Near Data Processing for Power-Efficient CNN in a RISC-V Processor Platform

VIGNAJETH KUTTUVA KISHORELAL
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Near Data Processing for Power-Efficient CNN in a RISC-V Processor Platform

Vignajeth Kuttuva Kishorelal
vignajeth@gmail.com

Department of Electrical and Information Technology
Lund University

Supervisor: Joachim Rodrigues

Examiner: Erik Larsson

November 16, 2021

© 2021
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Convolutional Neural Networks (CNNs) have become a widely used deep learning algorithm in recent years on applications such as image recognition, face classification and object detection. CNN is a computation-intensive application that requires large memory bandwidth and high power. This becomes a challenge for System on Chips (SoCs) as they have power budget constraints and limited memory capacity.

This thesis uses the concept of Near Data Processing (NDP) to efficiently execute CNN on power and memory bandwidth bound devices. The basic idea behind NDP is to keep computation tedious applications very close to memory for reducing data movements. Since high data transfers increase memory footprint and power dissipation.

Execution flow and memory reads occurring in CNN inference were carefully inspected. An NDP unit is designed based on the analysis to reduce data migrations by methodically reusing data reads. The developed NDP unit is then stowed near to memories of open-source microcontroller Pulpissimo.

The base scenario of running CNN inference on a bare Pulpissimo is analyzed with a target scenario of running CNN using NDP unit. The study focuses on power, area, and performance. Furthermore, NDP unit is hardware optimized by quantizing neural network parameters to save computations and data reads.

Reducing precision helped in decreasing bandwidth and memory space by a factor of 4 as neural network parameters of 32-bit were transformed to 8-bits by quantization technique called Truncation. However, a negligible loss in classification accuracy of 0.6% was encountered.

For one inference, NDP unit was able to execute at a speed of 33x compared to running a scalar program of CNN in Pulpissimo. Energy consumed by Pulpissimo with NDP unit is only 1.1% of the total energy of a base scenario. Moreover, the NDP unit covered just 0.8% of the total area of Pulpissimo.

Popular Science Summary

Consumer Electronics has encountered radical advancements in usability and intelligence over a couple of decades. They have transformed how people live, work and make decisions. This is primarily due to technological advancement that occurred in the semiconductor industry. The improvements in fabrication techniques and processes helped to pave way for designing more complex Integrated Circuits(ICs) that are smaller and faster.

The blooming semiconductor industry has supported researchers to execute computer programs that involve complex algorithms. This created distinct advancements in the field of Machine Learning (ML). ML provides intelligence and user-friendliness like reminding some daily tasks, recommending products that match the taste of consumers and voice assistance. As complexity of algorithms increases, the demand for larger memory bandwidth and space raises. This surge drives the need for a better solution to execute ML algorithms effectively.

Reducing memory utilization helps in lowering power expenditure as fewer data reads and writes occur. This reduction in power offers a great benefit for power-constrained devices running these algorithms. In addition, trimming memory space can help in scaling down the SoCs area, which will result in bringing down the manufacturing cost.

Considering these perspectives, the thesis focuses on creating an NDP unit that can effectively run CNN inference by utilizing minimal memory bandwidth and space. The designed NDP unit is triggered by the processor when it needs to classify an image. The study on the designed NDP unit is done by comparing it with the execution of CNN inference in a low-power microcontroller.

Acronyms

CNN Convolutional Neural Networks
NN Neural Networks
NDP Near Data Processing
IC Integrated Circuit
ML Machine Learning
SoC System On Chip
AI Artificial Intelligence
MNIST Modified National Institute of Standards and Technology
PULP Parallel Ultra Low Power
EEES Energy-efficient Embedded Systems
IIS Integrated Systems Laboratory
MCU Micro-Controller Unit
HAL Hardware Abstract Layer
FNN FeedForward Neural Networks
RNN Recurrent Neural Networks
JTAG Joint Test Action Group
µDMA Micro Direct Memory Access
FLL Frequency Lock Loops
ROM Read Only Memory
MAC Multiplier and Accumulator
NMC Near Memory Computing

Table of Contents

List of Figures

Chapter 1
Introduction

Machine Learning (ML) is a branch of Artificial Intelligence that can classify or predict a result by learning from data. Deep Learning a subset of ML, attempts to mimic the way a human brain processes and analyses information. This capability has enabled software applications to make use of Deep Learning extensively over the years.

System on Chips(SoCs) is the backbone to run ML applications on electronic devices like smartwatches, mobile phones, and tablets. They consume considerable amount of power to provide processing and memory needs.

As electronic devices function with limited battery capacity and require a recharge at a periodic intervals of time, it becomes necessary to skillfully utilize the power. The demand for higher data processing to handle ML workloads is increasing over time. It opens up new challenges in Integrated Circuits(ICs) design to optimize the power efficiency of SoCs for improving performance per watt.

1.1 Background

Over a few decades, computing performance had a steep increase obeying Moore's law [1] and following Dennard Scaling [2]. However, the bandwidth and latency of memory technologies have not been able to match computing performance growth. The demand for higher bandwidth and lower latency has led to the concept called memory wall [3], making memories a bottleneck in high data-movement applications. One of the approaches to tackle memory wall is Near Data Processing (NDP). NDP is a subset of Near Memory Computing (NMC) [4]. NDP has been in existence since the 1990s [5] but was not popular due to the limitation of IC fabrication technologies. The concept behind NDP is to place computation tedious applications very close to memory to reduce data movements. After all, high data movements are primary contributors to power consumption and memory bandwidth usage. The impact of NDP in increasing performance and energy efficiency has been proven effective in Deep Learning [6, 7, 8]. It is also noticed that when an NDP unit is used for convolution [9] it achieved speedup in computation and reduction in power consumption.

One of the widely used Deep Learning networks that uses convolution as a base algorithm is Convolutional Neural Networks (CNN). CNN was first introduced by Yann LeCun in the 1980s [10] and is one of the most thriving Deep Learning algorithms so far. It performs convolution operations to extract essential features from input data. The elements can be edges, texture, shape, or any information relevant to CNN. CNN has high data transfers between the processing unit and memory. Its memory access patterns are haphazard, giving additional data movements.

The NDP unit designed for CNN is trained specifically to classify numerical digits in an image. The dataset used for this purpose is the Modified National Institute of Standards and Technology (MNIST) dataset. The dataset contains 60,000 training images and 10,000 test images of digits with each image of size 28x28.

Open source microcontroller architecture called Pulpissimo is used to integrate the NDP Unit and to evaluate its performance. It is developed by a group named Parallel Ultra Low Power (PULP) platform. The PULP platform is a joint effort between Integrated Systems Laboratory (IIS) of ETH Zürich and Energy-efficient Embedded Systems (EEES) group of University of Bologna.

1.2 Thesis Goal

This thesis intends to design an Application Specific Integrated Circuit (ASIC) architecture for an NDP unit integrated into the memory sub-system of an MCU platform. It should make efficient use of data, to perform CNN Inference.
To successfully achieve this target, the thesis is divided into five significant tasks as follows.

1. **Developing a CNN Model**
   The first objective in this thesis is to train a CNN model that classifies handwritten images of numerical digits using MNIST dataset. The size and number of layers to be used are examined, activation unit after each layer is also investigated. Furthermore, appropriate size of neural network model that is reasonable to implement and run inference in hardware is selected.

2. **Designing a hardware optimized model**
   Running CNN model in floating point number representation consumes more power and area. Even though there is high precision in computed results, it comes at a cost of having a bigger memory footprint and the need for larger memory sizes. Hence, the model is reproduced and transformed to fixed point number representation in MATLAB using pre-trained weights, inputs, and biases. All network parameters are quantized by reducing their word length. The hardware model is perfected until a sensible loss in classification accuracy is achieved.

3. **Developing a NDP unit**
   Data movements occurring in the network were inspected, demands for

memory reads were probed. Using this information, an NDP unit covering minimal area is to be designed. The NDP unit should also have better throughput and low latency.

4. **Integrating NDP unit with Pulpissimo memories**
   Developed NDP unit is placed closely beside Pulpissimo memories. APB bus is configured to trigger the hardware whenever necessary by the processor. In addition, Hardware Abstract Layer (HAL) is created to execute it from the C code efficiently.
5. **Comparing NDP Unit with CNN execution on Pulpissimo**
   NDP unit integrated with Pulpissimo is compared with CNN executed on bare Pulpissimo. The comparison involves power, area, and computation speed.

1.3 Thesis Outline

Chapter 1: Introduction. This chapter presents the challenges this thesis tackles and the purpose of using near data processing for ML applications.

Chapter 2: Theoretical Background. Includes all technical information and concepts related to the thesis.

Chapter 3: NDP Design and Implementation. Describes architecture of designed NDP unit, and modifications made to interleaved memories of Pulpissimo.

Chapter 4: NDP Unit Interface with Pulpissimo. Explains methodology followed to link NDP unit with Pulpissimo and to make it accessible through C code.

Chapter 5: Results and Analysis. A study on performance, power and area utilization of the system performing CNN inference is conducted with and without the designed NDP.

Chapter 6: Conclusion and Future work. A short description about the work done and the results obtained is stated with future work of this thesis.

Chapter 2
Theoretical Background

This chapter first introduces technical concepts related to Artificial Neural Network, followed by concepts of Convolutional Neural Networks and Pulpissimo. These knowledge are necessary to follow further discussions in the thesis. It will also aid to understand the need for a distinctive approach to tackle neural network complexities.

2.1 Artificial Neural Network

The human brain can analyze, discover patterns and perform logical operations within fractions of time. This inspired Warren McCulloch and Walter Pitts, in 1943. They proposed a mathematical model that can imitate the operation of human brain. The model was coined as Artificial Neural Network (ANN). It is now commonly referred as Neural Network (NN).

Artificial Neurons are primary building blocks of Neural Network, they try to mimic the biological behavior of neurons. The human brain has around 100 billion biological neurons. Similarly, several artificial neurons as in Figure 2.1 get arranged together and work collectively in a Neural Network.

