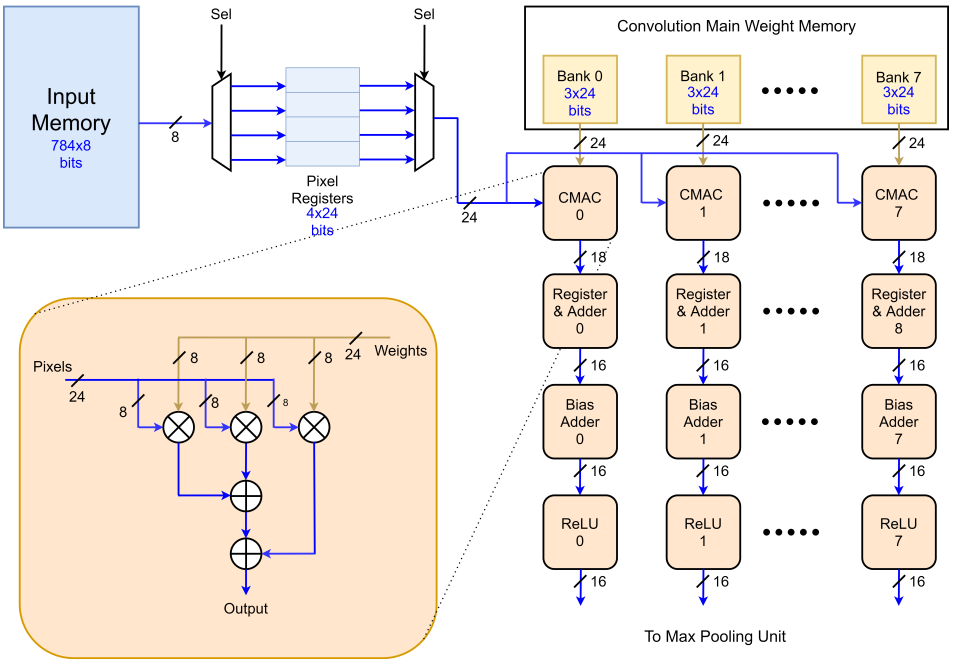

Figure 3.3: Simplified architecture of the convolution unit.

a cycle, 24 weights will be sent from the memory to the multipliers to enable 24 parallel multiplications. Since the convolution main weight memories are small in size, they are realized using registers. Bias weights are stored in registers added within the bias adder module to make sure those are closer to the adders that use the register contents.

The main computation unit has twenty-four 8-bit multipliers, 3 each for 8 channels, and is partitioned into 8 individual CMAC units with each having 3 multipliers and 2 adders. In addition, there are 8 registers to store the intermediate results and another 8 adders to add the values in the registers with the output from the CMAC units. The final output from the computation unit is truncated into 16 bits. Bias adder modules perform the bias addition on the neurons for all 8 channels in parallel. The ReLU operation is also performed on all 8 channels in parallel using 8 ReLU units.

The convolution controller is responsible to read the image pixels and weights, track and manage convolution operation, and control the CMAC units. It also ensures synchronization and informs the main controller when a convolution operation is completed. A set of 4 registers called the pixel registers, is integrated into the convolution controller. Each pixel register is 24 bits wide and can store 3 pixels of the input image at a time. These registers serve two purposes. One, it enables 3 multiplications per channel in a single cycle, thereby increasing parallelization. The second purpose is to improve the data reuse and to reduce the read operations

on the input memory. Once a pixel is read into the pixel registers, it can be reused as many as 6 times depending on its position in the original image. This helps to reduce the memory reads. Figure 3.4 shows the contents of the pixel registers when each output pixel is generated. Pixel 14 (P14) is a good example to show the data reuse, and hence it is highlighted to make the tracking easy. P14, once read from the input memory, is shifted within the pixel register and is used in 6 computations to produce output pixels from R0 to R5.

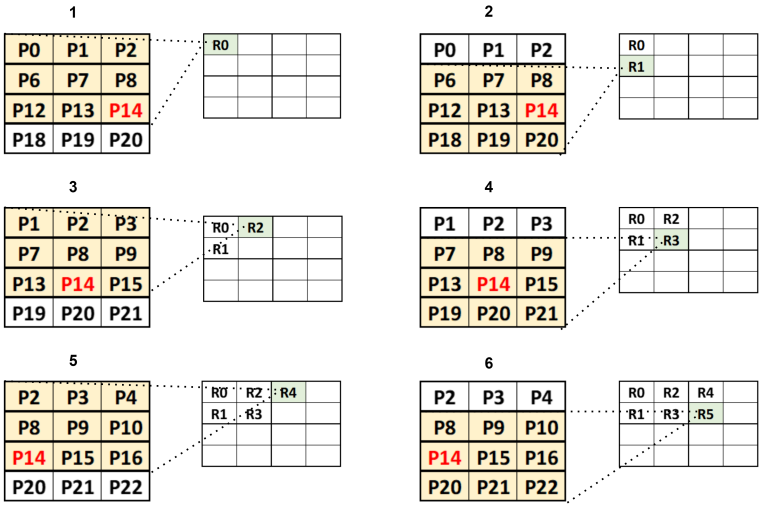

Figure 3.4: Visualization of data reuse using the pixel registers.

Operation

As mentioned before, the convolution process is completely managed by the convolution controller. Since the filter size is 3x3 per channel, to produce one output pixel per channel, 9 multiplications are required. Since there are only 3 multipliers allocated for a channel, it takes 3 cycles to compute one output pixel. For simplicity, the operations for one channel are described below. The same set of operations are performed in parallel for all the 8 channels.

