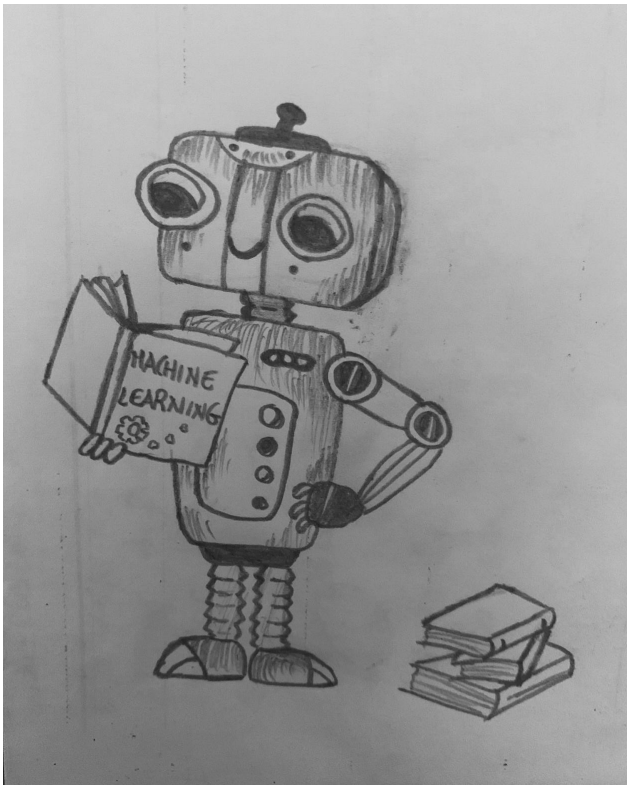


FPGA-based Tensor Accelerator for Machine Learning

Erasmus Master thesis in Computer science and engineering

Francesco Angione

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

FPGA-based Tensor Accelerator for Machine Learning

Francesco Angione

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Supervisor: Pedro Petersen Moura Trancoso, Department of Computer Science and Engineering, Chalmers University of Technology
Home Supervisor: Paolo Bernardi, Department of Control and Computer Engineering, Polytechnic of Turin
Examiner: Per Larsson-Edefors ,Department of Computer Science and Engineering, Chalmers University of Technology

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A robot involved into the self learning process, reading. Thanks to Mariateresa Angione, CopyRight free Image.

Abstract

Part of a Neural Network inference execution mainly consists in multiplications and additions, basic operation of tensor convolutions, and across several execution data, especially weight tensors, are reused. Clearly, those operations are executed on a CPU but, as it is well known, they are independent of each other and therefore they can be executed in parallel by the means of parallel architectures, such as GPU or domain specific hardware platform. In the following pages, the state-of-the-art for accelerating Neural Network inference is explored starting from the newest proposed GPGPU architecture by NVIDIA to the domain specific accelerator from Google, NVIDIA, and Habana.
With the state-of-the-art awareness, a hardware accelerator capable of execution tensor convolution, compute and memory intensive operation of a Neural Network, is designed from scratch. It is also designed for accommodating different data type computation request from Neural Network models, ranging from integer8/16/32/64 to floating-point 32 and brain floating-point 16. Starting from the hardware system development, through the software development of a library capable to use the underlying hardware, it ends with integration into a popular Machine Learning framework, Tensorflow.
The work is carried out on a configurable hardware, FPGA, which allows to explore different design points, in terms of latency and number of processing elements, for different Neural Network models and data type. Moreover, the impact of integrating the accelerator into the Neural Network model is measured and compared with different platforms. Energy consumption is also estimated in the case of deployment on mobile devices.

Keywords: Computer, science, computer science, engineering, hardware, accelerator, machine learning.

Acknowledgements

Here, you can say thank you to your supervisor(s), company advisors and other people that supported you during your project.

Francesco Angione, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

Machine learning is one of the hot technologies today as it is being used to solve complex problems that would otherwise be very hard or costly to solve with traditional methods. Speech and image recognition as well as many other complex decision-making problems such as self-driving vehicles are successfully solved with machine learning and deep-learning. In the last years, the number of published papers regarding Machine Learning have growth exponentially, and the success of machine learning has been driven by the current available hardware which could provide the required demands in terms of storage and compute capacity. But obviously as problems scale so do the demands and thus companies has started to develop, deploy and sell their own hardware platform, such as Tensor Processing Unit [1] from Google, NVDLA[2] from Nvidia and Gaudi [3] and Goya [4], respectively for training and interference, from Habana (acquired by Intel).