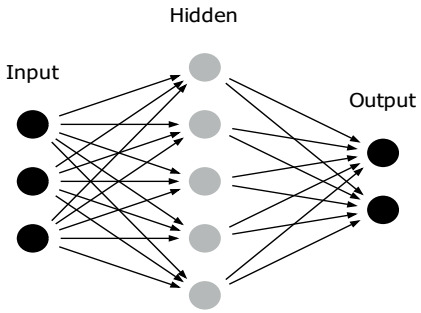


Figure 2.1: An Artificial Neural Network

Artificial Neurons are grouped in form of different layer types namely Input Layer, Hidden Layer, Output Layer. A NN can have many numbers of Hidden Layers. The mathematical formula of an artificial neuron is presented in Equation 2.1, where $\phi(x)$ is the activation function, w are weights, x are input element and b is bias. Each neuron has $k+1$ input signals, one output signal, and can be represented diagrammatically as in Figure 2.2.

$$y = \phi\left(\sum_{n=1}^{n=k} w_n x_n\right) + b_n \tag{2.1}$$

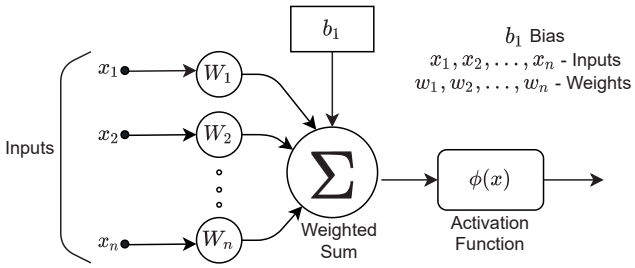


Figure 2.2: Artificial Neuron

Weights and Biases are learnable parameters of Neural Network. The value of weight describes how much impact input will have on the output. The bias value helps neuron to alter its action and make sure the neuron is active even when all weights of a neuron are zero. The result of weighted sum of an artificial neuron is passed through an activation function. Activation functions help to concise the output to a specific range. Apart from this, the activation function provides non-linearity to help neural networks work with a higher degree of polynomials.

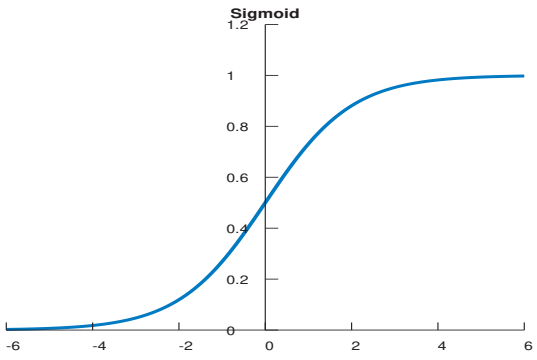


Figure 2.3: Sigmoid Function

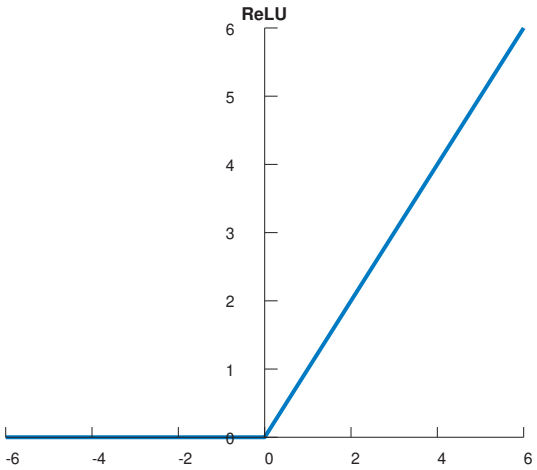


Figure 2.4: Rectified Linear Unit (ReLU)

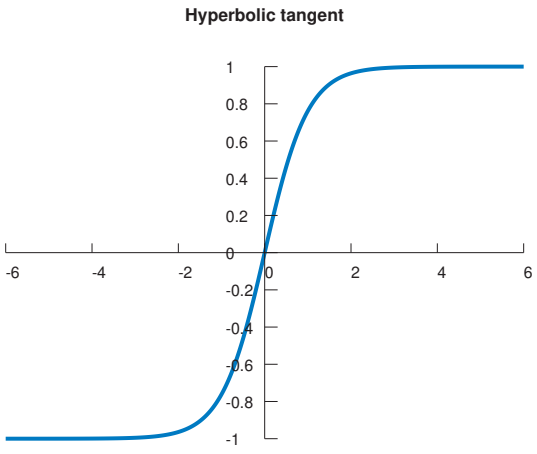


Figure 2.5: Hyperbolic Tangent (tanh)

The type of activation function to be used varies based on application and complexity of neural network. Some of widely used activation functions are presented in Figures 2.3, 2.4, 2.5.

2.2 Types of Artificial NN

The idea of NN sparked curiosity among widespread of researchers over the years and had resulted in more powerful and better algorithms for mimicking the brain. Among different neural networks, the following types are regularly employed.

1. FeedForward Neural Networks (FNN)
   FNN as in Figure 2.6 is simplest, and was earliest developed NN. It comprises of one Input and one Output Layer and with none or many Hidden Layers. Each neuron of a layer is connected to every neurons in next layer. The information moves from input to output through Hidden Layers in a forwarding direction, no results of artificial neurons are fed back into the network. Probabilistic Neural Network, Time Delay Neural Network, Auto encoder, or Convolutional Neural Networks are few examples of FNN.

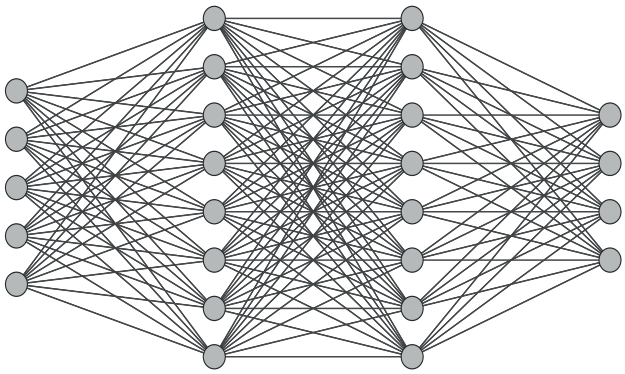


Figure 2.6: A FeedForward Network

2. Recurrent Neural Networks (RNN)
   RNN as in Figure 2.7 has data travelling in circles from one layer to another so that the state of model is influenced by previously predicted results. They have finite internal memory to store previous outputs for processing diverse lengths of input sequence as a consequence they are stateful neural networks. These networks are widely used in Natural Language Processing (NLP) for speech recognition and language modeling.

2.3 NN Development Cycle

Just like the human brain, NN strives to learn from its input data. But it takes many iterations for the model to understand patterns of given input. During the

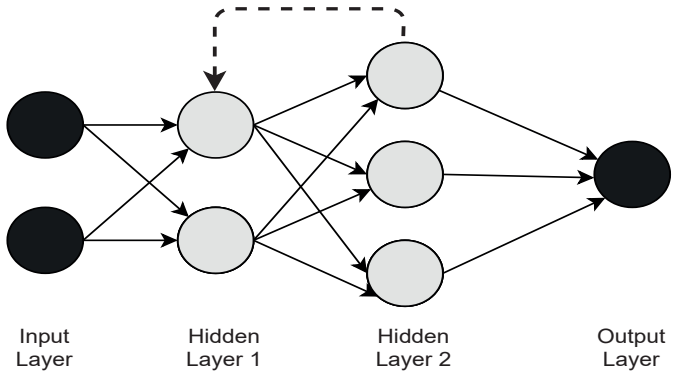


Figure 2.7: A Recurrent Neural Network

development process, three sorts of data set are required. They are training, validation and test set data. 60% of total dataset is allocated for training set. It is used by the network to learn during Training Phase. The remaining 40% of data is split equally for validation and test set. To validate the performance of network after each iteration, validation dataset is used and network accuracy is analyzed. The fine-tuning of learnable parameters ends once network provides a good classification or prediction accuracy with validation dataset. The test dataset is used to evaluate the efficiency of network.

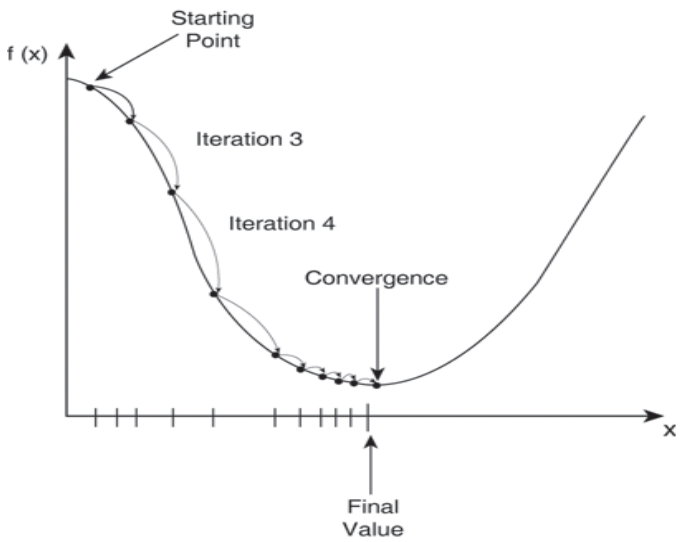


Figure 2.8: Cost Function value after each iteration [12]

The NN development can be subdivided into two stages. They are Training Phase and Inference Phase. During each iteration, parameters of NN are fine-tuned till it reaches an optimal minimum. This stage of preparation is termed as Training

Phase. Cost Functions and Optimizers help to calibrate weights and bias values in this phase.

Cost Function is a measure to know the quality of neural network model. It provides insight on how well the model is delivering and ways the model can be improved. It is simply a measure of difference or distance between predicted value and actual value. The value of cost function should reduce in subsequent iteration during training phase to denote that the model is learning to the given inputs. This is done with the help of optimizers.

Optimizers should reach bare minimum to give smallest value of cost function error. An illustration of cost function value at each iteration is shown in Figure 2.8.

Once Training Phase is successful, Neural Network model would have obtained ability to classify or predict any of input data. During Inference Phase, the trained model is put under test for input it was trained.