At first, the pixel registers in the convolution controller are loaded with the initial 3 pixels from rows 0 to 3 of the image. After the initial loading, in the first cycle of computation, 3 image pixels are multiplied with the corresponding weights from the first row of the filter, and the result is accumulated and stored in the register for intermediate results. A visualization of weights and pixels selected for multiplication in each cycle is shown in Figure 3.5. The weights and pixels sent to the multipliers in a particular clock cycle are highlighted in red. For simplicity, the sample image in the figure is truncated to a size of 6x6 pixels. Figure 3.5 shows that in cycle 1 of the computation, weights W0, W1, and W2 along with pixels P0, P1, and P2 are selected to compute the partial sum as shown in equation 3.1,

which is then stored in the register.

$$Partial\_Sum = W0 * P0 + W1 * P1 + W2 * P2. \tag{3.1}$$

In cycle 2 (as in figure 3.5), weights W3, W4, and W5, and pixels P6, P7, and P8 are multiplied and the result is added to the previous result stored in the register. The partial sum generated in this cycle is then stored in the register. In cycle 3 (as in figure 3.5), the last set of image pixels are multiplied with the last row of weights and the result is added to the value in the register to get the final value of the output pixel. Equation 3.2 shows all the weights and pixels used to compute the first output pixel.

$$\begin{aligned} Final\_Pixel\_Value = {} & W0 * P0 + W1 * P1 + W2 * P2 \\ & + W3 * P6 + W4 * P7 + W5 * P8 \\ & + W6 * P12 + W7 * P13 + W8 * P14 \end{aligned} \tag{3.2}$$

This result is then truncated into 16 bits and sent out to the bias adder. The bias adder module then adds the bias value to the pixel in the next clock cycle and sends the result to the ReLU unit which performs the ReLU activation function on the pixel. Both the bias addition and the ReLU operation happens in the same clock cycle. The result from the ReLU unit is then forwarded to the max-pooling unit.

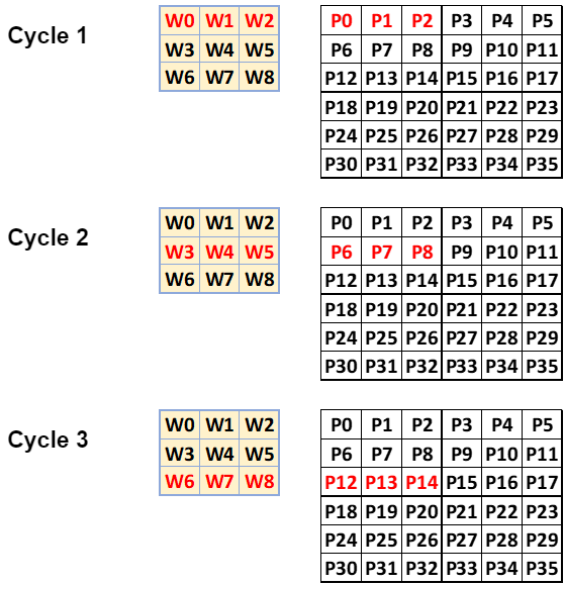

Figure 3.5: Visualization of weights and pixels used in each cycle to compute one output pixel during convolution.

As mentioned earlier, the accelerator was designed to reduce memory usage by consuming the pixels as soon as they are generated. In the convolutional layer,

this idea was implemented by designing the convolution operation to produce a set of 4 pixels required for a max-pooling operation consecutively. This way, the pixels will be consumed by the max-pooling unit and is not required extra storage. Figure 3.6 shows the sequence in which the filter is moved along the image and the corresponding output pixels. In step 1, the output pixel R0 is produced when the filter is at the very beginning of the image covering the pixels in rows 0 to 2 and columns 0 to 2. In the next step, the filter moves down by 1 row and generates the output pixel R1. In step 3, when the output pixel R2 is produced, the filter covers rows 0 to 2 and columns 1 to 3. Finally, in step 4 the filter moves down again by 1 row to generate the output pixel R3. Now by step 4, all the 4 pixels required for one max-pooling operation are produced. By moving the filter along the image in this sequence of steps, it can be ensured that the batch of 4 pixels required for a max-pooling operation will be produced in succession. So the max-pooling unit can consume the pixel as soon as it is produced and only needs a single 16-bit register to store the temporary result.

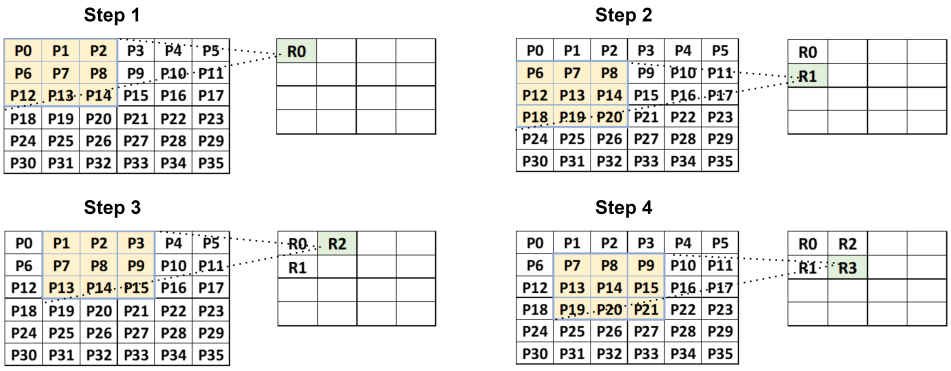

Figure 3.6: Sequence of movement of filter along the image and corresponding output pixels during convolution.