The use of commodity hardware is not the most effective and efficient way to execute Machine Learning, so research is looking at flexible hardware solutions [5] [6] that can satisfy the required demands for different Machine-Learning models but at lower cost and energy consumption in order to be deployed also on mobile devices. Moreover, during the inference process, a model does not need high precision computations [7] [8] for achieving high accuracy into its outputs. As it is very well-known, hardware accelerators are capable, if designed correctly, of delivering a lot of improvements in terms of the latency but also in terms of energy efficiency [9]. Thus, in order to obtain the best solution in every metric a hardware-software co-design is needed, requiring to the hardware designer a basic knowledge of machine learning algorithms.

Machine Learning includes two processes, the training and the inference. The training process is done off the field, on powerful machines, exploiting different algorithms for optimizing the models in terms of memory footprint, data type and feedback mechanisms for fine-tuning the weight values. On the other hand, the inference process is the execution of the trained model, applying the inputs and expecting the correct outputs. It is done on the field, for example a mobile device, which is area and energy constrained. The inference process is massive composed of multiplication and addition and on a normal CPU-based system they are executed sequentially, increasing the latency of the model and the energy consumption due to data movement.

Thus, the goal is to develop a hardware accelerator from scratch, which implements a tensor-based convolution. Exploiting a non Von Neumann architecture and data locality and reuse for weights reduces the CPU workload and boost the models performance. The use of different arithmetic data type can drastically reduce the computations without reducing the final accuracy of the Neural Network [7] [10]. From a hardware perspective, the use of different arithmetic precision [11], such as the use of integer operations instead of floating-point operations, can lead to benefits in terms of area, energy consumption and latency.

In order to have the possibility of exploring different solutions, in terms of size and latency, of the accelerator the work is deployed on FPGA and it is integrated into a common Ml-Framework, Tensorflow. Accuracy of operations, reliability, performance and energy efficiency are evaluated and compared to the implementation of the same models executed on a GPU.

2

Background

Can a machine think?
— Alan Turing, Computing Machinery and Intelligence

2.1 Overview

In the past decade many companies have started to advertise the use of AI, even if they are using a subfield of the AI, in their products and software applications. Nevertheless, the recent growth, the AI is not youth.
It takes one of its roots from a theoretical paper of *Alan Turing* published by journal *Mind* in the 1950 [12].

The general definition of Artificial Intelligence (AI): *intelligence demonstrated by machines, any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals* [13].
In general, "artificial intelligence" is used when machines mimics the cognitive functions of the human mind, i.e. learning and problem solving.
According to the definition, AI is too vast to be studied and simulated [13]. Therefore, it has been divided into subfields, characterized by different traits, such as knowledge representation, planning, learning, natural language processing, perception, motion and manipulation, social intelligence and general intelligence.

Artificial Intelligence can be seen as a general purpose technology. It does not exist a general task on which it excels neither how to characterize them.

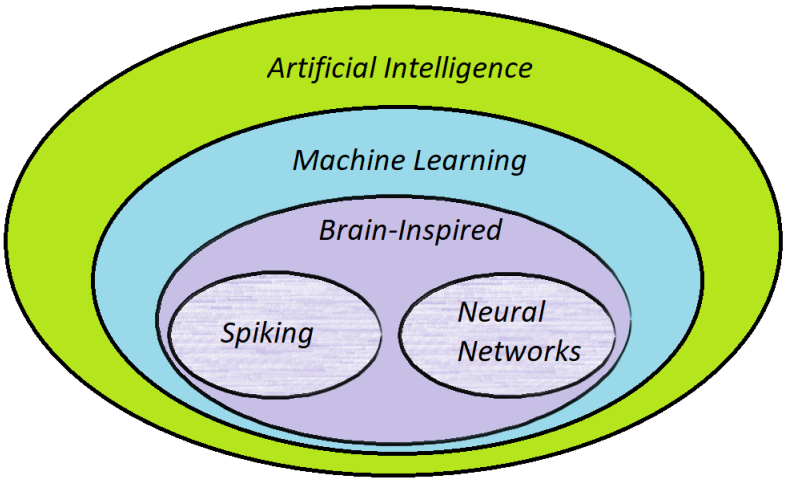


Figure 2.1: Classification of AI with emphasis on Machine Learning and its subclassification

2.2 Machine Learning

A particular interesting subcategory of AI in Computer Science is the machine learning. It is the study of algorithms used to perform a specific task without explicit programming the machine, relying on patterns and inference, in order to make decisions. This approach is used where it is tricky, or unfeasible, to develop a conventional algorithm for solving the task.