2.4 Convolutional Neural Networks

CNN is a Feed Forward Network built up using the following layers:

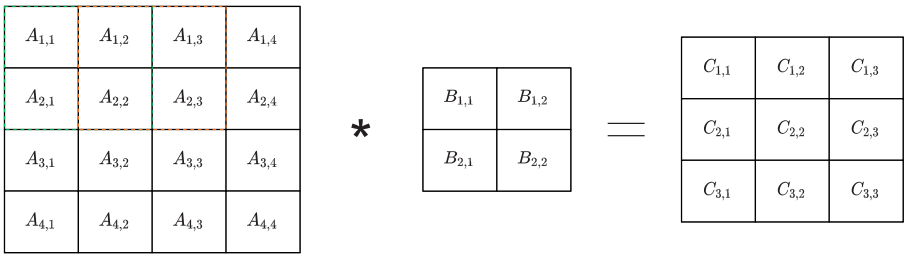
- Convolutional Layer
- Maxpooling Layer
- Dense Layer

2.4.1 Convolutional Layer

Convolutional layer performs mathematical operation convolution. The input data is convolved by using small matrices called kernels. Kernels are used to extract specific features of image. The features can be any relevant data to CNN like edges, shape, textures. Unlike image processing algorithms, values of kernel are fine-tuned during training phase of CNN. The working of this layer is present as an image in Figure 2.9 and mathematical formula that denotes convolution is present in Equation 2.2.

$$C = \sum_{i=1}^{q} \left[\sum_{j=1}^{q} A_{ij} B_{ij} \right] \tag{2.2}$$

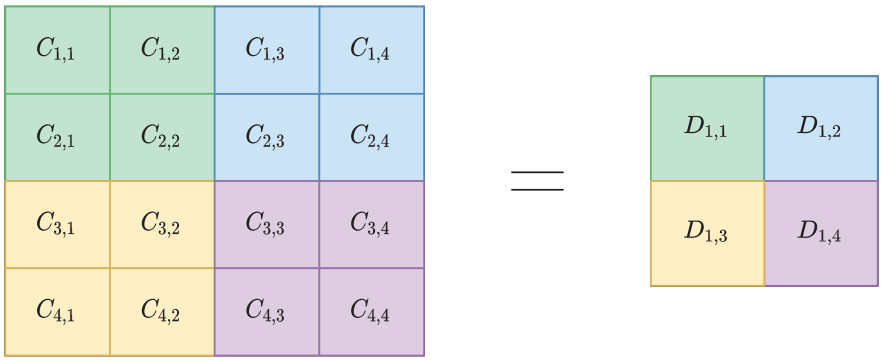
where: A_{ij} = The value of the pixel corresponding to image index i,j
B_{ij} = coefficient of convolutional kernel at index i,j
C = convoluted result
q = width of kernel



$$C_{1,1} = A_{1,1} * B_{1,1} + A_{1,2} * B_{1,2} + A_{2,1} * B_{2,1} + A_{2,2} * B_{2,2}$$
$$C_{1,2} = A_{1,2} * B_{1,1} + A_{1,3} * B_{1,2} + A_{2,2} * B_{2,1} + A_{2,3} * B_{2,2}$$
$$\cdot$$

Figure 2.9: 2D Convolution

2.4.2 Maxpooling



$$D_{1,1} = max(C_{1,1}, C_{1,2}, C_{2,1}, C_{2,2})$$
$$D_{1,2} = max(C_{1,3}, C_{1,4}, C_{2,3}, C_{2,4})$$

Figure 2.10: 2 X 2 Maxpooling with stride 2

Maxpooling [11] is usually placed after convolution layer. This is done to scale down spatial size of feature map. Down sampling helps in reducing dimensionality thereby lessening computation needs in forthcoming layers. It further helps in de-noising as weak noisy values are eliminated.

In Maxpooling, a window of specific size is moved over feature map and its maximum value in that window is picked. This process helps in choosing strongest features from the result of convolution. Maxpooling concept can be represented as Figure 2.10 where $C_{i,j}$ is the feature map and $D_{i,j}$ are Maxpool result.

2.4.3 Flattening

Flattening is a method of converting multidimensional result into a single column vector as in Figure 2.11. This process of transformation makes maxpooling result compatible for a dense layer.

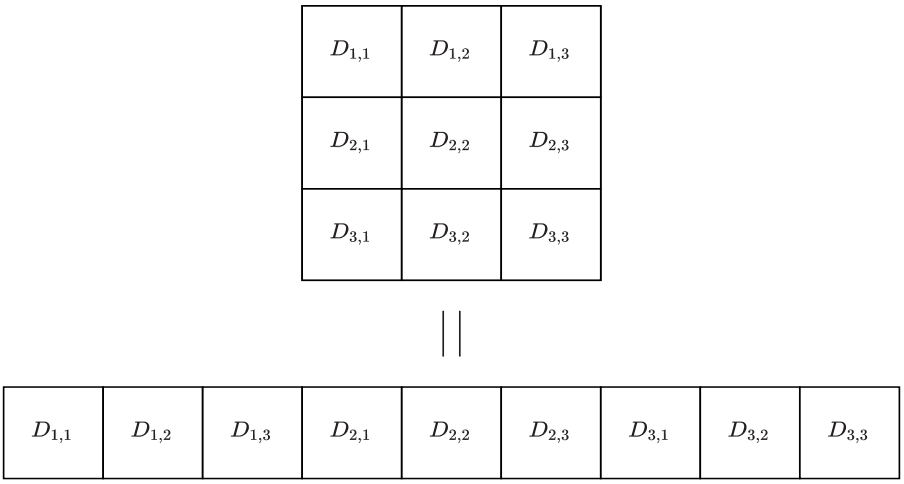


Figure 2.11: Flattening of Maxpool Results

2.4.4 Dense Layer

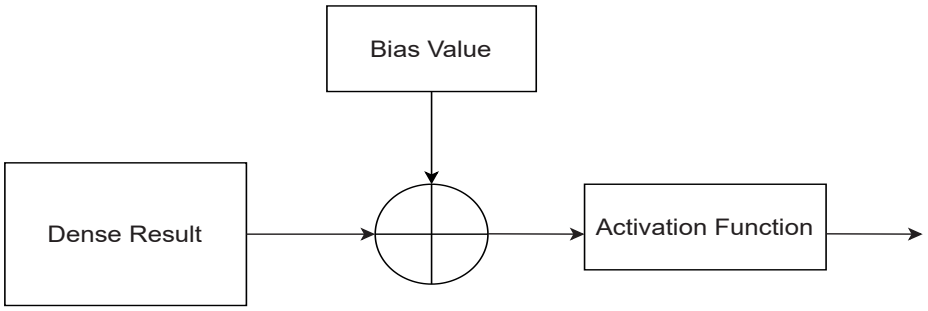


Figure 2.12: Dense Layer Data Flow

One or more layers of artificial neurons are connected to make CNN learn high-level features. The weights of this layer are matrix multiplied with flattened data that is sent as input as in Figure 2.13. Matrix multiplied result is then added with bias and result is passed through an activation function as in Figure 2.12 . Normally, the final layer's activation function is softmax. It helps in determining probability of the class of image the input belongs to.

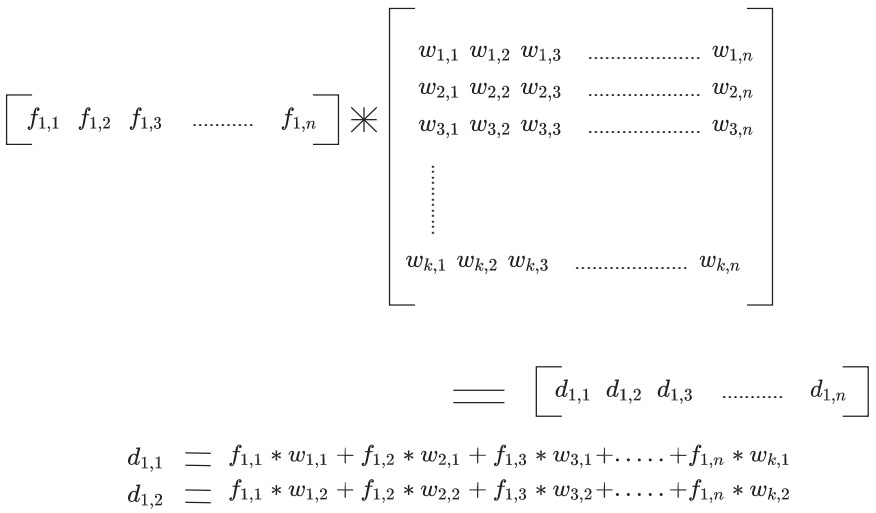


Figure 2.13: Fully Connect Layer Multiplication

2.5 Pulpissimo

The microcontroller Pulpissimo is an open-source edge processing device that aims to bring high computation capacity on-chip. It has an 32-bit single-core architecture that can do parallel processing. This thesis uses Pulpissimo configured with RI5CY core. RI5CY is a 4 stage pipelined 32-bit RISC-V processor.

Pulpissimo contains an inbuilt Hardware Processing Engine (HWPE). HWPE can be used to perform convolution and matrix multiplications efficiently. They extensively use multi-bank memory called Interleaved Banks to speed up the computation by leveraging bandwidth. However, HWPE communicates with memories through Logarithmic Interconnects, and as a result it can be stalled if the bus is busy in satisfying data requests of either core or μDMA.

The overall architecture of Pulpissimo can be found in Figure 2.14.

Variety of peripherals is supported by Pulpissimo. It has a Camera Interface, one SPI, I2S, two I2C and one JTAG debug unit. The load to handle multiple peripherals by the core is avoided by using μDMA. The μDMA works autonomously by responding to requests and requirements of peripheral devices attached to microcontroller. This helps the core work on other tasks parallelly and handle events produced only by μDMA.

The Joint Test Action Group(JTAG) plays a massive role in dumping data and instructions into private banks. It also helps in configuring boot address and other registers in peripherals that are required to operate Pulpissimo.