3.3.4 Max-Pooling Unit

The max-pooling unit consists of 8 max-pooling sub-modules, with one sub-module allocated for each channel. Each sub-module mainly consists of a comparator, a multiplexer, a register to store the intermediate results, and a counter. It implements max-pooling operation on the output from the convolution and ReLU operation and downsizes the spatial dimensions of the feature map by 75%.

Operation

The max-pooling unit performs the max-pooling operation on the pixels from all 8 channels in parallel. For simplicity, the operations for one channel are described here. The max-pooling sub-module gets its input from the ReLU unit. Since all the 4 pixels required for a max-pooling operation are received successively, the max-pooling sub-module just needs to keep track of the number of pixels and perform

a comparison on these pixels. At the beginning of a max-pooling operation, the max-pooling sub-module stores the first pixel into its register. For the second and third pixels, the stored pixel is compared with the incoming pixel. If the value of the incoming pixel is higher, then the content in the register is replaced with the incoming pixel. If not, the pixel in the register will be retained. For the fourth pixel, it again compares the pixel in the register and the incoming pixel and will send the pixel with the larger value to its output. In the same cycle, the register will be cleared to prepare for the next max-pooling operation.

3.3.5 Dense Unit

The dense unit, shown in Figure 3.7, performs the operations for the fully connected layers in the network. It consists of two memory sub-systems for storing the weights, 10 multiply-accumulate (MAC) units for computation, 10 registers to store the intermediate results, a controller unit, and 10 bias adders. It also has a set of 8 registers to store the results from the max-pooling unit.

Similar to the convolution weight memory sub-system, the dense weight memory sub-system also consists of two parts. The first one is for the main weights and the second for the bias weights. Main weight memory consists of 10 banks of word length 8 bits and each of these banks is connected to only one multiplier. While loading the weights only one bank is accessed at a time. But during computation, all the 10 banks are read in parallel so that 10 weights will be sent to all the 10 multipliers in a cycle. The bias weights are stored in registers integrated within the bias adder unit.

Dense main weight memories are the biggest memory units within the accelerator and these memories provide good opportunities to experiment with different memory breakdown structures.

The computation unit has 10 MAC units with each having a 16x8 multiplier. Along with these, there are 10 registers to store the intermediate results. All the data flow and operations associated with the dense layer are controlled by the dense controller unit. Other responsibilities of this unit include tracking the address for weights, handling the data received from the max-pooling unit, and sending out necessary synchronization signals to the max-finder unit. For storing the data sent out from the max-pooling unit, the dense unit has eight 16-bit registers known as the max-pool buffer.

Operation

The dense controller receives the pixels for all the 8 channels from the max-pooling unit and stores them in eight 16-bit registers. It sends one pixel at a time to the MACs thereby completing one set of operations in 8 cycles. Each pixel is sent to all the 10 multipliers and is multiplied with 10 different weights read from the 10 banks in the dense main weight memory. The result from each of the multipliers

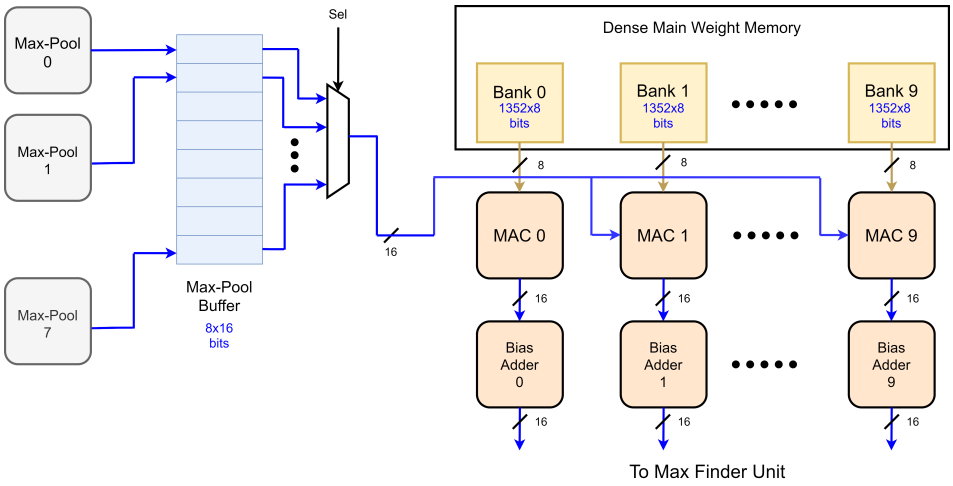

Figure 3.7: Simplified architecture of the dense unit.

is then added to the contents of the intermediate result registers and are stored back into these registers.

Once all pixels are processed, the values from each of the 10 registers are truncated into 16 bits and are sent to the bias adders. Bias adders then add the bias value to all the 10 pixels and send the results to the max-finder unit.

3.3.6 Max-Finder Unit

This module is responsible to find the neuron with the maximum value that defines the result digit. It has a comparator, a counter, 10 registers to store the results from the dense unit, and 2 registers to store the temporary results of the maximum value and the index of the neuron that has the maximum value.