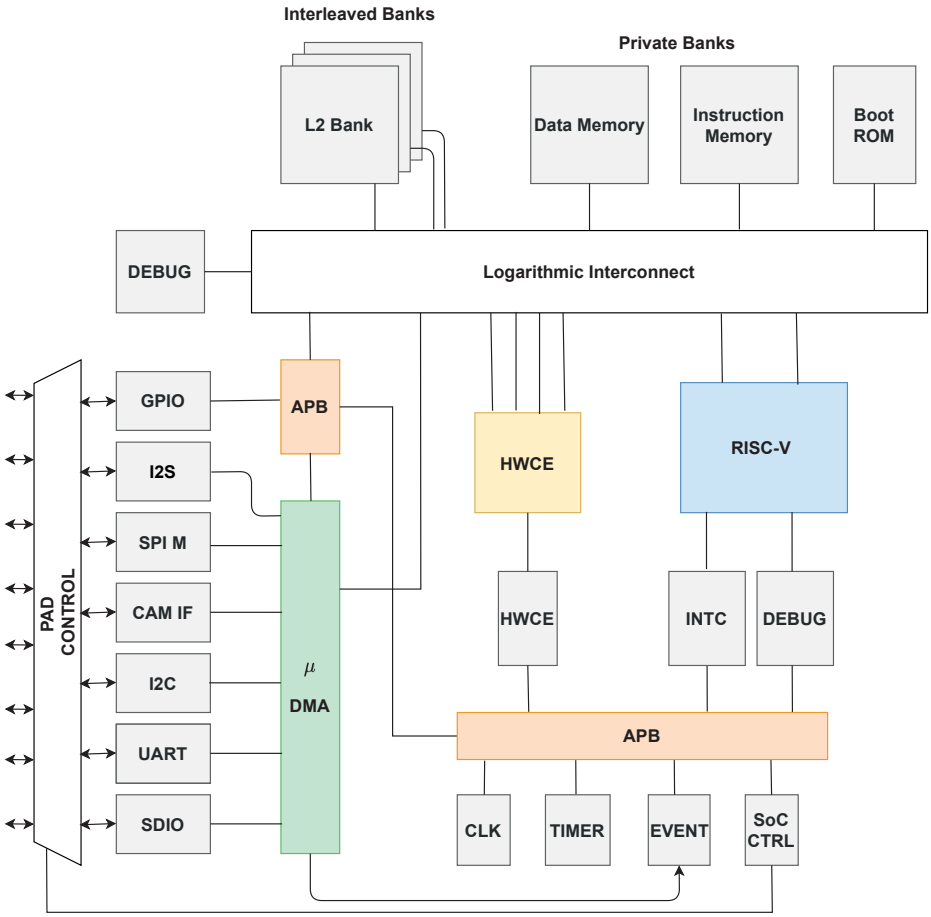


Figure 2.14: Pulpissimo Architecture

Pulpissimo has two Frequency Lock Loops (FLL). The core and memory uses one of the FLL. The peripherals uses the other FLL. The subsystems running at different clock frequencies are synchronized using clock domain crossing.

2.5.1 Soc Memory Map

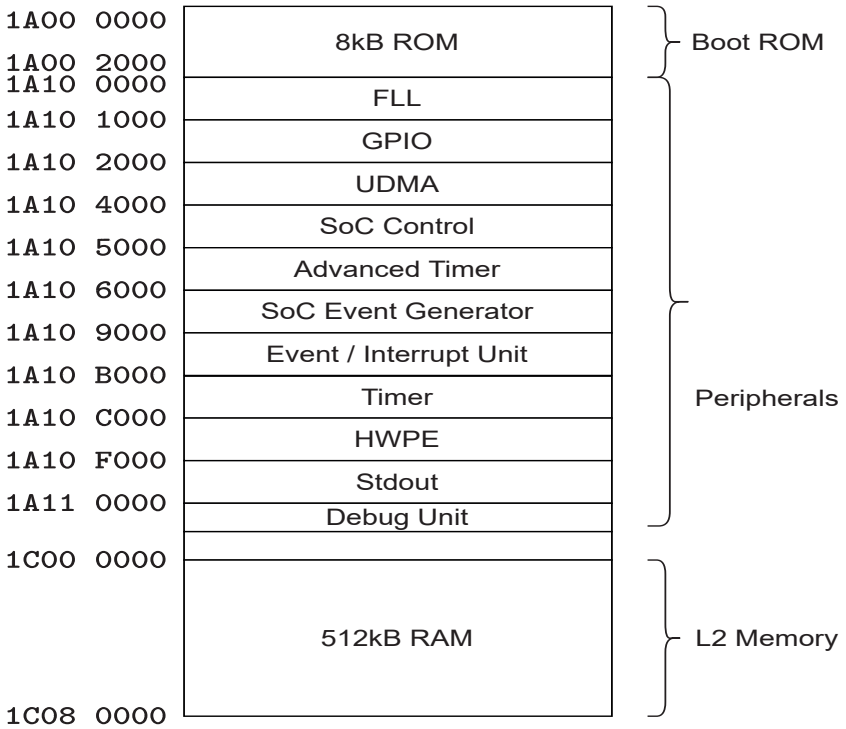


Figure 2.15: Pulpissimo Memory Map

Pulpissimo's memory map as in Figure 2.15 ranges from address space `0x1A00 0000` to `0x1C0 80000`. The range of address from `0x1A10 0000` to `0x1A11 0000` is allocated for peripherals, and the configuration of HWPE falls inside this reservation. The read and write to interleaved memories can be done using the address series from `0x1C00 0000` to `0x1C08 0000`.

Chapter 3

NDP Design and Implementation

The procedure followed to build a CNN model in software and design of NDP unit to perform CNN inference is explained in this chapter. Additionally, this chapter covers details of quantization undertaken to reduce 32-bit floating point CNN model to 8-bit fixed point model.

3.1 CNN Software Model

The software model of CNN is first designed with python using machine learning framework Keras. Keras offers ability to train CNN model by applying backpropagation using cost function and optimizers.

The software model designed to perform classification of numeric digits can have many hidden layers, including two or more convolutional layers. Transforming the same model into hardware as an NDP unit is not viable. This is primarily because going for multiple convolutional layers increases data movements as data access pattern in memory for convolution is not streamlined, and data read is not appropriately utilized. Furthermore, large dimensions of fully connected layers raises the number of network parameters. This results in more computation and require bigger memories.

To find optimal model suitable for NDP. The software CNN model was iteratively designed and analyzed by tweaking following parameters.

- Number of convolutional layers
- Dimension of convolutional kernels
- Number of convolutional kernels
- Length of the stride in convolution
- Number of maxpooling layers
- Size of the stride in maxpooling
- Number of fully connected layers

- Size of fully connected layers
- Type of optimizer and cost function
- Number of iterations
- Type of activation function

The final model that gave a good classification accuracy of 98.06% offering good workability to implement it as NDP unit is shown in figure 3.1. This model contains one convolution with three kernels and a maxpooling layer with two fully connected layers. The final result of fully connected layers is carried through softmax activation function to determined the classified digit.

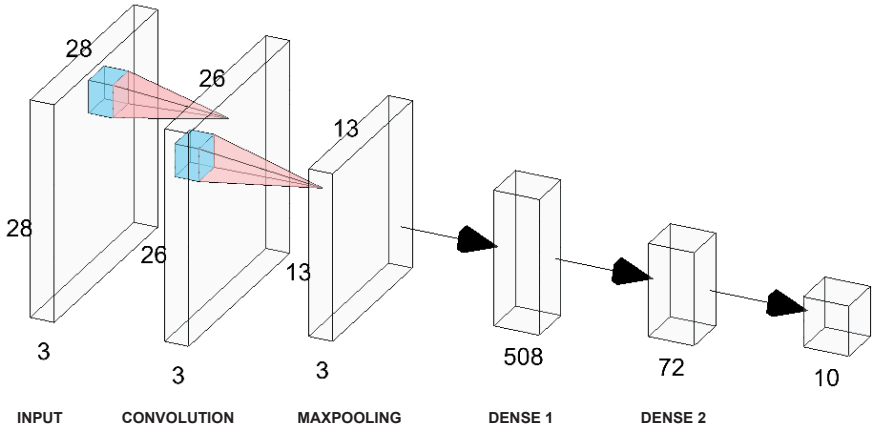


Figure 3.1: Model Convolutional Neural Network implemented in NDP

3.2 Fixed Point Implementation

The primary aim of an NDP unit is to give better power and area efficiency.Therefore, it becomes necessary to implement strategies further to uphold these purposes. One such approach is fixed-point implementation.

The developed CNN model in software performs inference in 32-bit floating-point integer format. The hardware required to do floating-point number computation and their power utilization is very high.The number of loads and stores also increases if 32-bit word length is maintained.

To tackle overload of floating-point integer format. The CNN model inference was reconstructed in MATLAB by using weights and biases of software CNN model.

The reconstruction is necessary to transform network parameters to a fixed point format.

Fixed point format can help reduce area, data reads and power as fractional part of integer is fixed. The word length can also be pinched with a negligible loss in network accuracy. Reducing word length can significantly help decrease memory space as more integers can be narrowed inside a single address space of memory.

The Model regenerated in MATLAB was tweaked and classification accuracy was examined to pick right fixed point CNN model. It was noted that higher the number of bits in an integer, better was the model accuracy. But considering memory space used in mind, 8-bit fixed point was preferred as it provided only 0.6% loss in accuracy.

3.3 Interleaved Memories

Each interleaved memories in Pulpissimo are 128KB in size. It is possible to have one single large 128KB memory for each of them. However, to get a larger bandwidth for NDP unit, each 128KB of memory is substituted with multiple smaller memories using a memory wrapper. The wrapper helps to imitate the functionality of larger memory without any increase in read or write latency.

3.3.1 Memory Wrapper

In this thesis, each 128KB interleaved memory is built using four sub banks of 32KB memories as in Figure 3.2. The wrapper consists of a combinational logic for Chip Enable (CEN) pin. This logic helps to enable one of the sub banks at a time. A multiplexer is applied to choose read data port of enabled memory bank. The hardware design of wrapper can be seen in Figure 3.3.