Operation

The max-finder unit saves the results from the dense unit into ten 16-bit registers and then finds the maximum value out of those 10 neurons. Initially, it compares the values of first and second neurons and saves the higher value into the register for the temporary maximum value. Depending on which neuron has the maximum value, the corresponding index, i. e. either 0 or 1, is stored into the register for the temporary index value. Here the index of the neurons is in the range 0 to 9, with each index pointing to the respective digit. From cycle 2 onwards, it compares the value in the register for the temporary maximum value, with the value of the next neuron. If the value of the neuron exceeds the value in the register, then the register is overwritten and the register for the temporary index is updated. This process is continued until all the 10 neurons are processed. The final results forwarded to the output are the index of the neuron with the maximum value and the value of that neuron.

3.4 Verification

The hardware accelerator design was verified using the simulation tool QuestaSim with the help of a testbench designed for the top module. The testbench handled all the file read/write operations for weights, input images, and the final results. The final classification results of the hardware accelerator matched with the results from the MATLAB fixed-point model resulting in an accuracy of 98.25%.

Chapter 4

Memory Analysis and Results

This chapter presents the study results of various memory breakdown structures (or configurations). The analysis was done only for the memory dedicated to the dense layer (hereafter, called dense memory) of the CNN model introduced in previous chapters. The selection of dense memory for this study is due to the fact that its size allows for the study of a wider range of configurations. The primary analysis was with synthesis results and a selected set of configurations were pushed into the PNR flow for more realistic power and area analysis. All the ASIC design steps have been done using ST 28nm FD-SOI process technology with the PVT parameters of typical-typical at nominal 25C, and low threshold voltage transistors. The target clock frequency for the synthesis was 200 MHz.

4.1 Memory Configurations

A total of 12 different memory configurations, using only SRAM, were analyzed in the initial part of the study. Each configuration creates a continuous address space from a combination of different SRAM sizes arranged within a wrapper, which works as an Input/Output (IO) interface between memories and other computational logics. As mentioned before, the dense memory consists of 10 banks with each bank covering 1352 Bytes (1352x8 bits). The selected set of SRAM sizes for this study comprises of 1024x8, 512x8, 256x8, 128x8, and 64x8 bit macros. Since all the macros have the same word length, the representation is done in terms of size in Kilobytes (kB). In addition to these 12 SRAM based configurations, a few additional configurations comprised of both SRAM and SCM were analyzed, and the details will be presented towards the end of this chapter.

Table 4.1 shows the analyzed SRAM memory configurations. All the configurations were designed such that the size of one bank is 1408 words. This is the minimum size that can be satisfied with the available SRAMs and can meet the minimum size requirement of 1352 words.

A separate wrapper was designed for each configuration in which the required SRAM modules were instantiated. All these wrappers had identical interfaces and the same total size of 1408x8 bits. The interface consists of an 8-bit data-in bus, another 8-bit data-out bus, an 11-bit address bus, together with the clock, reset,

Table 4.1: Details of the memory configurations analyzed.

Configuration	Configuration Breakdown	Total Modules
config0	1kB, 0.25kB, 0.125kB	3
config1	1kB, 3x0.125kB	4
config2	2x0.5kB, 0.25kB, 0.125kB	4
config3	2x0.5kB, 3x0.125kB	5
config4	0.5kB, 3x0.25kB, 0.125kB	5
config5	0.5kB, 2x0.25kB, 3x0.125kB	6
config6	0.5kB, 0.25kB, 5x0.125kB	7
config7	0.5kB, 3x0.25kB, 2x0.0625kB	6
config8	5x0.25kB, 0.125kB	6
config9	11x0.125kB	11
config10	0.5kB, 2x0.25kB, 2x0.125kB, 2x0.0625kB	7
config11	2x0.5kB, 2x0.125kB, 2x0.0625kB	6

and the chip select (CS) signals. The wrappers have additional logic to enable the CS signal of the right SRAM module based on the address range. The address also acts as the select signal for the data out multiplexer and the selector for the data in and address buses. Figure 4.1 shows how the address space is divided among different SRAM modules in some of the configurations. Numbers on the left side of each block shows the address range for that block.

4.2 Primary Analysis

Since all the 10 banks of the dense memory have the same access pattern (all banks have the same number of read and write operations), multiple configurations were analyzed at the same time. Each configuration is evaluated in one bank of the dense memory. This arrangement made it possible to run the analysis for a maximum of 10 configurations at a time and helped to make the procedure efficient and fast. The power consumption for all configurations was analyzed carefully up to the post-synthesis phase. Post-synthesis power analysis was done with Synopsys PrimeTime tool and a value change dump (VCD) file of length 1 ms.

Post-synthesis results are presented in Table 4.2. Config2 is the most efficient configuration in terms of power, whereas config0 is the most efficient in terms of the number of modules and area. Config9, a case for extreme partitioning has shown inefficiency in terms of both area and power. It is the most power and area-hungry configuration and acts as the baseline for the comparison.

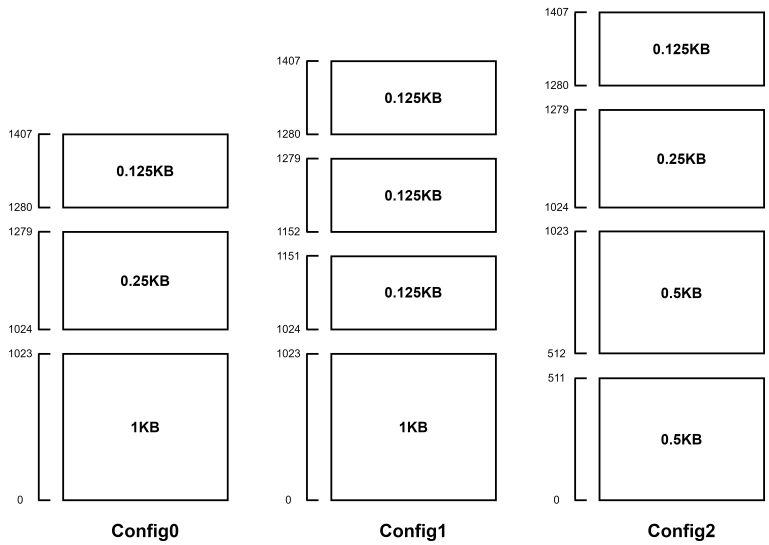

Figure 4.1: Visualization of address space division in some of the configurations.

Table 4.2: Post-synthesis results.

Configuration	Configuration Breakdown	Power Gain	Area Gain
config0	1kB, 0.25kB, 0.125kB	15.43%	59.30%
config1	1kB, 3x0.125kB	11.73%	52.04%
config2	2x0.5kB, 0.25kB, 0.125kB	26.54%	50.87%
config3	2x0.5kB, 3x0.125kB	22.22%	43.56%
config4	0.5kB, 3x0.25kB, 0.125kB	23.46%	43.61%
config5	0.5kB, 2x0.25kB, 3x0.125kB	19.14%	36.33%
config6	0.5kB, 0.25kB, 5x0.125kB	15.43%	29.06%
config7	0.5kB, 3x0.25kB, 2x0.0625kB	19.14%	36.38%
config8	5x0.25kB, 0.125kB	20.99%	36.40%
config9	11x0.125kB	0.00%	0.00%
config10	0.5kB, 2x0.25kB, 2x0.125kB, 2x0.0625kB	15.43%	29.08%
config11	2x0.5kB, 2x0.125kB, 2x0.0625kB	18.52%	36.34%

Config0 uses just 3 SRAM modules to reach the size of 1408x8 bits. But, in terms of power consumption, it is the 8th best configuration and provides only 15.13% gain compared to config9, whereas config2, the most power-efficient configuration provides 26.54% gain. When it comes to area, config0 provides 59.3% gain as against the 50.87% gain provided by config2.

The only difference between config0 and config2 is the use of two 0.5kB SRAM modules in config2 instead of one 1kB SRAM module used in config0. The higher power for the 1kB SRAM module can be attributed to a larger bit-line capacitance. As per equation 2.2, the dynamic power is directly proportional to the capacitance which in this case is the bit-line capacitance. In the case of the 0.5kB SRAM module, the bit-lines are smaller in length and fewer transistors are connected, resulting in less wire capacitance and parasitic capacitance leading to significant power savings.

However, this observation does not stay true for all the sizes. Comparison between config2 and config4 provides more details. The second 0.5kB SRAM module in config2 is split into two 0.25kB SRAM modules for config4. This change has resulted in a decrease in power gain by 3.1%. This result does not follow the trend observed in the case of config0 and config2. The reason for the increase in power in the case of two 0.25kB SRAM modules despite having a smaller bit-line capacitance is the auxiliary circuits. For example, the sense amplifiers and other read support circuits like the pre-charge circuit will remain the same irrespective of the bitcell array size as long as the word length is the same. Compared to the 0.5kB SRAM module, in the 0.25kB SRAM module, the share of these sense amplifiers and other support circuits of the overall power budget will be higher. This trend applies to the address decoder as well. Even though the decoder becomes smaller and uses fewer transistors, there are still two distinct decoders for the two 0.25kB modules resulting in an increase in the net count of transistors. Due to these facts, the savings gained by splitting the bitcell array into two can be less than the extra power required for the additional auxiliary circuits such as decoders, sense amplifiers, and other drivers. This is the reason behind the increased power consumption in config4 compared to config2. However, in the case of config0 and config2, the savings from splitting the bitcell array is more than the extra power consumed by the additional set of auxiliary circuits, thereby providing net savings.

This observation is reiterated in the case of config9, which uses eleven 0.125kB SRAM modules. Despite having a small bitcell array and bit-line capacitance, the power consumption by eleven sets of auxiliary circuits is dominating, making it the most power-hungry configuration. The same observation applies to the area as well.

This trend in the change in power consumption with the size of the SRAM module is evident from the power gain data for the SRAM modules also. Figure 4.2 shows the power per word access (total power of an SRAM module divided by the number of words in the module) gain for all the SRAM modules compared to a 0.0625kB SRAM module. For the access patterns in this hardware accelerator,

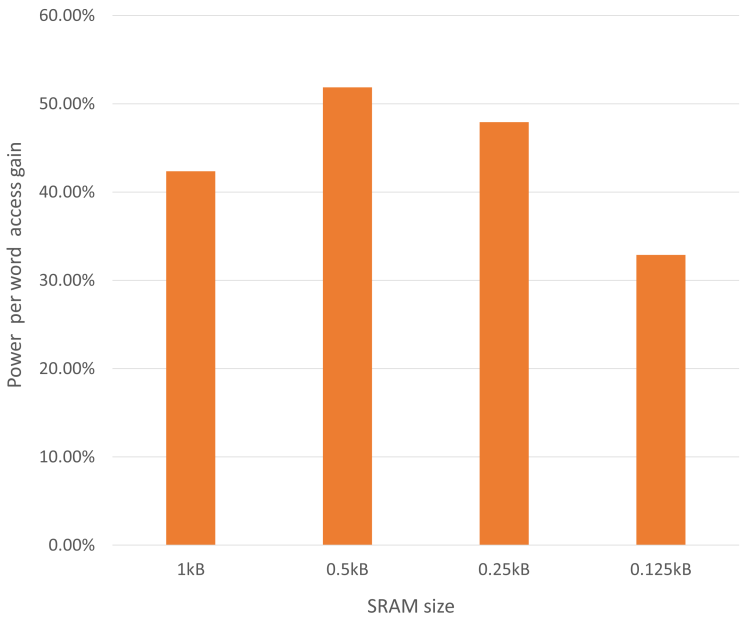

Figure 4.2: Power per word access gain for the SRAM modules compared to a 0.0625kB SRAM module.