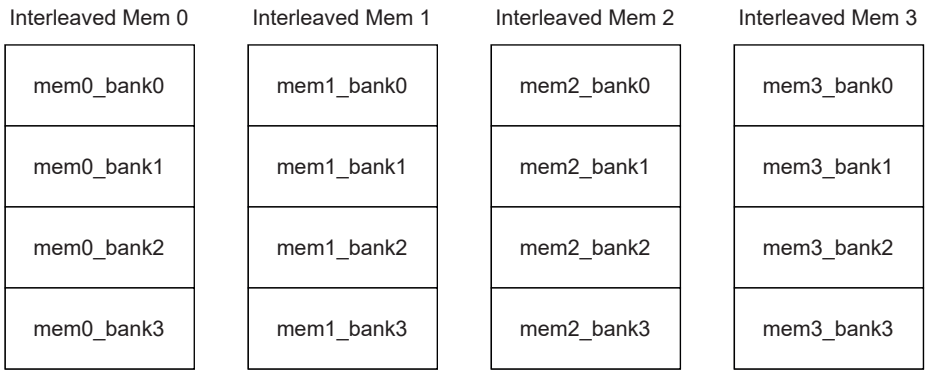


Figure 3.2: Interleaved Memory's Sub Banks

3.3.2 Data Allocation

First layer of CNN that performs convolution requires an input image. The input image with 8-bit of word length and in dimension of 1x784 is stored in first two interleaved memories. Each row of input image is stored alternatively in these two memories, as in Figure 3.4. This arrangement increases bandwidth when performing convolution as two rows of input image data can be accessed in one clock cycle.

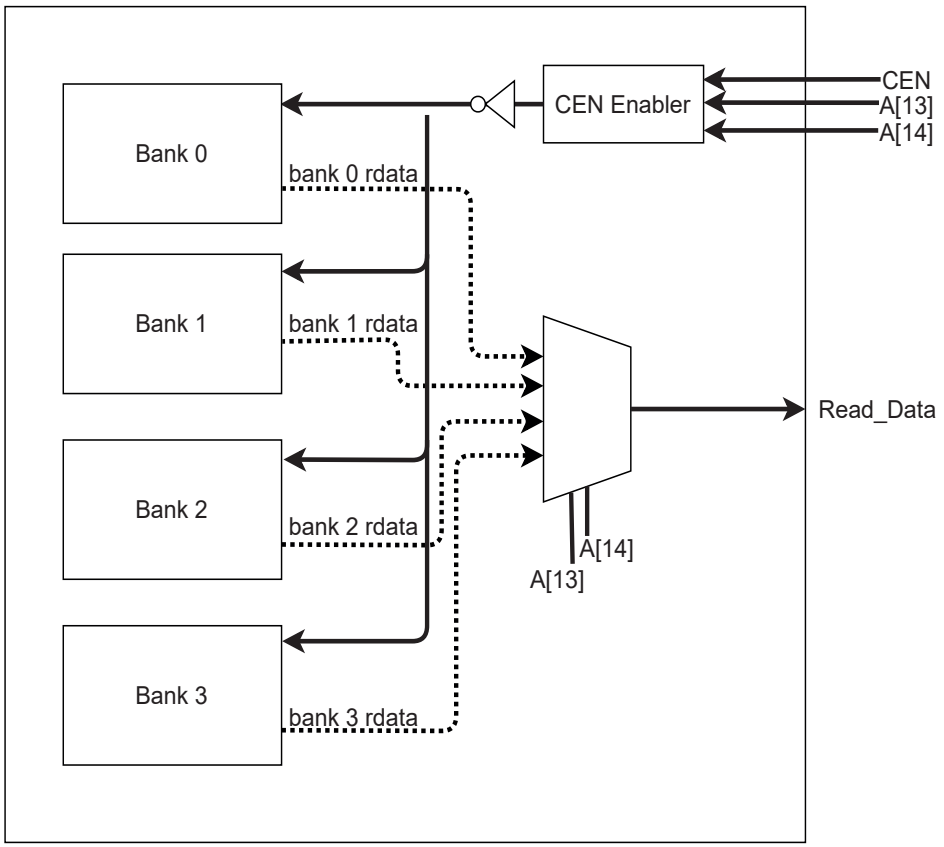


Figure 3.3: Interleaved Memory Wrapper

Parameters required in fully connected layers are stored in each of interleaved memory sub banks as in Figure 3.5. The rectangular boxes inside interleaved memories are sub banks and dotted rectangular boxes are spaces the neural network parameters occupy.

Dense layer parameters placed in banks 1,2,3 of each interleaved memory have a specific format as in Figure 3.6. This format helps to reduce the logic overhead required for bias addition. Bias values get stored in accumulator initially, and then the multiplier values are added with accumulated result.

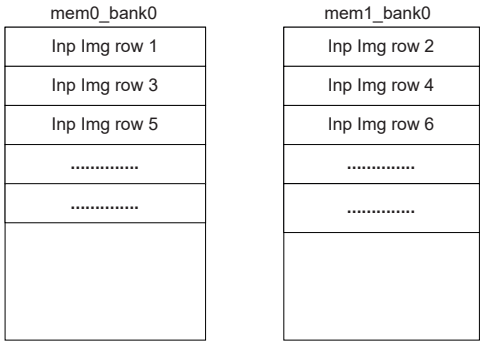


Figure 3.4: Input Image Arrangement

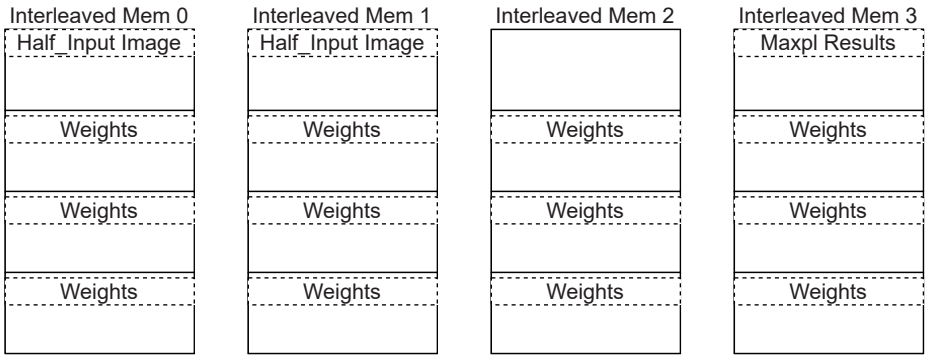


Figure 3.5: Weights and Biases Arrangement

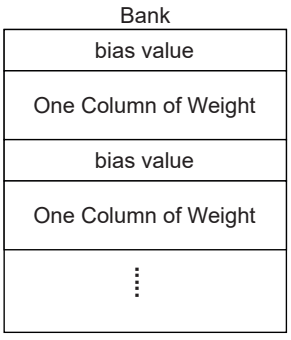


Figure 3.6: Bias and Weights Ordering

3.4 NDP Unit Block Diagram

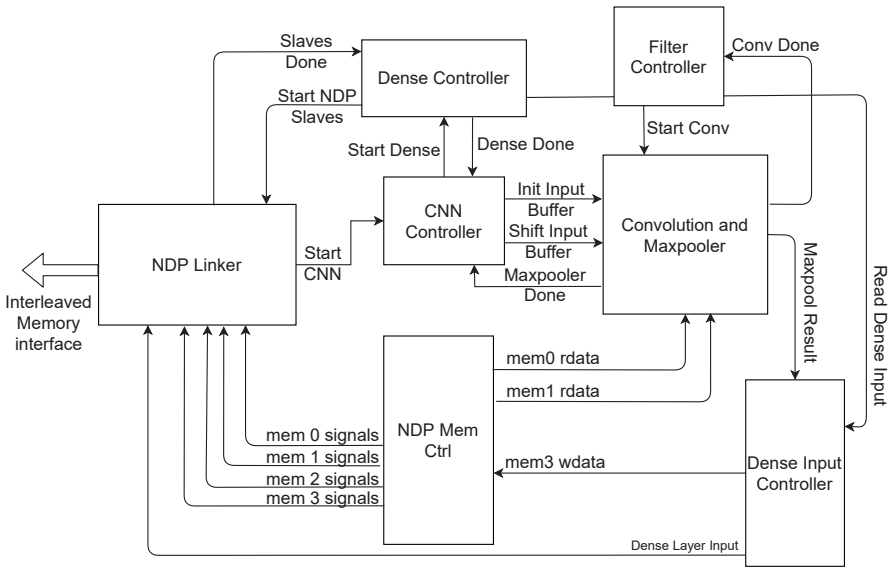


Figure 3.7: NDP Unit Block Diagram

NDP block diagram shown in Figure 3.7 contains `NDP Linker` as front facing module. The linker covers complications involved in aligning NDP unit with memories.

Main controlling module, called `CNN Controller`, is responsible for handling signals required to start convolution with maxpooling and dense top module. The controller focuses more on input buffer present inside convolution with maxpooling module. The input buffer should be shifted, or new input data must be inserted for every four consecutive shifts. Shifting helps convolution to happen in form of a sliding window technique. Adding new data or shifting the buffer is influenced by `Filter Controller` module.

The `CNN Controller` triggers `Filter Controller` once input buffer is set for convolution. The `Filter Controller` begins convolution with each of three kernels it has. This approach assists in data reusability and positively affects power and performance.

Every time a maxpooling result is produced, `Dense Input Controller` module is activated. This module stores the 8-bit result in a 32-bit shift register. Once four shifts are completed, the data is written into fourth interleaved memory. This module also plays a significant role in reading, writing max pooling data and first dense layer outputs.

During dense layer execution, `CNN Controller` triggers `Dense Controller` module and waits until the dense layer is performed. The `Dense Controller` starts

`Dense Input Controller` frequently to read dense input and then instructs NDP slave modules to begin computation. The design and placement of NDP slave modules are explained in section 3.4.2.

3.4.1 Convolution with Maxpooling Module

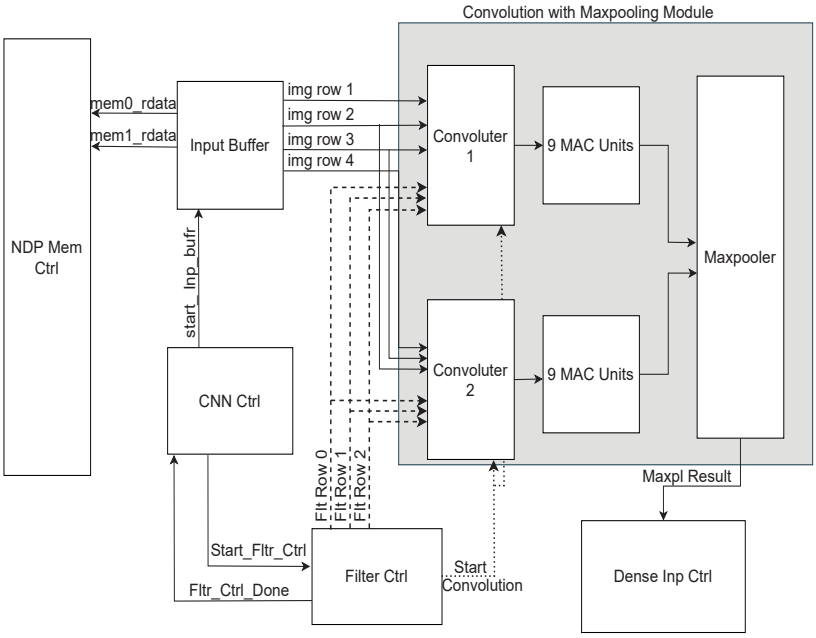


Figure 3.8: Convolution with Maxpooling Module Block Diagram

This module as in Figure 3.8 performs convolution along with maxpooling operations with a total of 18 MAC units. Each MAC unit has a 8-bit multiplier and a 18-bit adder. There is an `Input Buffer` module with four buffers, each of size 64 bits. During start of convolution, four memory reads are performed to fill the four buffers. After this, `Filter Controller` is initiated to conduct convolution with kernels. The NDP unit has two convoluters that perform convolution in parallel using their 9 MAC units. This increases throughput and makes `Maxpooler` module easier to pick maximum result. Since `Maxpooler` module needs four convolution results to pick maximum value, each `Convoluter` module performs convolution twice with different inputs.

Diagrammatic representation of convolution happening parallelly inside NDP unit's convoluters is explained in Figure 3.9.

3.4.2 Dense Layer Module

The designed neural network model consists of two layers of dense with sizes of 508x72 and 72x10. Since dense layer calculation is computation-intensive, the

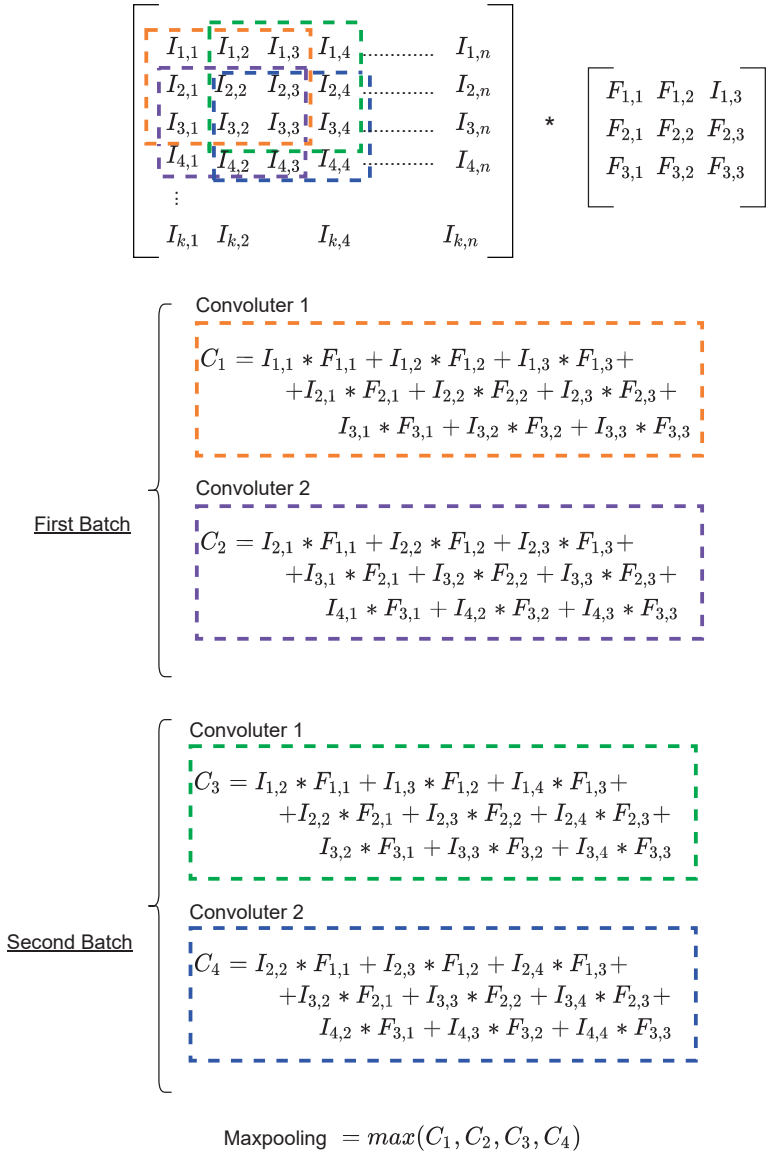


Figure 3.9: Convolution with Maxpooling Module Block Diagram

NDP Unit has its slaves. The slaves are attached to all three banks present in an interleaved memory as in Figure 3.10. In total, 12 MAC units are used for dense layer as three slaves are present in each of four interleaved memory.

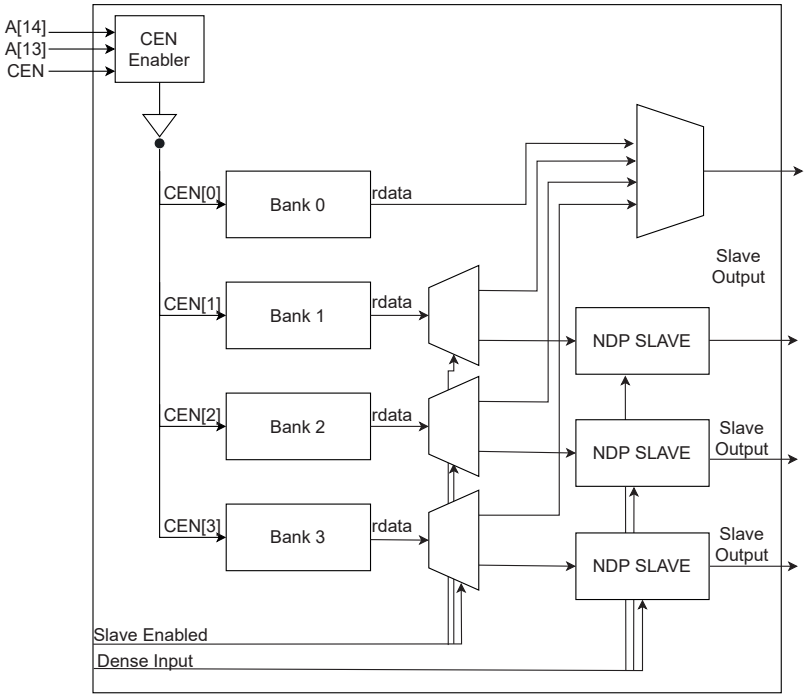


Figure 3.10: NDP slaves placement inside interleaved memory

The use of NDP slaves helps to exploit memory bandwidth as each of them have dedicated memory that can be read. Each slave unit as in Figure 3.11 has its controller. The controller is responsible for configuring default value to adder, counting computation, triggering `MAC Controller` and `Activation Function`. The `MAC Controller` feeds mac unit with 8-bit of value in each cycle. The mac unit consists of one 8-bit multiplier and an 18-bit adder. The adder accumulates multiplier output until the end of computation, and its result is used by `Activation Function` when slave controller initiates it.

Since first value read from memory is a bias value. It is sent directly to adder and is configured as default value. After this, contents read from memory are values of weights that are multiplied with input given by NDP master. The NDP master gives input based on dense layer. For first dense layer, maxpooling results are provided. Second dense layer gets first dense layer output as input.

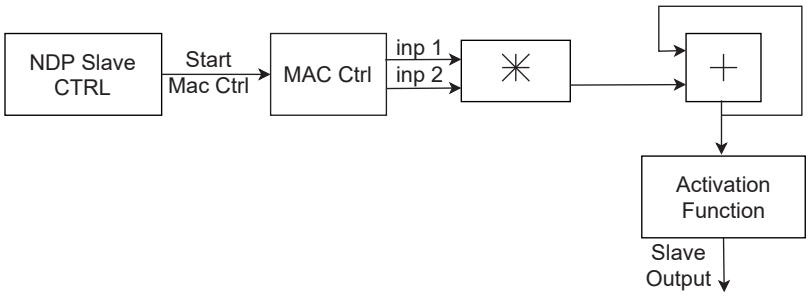


Figure 3.11: NDP Slave

Chapter 4

NDP Unit Interface with Pulpissimo

This chapter explains positioning of NDP unit around interleaved memories. The changes made in peripheral bus and extension made in SoC memory map to make NDP interact with processor is also discussed here.

4.1 NDP Unit Placement

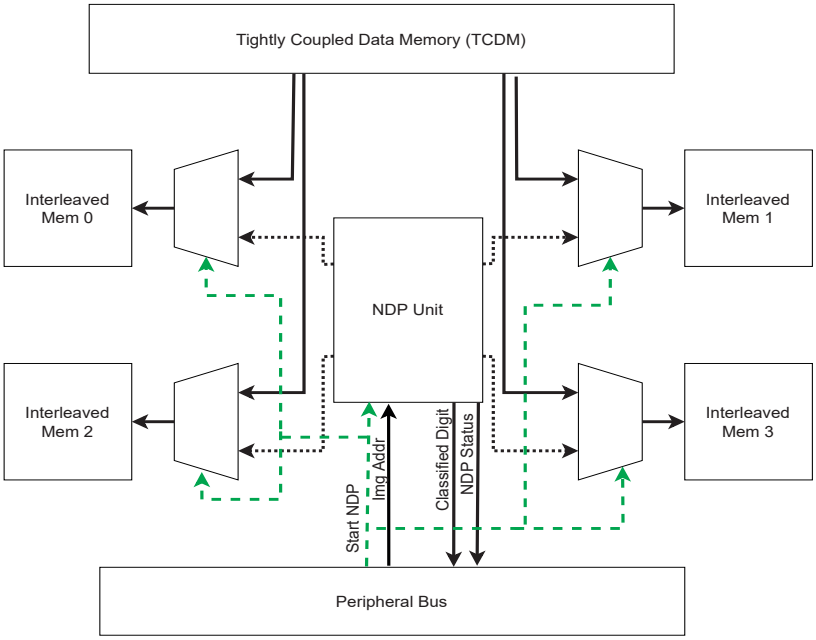


Figure 4.1: Placement of NDP Unit

The concept of NDP is to have computation unit very close to memory. This is realized by placing developed hardware unit nearby interleaved banks of Pulpissimo. Multiplexers are included to access interleaved memories by NDP unit when it is instantiated. This helps to maintain existing functionality of Pulpissimo and for NDP unit to use memories when necessary. The data is written into interleaved

memories by executing a C programming language code in Pulpissimo.