0.5kB is the most efficient macro size with the highest power per word access gain, followed by 0.25kB. This is reflected in the power numbers of the different configurations as well. Config2 makes the best use of the most efficient modules, and as a result, has become the most efficient configuration in terms of power.

4.3 Post-Layout Power Analysis

Following the primary analysis, five configurations were selected for further analysis using the post-pnr netlist and the parasitic data for more realistic power numbers. For place and route, the tool Innovus from Cadence was used. The post-pnr netlist was then verified and then the power was analyzed using Primetime and a 1ms long VCD file. The results are presented in the Table 4.3.

The post-pnr results follow the same trend as the post-synthesis results. Config2 is the most efficient configuration in terms of power consumption whereas, config0 is the most area-efficient configuration. For both area and power, config9 is the least efficient one. Config2 provides 34.66% gain in power and config0 provides 25.57% gain compared to config9. Config2 provides 49.13% gain in terms of area compared to config9, whereas config0 provides 57.31%. Based on the results from the post-pnr analysis, config2, the most power-efficient configuration is selected for the final implementation.

Table 4.3: Post-layout results.

Configuration	Configuration Breakdown	Power Gain	Area Gain
config0	1kB, 0.25kB, 0.125kB	25.57%	57.31%
config1	1kB, 3x0.125kB	19.89%	50.28%
config2	2x0.5kB, 0.25kB, 0.125kB	34.66%	49.13%
config9	11x0.125kB	0.00%	0.00%
config11	2x0.5kB, 2x0.125kB, 2x0.0625kB	23.86%	35.11%

4.4 Additional Analysis

Since the smaller SRAM modules were not so efficient in terms of area and power, few additional configurations made up of a combination of SRAM and SCM were analyzed. The SCM used for this analysis had D flip-flop based storage elements and did not have any clock gating. However, these configurations turned out to be much more expensive than the SRAM only configurations.

One of these configurations was pushed into the PNR stage for more realistic analysis. This configuration is similar to config11 and uses two 0.5kB SRAM modules and SCM for memory modules of size 0.125kB and 0.0625kB. The results are shown in Table 4.4. In terms of both power and area, the SRAM only configuration proved to be much better. SRAM only configuration has a whopping 95.71% gain in power and 52.44% gain in area compared to the configuration with SCM.

Table 4.4: SRAM vs SCM comparison.

Configuration	Configuration Breakdown		Power Gain	Area Gain
	SRAM	SCM		
config11	2x0.5kB, 2x0.125kB, 2x0.0625kB	0x0	95.71%	52.44%
config26	2x0.5kB	2x0.125kB, 2x0.0625kB	0.00%	0.00%

More studies are required in the SCM design space as part of future work. Even though the SCM generally tends to have fewer and smaller auxiliary circuits [18], large storage elements made up of D flip-flops have resulted in excess power consumption. Instead of using a D flip-flop, a more area and power-efficient latch-based storage element could provide substantial savings in both area and power. Furthermore, adding clock gating techniques can present a significant effect on power reduction. An SCM with these changes needs to be analyzed to see if it can replace some of the smaller SRAM modules for a more power-efficient solution.

Chapter 5
PULPissimo Integration

The PULP platform was started as a joint project between ETH Zürich and the University of Bologna with an aim to develop an open-source, scalable ultra-low-power processing platform. It is a RISC-V instruction set architecture based implementation. PULPissimo is a SoC that can be configured to use multiple cores from the PULP platform. The version of PULPissimo used in this project has a single core of 32-bit RI5CY [35] as its main core. PULPissimo supports the logarithmic interconnect [36], an efficient I/O subsystem using μ-DMA [37], hardware accelerators [38], and a wide variety of peripherals such as camera interface, QSPI, JTAG, GPIO, I2C, I2S, etc. PULPissimo includes 512KB of L2 memory and a Read-only memory (ROM) for storing the boot code. The low latency logarithmic interconnect supports multiple access ports and enables direct memory access to the L2 memory for the μ-DMA subsystem or the optional hardware accelerator. The hardware accelerator can be connected directly to the logarithmic interconnect, or through the AXI plug which supports the AXI4 or AXI-Lite protocol. Figure 5.1 shows the simplified architecture of PULPissimo with the CNN accelerator.

5.1 Accelerator Integration and Functional Verification

There are two methods for the integration of custom-developed hardware accelerators into the PULPissimo. The first method, as proposed in [38], is more suitable when the accelerator requires frequent access to the L2 cache memory. The datapath of the accelerator is connected to the logarithmic interconnect whereas the control is memory-mapped and accessed through the peripheral interconnect. The connection to the logarithmic interconnect ensures that the accelerator has direct access to the L2 memory.