The modified memory subsystem of Pulpissimo is shown in Figure 4.1. The *start_ndp* signal given by peripheral bus switches multiplexers for NDP unit to control interleaved memories.

4.2 Loading Pattern of CNN Parameters

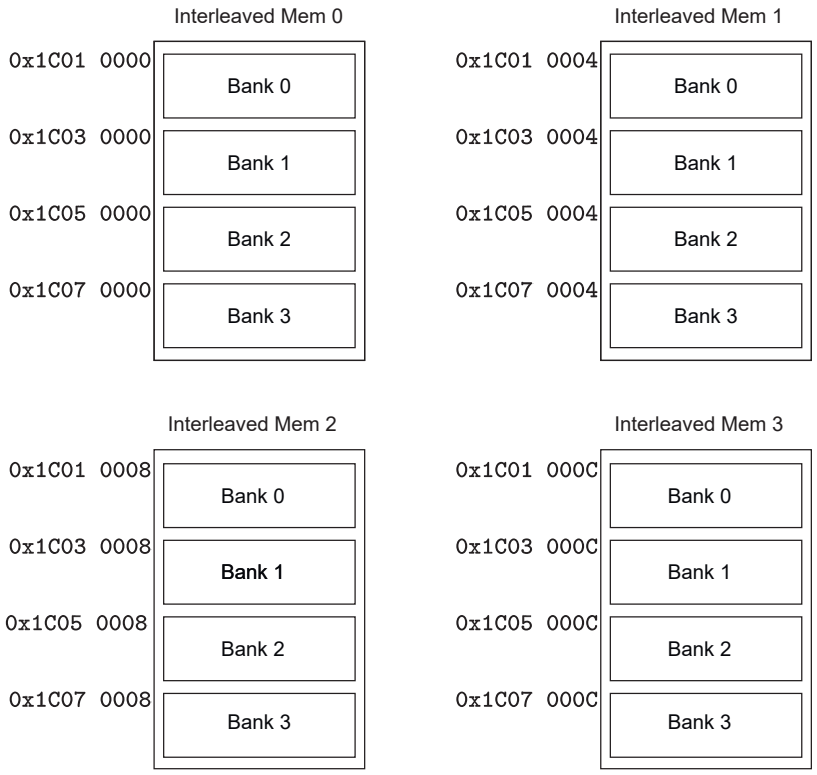


Figure 4.2: Address Distribution Across Interleaved Memories

The data written from C code in form of an array gets distributed across interleaved memories in a round-robin fashion. The first element in array gets written at address `0x1C01 0000`, the second element at `0x1C01 0004`, third and fourth at `0x1C01 0008` , `0x1C01 000C`. The fifth element of array gets written into first interleaved memory at address `0x1C01 0010`. This rule is followed for all elements in array. With the use of a pointer from C Language, data was written to each of banks in interleaved memories.

4.3 SoC Memory Map Extension

Pulpissimo SoC Memory Map defines allocation of address range for subsystems.The SoC control address space in memory map is expanded to accompany NDP associated details as in Figure 4.3. This is done to configure and trigger NDP unit whenever necessary. Moreover, NDP unit can give its status of computation and the digit it has classified by writing back into allocated registers.

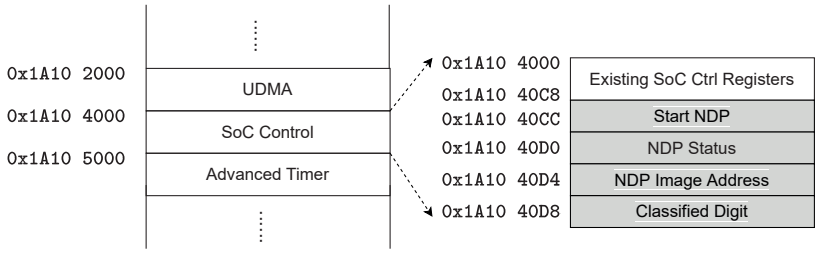


Figure 4.3: Soc Control Extension

Chapter 5
Results and Analysis

A detailed breakdown of analysis conducted on target scenario and base scenario is discussed here. The target scenario is Pulpissimo integrated with NDP unit and base scenario is just the bare Pulpissimo.The study focuses on performance, area, and power consumption. Synthesis was performed with 28nm FD-SOI transistor technology that has its corner as typical with 25-degree temperature and low voltage threshold.

To make a fair judgment, performance analysis was done individually for each scenario. The C programming code that performs CNN was used for base scenario and NDP unit to carry out CNN inference in target scenario. Inbuilt performance counters were incorporated in both scenarios to measure total cycles taken to execute one inference for a given input image.

5.1 Convolution with Maxpooling Layer

5.1.1 Performance and Power

The performance of target scenario computing convolution and maxpooling was matched with base scenario. It was noted that the NDP unit completed its execution 27x times faster by taking only 3816 clock cycles, whereas the base scenario took 105133 clock cycles as in Figure 5.1. This leap in performance is mainly due to data reusability the NDP unit employed to reduce the number of reads. The convolution and maxpooling are conducted on all three kernels for one read of the input image, whereas in base scenario the Pulpissimo reads input image every time for each of the three kernels.

Total power consumed by base scenario to run convolution and maxpooling C code from Figure 5.2 is almost similar to target scenario, as the difference is only 0.8mW. However, it can be noted that the dynamic power consumption of target scenario is higher by 1mW due to the high switching activity that occurs inside the NDP unit.