The second method is to connect the accelerator to the parameterized AXI crossbar within the soc interconnect. PULP platform supports both AXI4 and AXI-Lite standards and provides support for the protocol conversion [40]. Since the CNN accelerator used in this project has its own dedicated memory and does not need frequent access to L2 memory, it was connected to the system using an

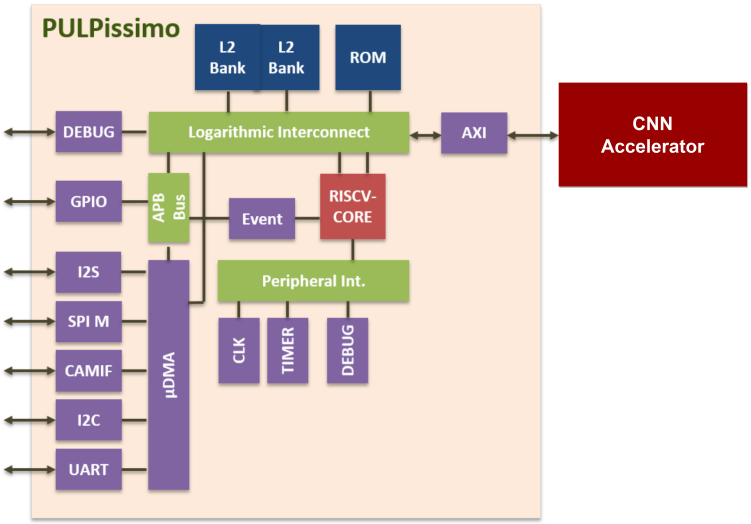

Figure 5.1: Simplified architecture of PULPissimo with the CNN accelerator attached [39].

AXI-Lite bus with a 32-bit data bus.

To connect the accelerator with the soc interconnect via the AXI-Lite bus, an axi interface module was designed. The module has six 32-bit registers, out of which, three act as status registers, one as a control register, one as a data input buffer to store the data sent from the core, and the final one as a buffer to store the output from the accelerator. Apart from handling the protocol for the interface, it also handles the data transfer between the buffer registers and the accelerator. Each of the registers is mapped to a unique address within PULPissimo's address space.

Depending on the value written in the control register, the subsequent input to the interface module from the core will be sent to either the weight loader unit or the input loader unit of the accelerator with the appropriate request signals. The busy signal of the accelerator is mapped to the busy status register in the interface module, which the core can read. This signal is set to '1' on two occasions. First, this signal is asserted immediately after the initialization and is kept high until the accelerator receives all the weights. The second occasion is when the accelerator is running the convolution operation on an image. At the end of processing an image, the accelerator sends the result to the interface module, which then stores it in its result buffer and changes the result status to '1' to show that a valid result is available.

The SoC integrated with the accelerator was tested and verified using a custom C program and behavioral simulations. The C program handles the loading of weights and inputs into the accelerator from a file, monitoring the status registers,

and reading and displaying the final output. The program starts with loading the weights after the initialization sequence of the system. Once all the weights are loaded, it starts sending the input pixels of the first image. While sending data to the accelerator, it monitors the write-status register to properly time the data to prevent any data loss.

Once the input pixels are sent, the program monitors the busy-status register and when the busy-status becomes '0', it sends the pixels for the next image. After sending the pixels for the second image, the program checks the value in the result-status register. If the value is '1', then the program will read the result of the first image from the result register and displays it.

Chapter 6
Conclusion

As neural networks become popular and find new practical use cases for battery-powered edge devices, the need for power-efficient hardware is on the rise. This project aimed to study and define a memory breakdown structure for improving the energy efficiency of low-power neural network hardware implementations. For this study, a hardware accelerator was designed, optimized, and integrated into an open-source low-power PULPissimo SoC. The accelerator performs inference of a CNN, capable of classifying the images of handwritten digits from the MNIST database.

To reduce the area and power cost and improve the performance, various optimization techniques were employed. By using the quantization and fixed-point computations, memory requirement was reduced by 75%. An efficient implementation of convolution optimized the memory utilization by reducing the frequency of memory write-back for the intermediate results. The use of multiple banks enabled higher bandwidth and power efficiency for memories. Finally, to generate the final output, the computation-intensive softmax activation function was replaced with a simple algorithm to find the neuron in the output layer with the maximum value.

The presented study led to some interesting observations about memory breakdown structure as a combination of different memory macro sizes. The power consumption for the SRAM macros is not directly proportional to the size. Among the SRAM macros used for the study, the most power-efficient one is the macro of size 0.5kB (512x8 bits) closely followed by the one with the size of 0.25kB (256x8 bits). But, "the smaller, the more efficient" approach does not work for all sizes. In fact, going below a threshold will result in much higher power consumption. The smallest SRAM module with the size of 0.0625kB (64x8 bits), had the highest power per word access consumption. Compared to this one, the most power-efficient SRAM module with the size of 0.5kB and the largest SRAM module with the size of 1kB (1024x8 bits) provided 51.85% and 42.36% gain in power per word access, respectively.

For the dense memory in the accelerator, the most power-efficient memory breakdown configuration consists of two 0.5kB, one 0.25kB, and one 0.125kB SRAM macros. The most area-efficient configuration, on the other hand, is a

breakdown structure as a combination of 1kB, 0.25kB, and 0.125kB SRAM macros. An extremely partitioned configuration evaluated involving eleven 0.125kB SRAM modules turned out to be a highly expensive configuration in terms of both area and power. Compared with the extremely partitioned configuration, the most power-efficient configuration provides 34.66% power gain and the most area-efficient configuration provides 25.57% power gain. Through the study, it was learned that for SRAM based memories, partitioning can help to reduce the power consumption to some extent, but partitioning to extremely small modules will create a reverse effect.