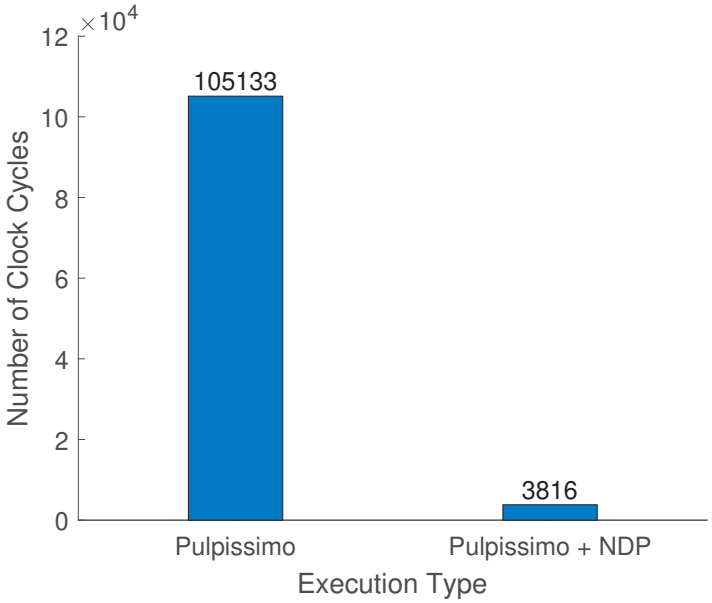


Figure 5.1: Convolution with Maxpooling Performance

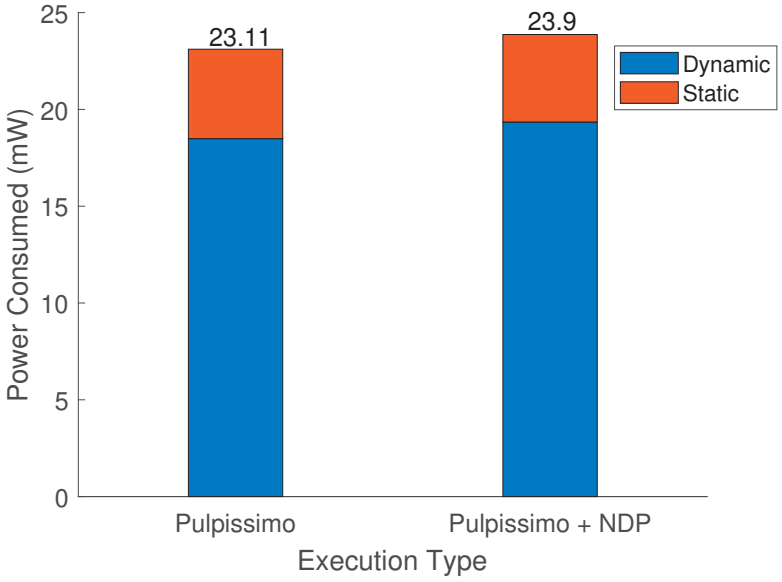


Figure 5.2: Convolution with Maxpooling Power Consumption

5.2 Dense Layer

5.2.1 Performance and Power

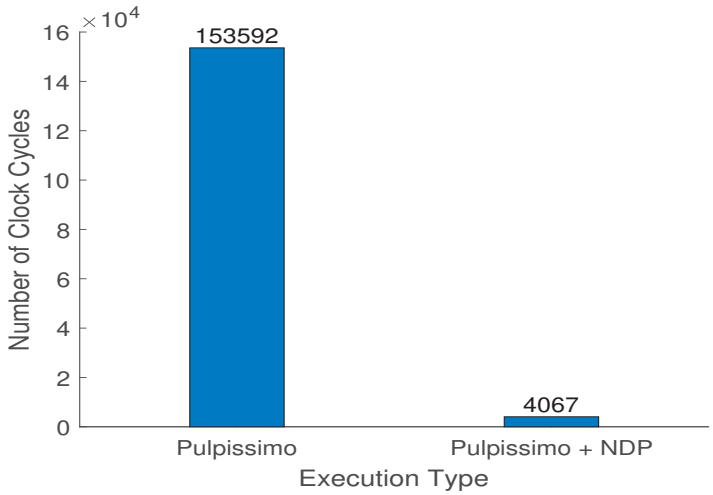


Figure 5.3: Dense Layer Performance

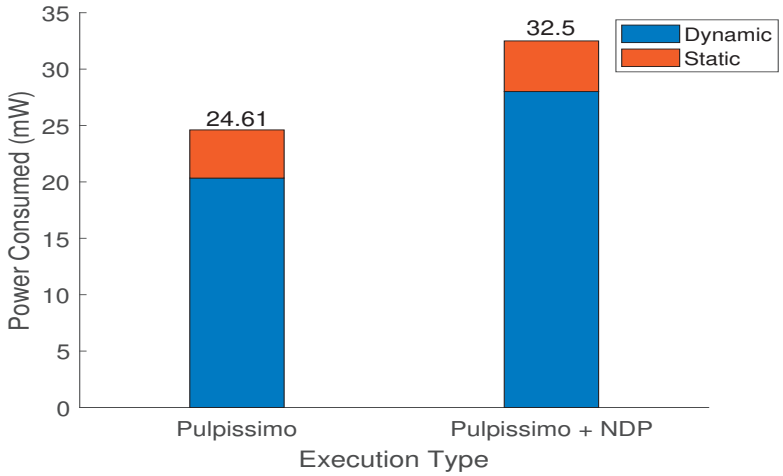


Figure 5.4: Dense Layer Power Consumption

NDP unit in target scenario completed dense layer operation at a speed of 37x compared to base scenario. This increase in performance can be viewed graphically using Figure 5.3. In base scenario, Pulpissimo took many clock cycles due to the memory bandwidth limitation, but NDP unit tackled this issue by utilizing its slave units. The salve units are attached to each of the sub banks in an interleaved

memory. Thus, they can independently read memory and compute for dense layer.

The target scenario takes 32.5mW of power for dense layer computation, 7.9mW more than base scenario. This is primarily due to the high switching activity of NDP unit slaves. The dynamic and static power consumption of both the scenario's can be seen in Figure 5.4.

5.3 Full CNN Inference

5.3.1 Performance and Power

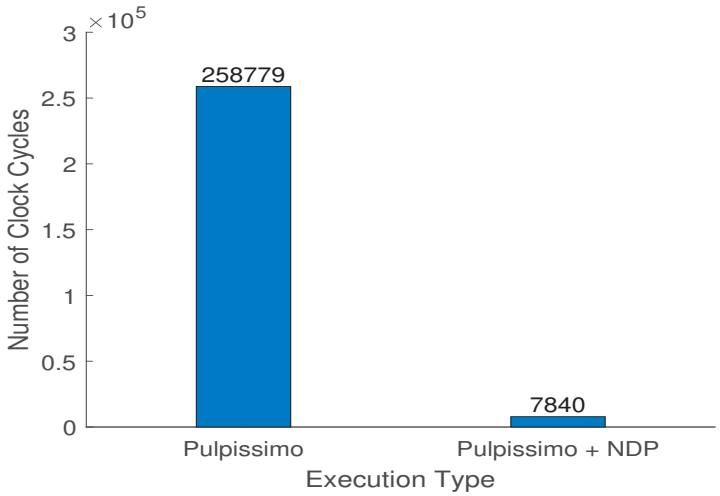


Figure 5.5: CNN Inference Performance

The number of clock cycles taken by NDP unit in target scenario for a full CNN inference is 7840. Whereas, the clock cycles taken by base scenario to do the same task is 258779. Hence a 33x increase in performance is achieved. The reason for such a large improvement is explained in section 5.1.1 and 5.2.1 as a full inference is summation of the convolution with maxpooling and dense layer.

From Figure 5.6, It can be observed that the target scenario takes 4.6mW of additional power to give this high performance.

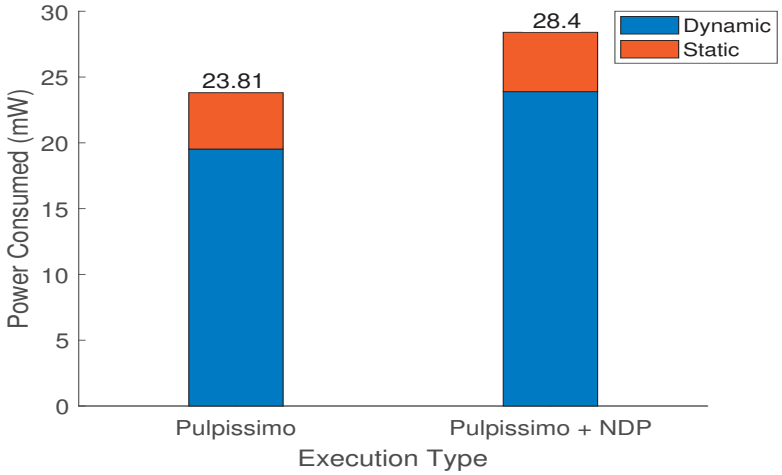


Figure 5.6: CNN Inference Power Consumption

5.4 Pulpissimo and NDP Area

Pulpissimo covers an area of $3.081mm^2$ out of which the NDP unit occupies an area of 0.8%. From Figure 5.7, it can be noted that the *ram* module containing memories of Pulpissimo is the highest area possessing module.

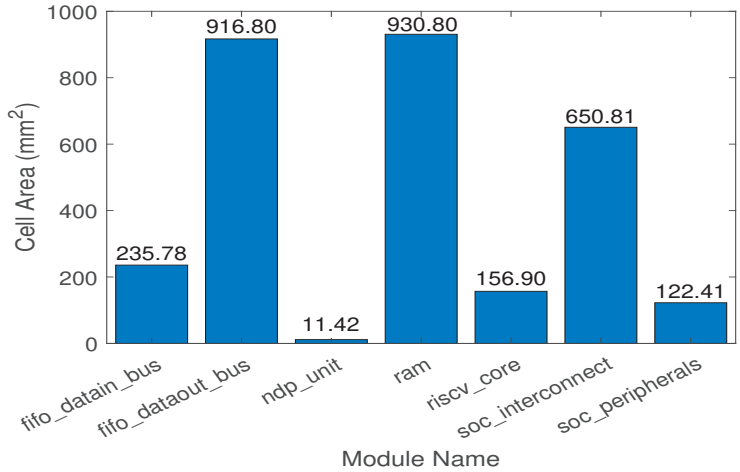


Figure 5.7: Area Occupied by Pulpissimo and NDP Unit

Chapter 6
Conclusion

The concept of Near Data Processing has been investigated and implemented in this thesis. A Convolutional Neural Network model that can classify MNIST images was chosen and realized as an NDP unit. It was merged into multi-bank memories of a microcontroller platform. The multi-bank memories were built using multiple smaller memories to increase memory bandwidth for NDP unit. The APB peripheral bus were extended to make the RISC-V processor interact with NDP unit.

The Pulpissimo covers an area of $3.081mm^2$ out of which the NDP unit occupies only 0.8% of total area. The NDP unit gave 33x more performance when compared with running C code in bare Pulpissimo. In terms of energy, the Pulpissimo with NDP unit achieved 28x increase in efficiency.

The NDP unit is designed to be power efficient besides high performance. Power consumption is decreased by transforming neural network model to a fixed-point number representation format. In addition, reducing word length from 32-bit to 8-bit substantially helped to save memory space by a factor of four. The data movements that occurred in CNN model were investigated to make NDP unit have fewer data transfers. Data reusability was a primary factor for high performance and low power consumption for the NDP unit.

In the future, accuracy of fixed point CNN model should be improved to be greater than 98.5% to further reduce misclassifications. In addition, reducing size of multi-bank memories can help in lowering power consumption. The NDP unit can be designed much more generic to be flexible with number of convolutional and dense layers.

References

[1] G. E. Moore *Progress in digital integrated electronics* in Proc. IEDM Tech. Dig., 1975, pp. 11–13.

[2] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc *Design of Ion-implanted MOSFET's with Very Small Physical Dimensions*,IEEE Journal of Solid-State Circuits, vol. 9, no. 5, pp. 256–268, Oct 1974

[3] W. A. Wulf and S. A. McKee, *Hitting the Memory Wall: Implications of the Obvious* SIGARCH Comput. Archit. News, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[4] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk, Corporaal, Albert-Jan Boonstra, *Near-Memory Computing: Past, Present, and Future*, `https://arxiv.org/abs/1908.02640`

[5] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi *A scalable processing-in-memory accelerator for parallel graph processing*, 2015 ACM/IEEE 42nd Annual International Symposium on, pages 105–117. IEEE, 2015.

[6] Ivan Fernandez, Ricardo Quislant, Christina Giannoula, Mohammed Alser, Juan Gómez Luna, Eladio Gutiérrez, Oscar Plata, Onur Mutlu (2020). *NATSA A Near-Data Processing Accelerator for Time Series Analysis.* `https://arxiv.org/abs/2010.02079`

[7] Eugene Tam, Shenfei Jiang, Paul Duan, Shawn Meng, Yue Pang, Cayden Huang, Yi Han, Jacke Xie, Yuanjun Cui, Jinsong Yu, Minggui Lu. (2020). *Breaking the Memory Wall for AI Chip with a New Dimension*, `https://arxiv.org/abs/2009.13664`

[8] Stefano Corda, Bram Veenboer, Ahsan Javed Awan, Akash Kumar, Roel Jordans Henk Corporaal. (2020). *Near Memory Acceleration on High Resolution Radio Astronomy Imaging.* `https://arxiv.org/abs/2005.04098`

[9] P. Das, S. Lakhotia, P. Shetty and H. K. Kapoor *Towards Near Data Processing of Convolutional Neural Networks*, 2018 31st International Conference on

VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), Pune, 2018, pp. 380-385, doi: 10.1109/VLSID.2018.94.

[10] LeCun Yann, HaffnerPatrick, Bottou Léon, Bengio Yoshua *Object Recognition with Gradient-Based Learning*, `https://doi.org/10.1007/3-540-46805-6_19`

[11] W. Xu, Z. Wang, X. You and C. Zhang,*Efficient fast convolution architectures for convolutional neural network*, 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, 2017, pp. 904-907, doi: 10.1109/ASICON.2017.8252623.

[12] A Concise History of Neural Networks `https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec`

Printed by Tryckeriet i E-huset, Lund 2021

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2021-853
http://www.eit.lth.se