Since the smaller sized SRAMs proved to be very expensive, a few additional configurations involving a combination of SRAM and D flip-flop based SCM were analyzed to evaluate an improvement in power consumption. However, the configurations with the SCM turned out to be more power and area-consuming than the SRAM only configurations, due to the much larger storage elements compared to the SRAM bitcells.

6.1 Future Work

There are many opportunities to explore and expand this study in the future. Among those, the first one would be to explore an SCM with smaller latch based storage elements and with clock-gating techniques enabled for both read and write circuits. Such an SCM needs to be evaluated to see if it can reduce the power, to an extent that it can replace some of the smaller SRAM modules.

There are options to explore more in the SRAM only configurations as well. First of all, SRAMs with a larger word length than 8 bits can be tried in the dense memory to see if it helps to improve the power consumption. Secondly, another study can be performed with a larger network with bigger memories, allowing to experiment with more breakdown structures.

Another topic to explore is the integration of the accelerator with PULPissimo. In order to improve the performance and efficiency, the accelerator can be connected directly to the logarithmic interconnect. This will help to remove the AXI-4 and AXI-Lite interfaces, and the protocol conversions to and from AXI-4 and AXI-Lite.

Bibliography

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] B. Fang, J. Co, and M. Zhang, "Deepasl: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017, pp. 1–13.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[4] P. A. Keane and E. J. Topol, *With an eye to ai and autonomous diagnosis*, 2018.

[5] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[6] E. Polyakov, M. Mazhanov, A. Rolich, L. Voskov, M. Kachalova, and S. Polyakov, "Investigation and development of the intelligent voice assistant for the internet of things using machine learning," in *2018 Moscow Workshop on Electronic and Networking Technologies (MWENT)*, IEEE, 2018, pp. 1–5.

[7] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[8] J. Lau, B. Zimmerman, and F. Schaub, "Alexa, are you listening? privacy perceptions, concerns and privacy-seeking behaviors with smart speakers," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–31, 2018.

[9] M. Zhang, F. Zhang, N. D. Lane, Y. Shu, X. Zeng, B. Fang, S. Yan, and H. Xu, "Deep learning in the era of edge computing: Challenges and opportunities," *Fog Computing: Theory and Practice*, pp. 67–78, 2020.

[10] F. Conti, M. Rusci, and L. Benini, "The memory challenge in ultra-low power deep learning," in *NANO-CHIPS 2030*, Springer, 2020, pp. 323–349.

[11] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[12] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, IEEE, 2015, pp. 1–39.

[13] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, "Quentin: An ultra-low-power pulpissimo soc in 22nm fdx," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, IEEE, 2018, pp. 1–3.

[14] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard, "Handwritten digit recognition: Applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, pp. 41–46, 1989.

[15] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[16] C.-L. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: A case study," in *Proceedings of the 1995 international symposium on Low power design*, 1995, pp. 63–68.

[17] L. Benini, A. Macii, and M. Poncino, "Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 2, no. 1, pp. 5–32, 2003.

[18] P. Meinerzhagen, C. Roth, and A. Burg, "Towards generic low-power area-efficient standard cell based memory architectures," in *2010 53rd IEEE International Midwest Symposium on Circuits and Systems*, IEEE, 2010, pp. 129–132.

[19] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[20] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[21] J. Heaton, "Applications of deep neural networks," *arXiv preprint arXiv:2009.05673*, 2020.

[22] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, pp. 978–3, 2018.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[24] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, 2. MIT press Cambridge, 2016, vol. 1.

[25] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[27] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[28] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, *Digital integrated circuits: a design perspective*. Pearson education Upper Saddle River, NJ, 2003, vol. 7.

[29] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE, 2014, pp. 10–14.

[30] S. Natarajan, S. Chung, L. Paris, and A. Keshavarzi, "Searching for the dream embedded memory," *IEEE Solid-state circuits magazine*, vol. 1, no. 3, pp. 34–44, 2009.

[31] K. Tanaka, *Embedded Systems: Theory and Design Methodology*. BoD–Books on Demand, 2012.

[32] O. Andersson, B. Mohammadi, P. Meinerzhagen, A. Burg, and J. N. Rodrigues, "Ultra low voltage synthesizable memories: A trade-off discussion in 65 nm cmos," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 6, pp. 806–817, 2016.

[33] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, IEEE, 2017, pp. 1474–1479.

[34] R. Goyal, J. Vanschoren, V. Van Acht, and S. Nijssen, "Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms," *arXiv preprint arXiv:2102.02147*, 2021.

[35] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[36] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters," in *2011 Design, Automation & Test in Europe*, IEEE, 2011, pp. 1–6.

[37] A. Pullini, D. Rossi, G. Haugou, and L. Benini, "μDma: An autonomous i/o subsystem for iot end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, IEEE, 2017, pp. 1–8.

[38] F. Conti, P. D. Schiavone, and L. Benini, "Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[39] *T. P. Team. PULPissimo: Datasheet.* `https://github.com/pulp-platform/pulpissimo/blob/master/doc/datasheet/datasheet.pdf`, Accessed: February, 2021.

[40] A. Kurth, W. Rönninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini, "An open-source platform for high-performance non-coherent on-chip communication," *arXiv preprint arXiv:2009.05334*, 2020.

Printed by Tryckeriet i E-huset, Lund 2021

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2021-850
http://www.eit.lth.se