A peculiarity of machine learning model is that it is composed of two processes, training and inference.
The inference process is the process in which a conclusion is given at the end of the evaluation process, i.e. the input stimulus are applied to the model and the output is observed.
The training process has to be done before the model is put on the field, before the inference process, otherwise the latter can give wrong results. As the name suggests, in this process the model learns how to behave, adjusting the weight accordingly to the applied inputs and expected outputs. Besides this type of training and according to [13], other exists, characterized by approach, type of data and tasks:

- Supervised Learning, it builds a mathematical model of a set of data that contains both the inputs and the desired outputs.
- Unsupervised Learning, it takes a set of data that contains only inputs and find structure in the data.
- Semi-supervised Learning, it falls between unsupervised learning and supervised learning.
- Reinforcement Learning, it concerns how software agents should take actions in order to maximize some notion.
- Self Learning, It is a learning with no external rewards and no external teacher advices.

- Feature Learning, also called representation learning algorithms, often attempts to transform data and preserve at the same time. It is used as a preprocessing step before any classification or predictions.
- Sparse Dictionary Learning, it is a feature learning method where a training example is represented as a linear combination of basis functions, and is assumed to be a sparse matrix.
- Anomaly Detection, also known as outlier detection, identifies rare items, events or observations which are significantly different from the majority of data.
- Association Rules, it is a rule-based method for discovering relationships between variables in large databases.

Machine learning space is also divided into other type of models such as decision tree, support vector machines, regression analysis, Bayesian networks and genetic algorithms. As it can be seen in Figure 2.1 Brain Inspired machine learning is also divided in subcategories.

2.2.1 Brain Inspired

It is based on algorithms which take its basic functionalities from our understanding of how the brain operates, trying to mimic the functionalities.

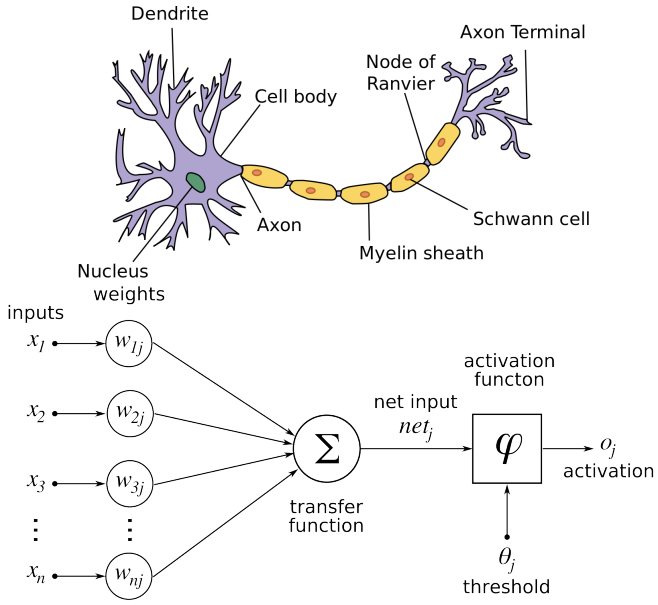


Figure 2.2: A parallelism between a human-brain neuron and a neuron in a Brain Inspired Network[1]

In the human brain, the basic computational unit is the neuron.
Neurons receive input signal from dendrites and produce output signal along the axon which interacts with other neurons via synaptic weights.
The synaptic weights are obtained after a learning process, which can strengthen them or not.

[1]Figures under CC license

2.2.1.1 Neural Networks

Neural Networks (or Artificial Neural Networks) are graphs in which every node is interconnected to others using edges, which have a weight properly tuned during the training process.
As mentioned before, each and every node of the neural networks is called artificial neurons (a loosely model of its biological counterpart) and the connections (synapses in biological brain) can transmit information from a neuron to another. In Figure 2.2 the neurons receive signals, which is processed internally, and then they propagate it to the other connected neurons.
The information exchanged between a neuron and another is a real number, a result of a non-linear function of the sum of all its input.
In the Figure 2.3 an implementation of a Neural Network can be appreciated.

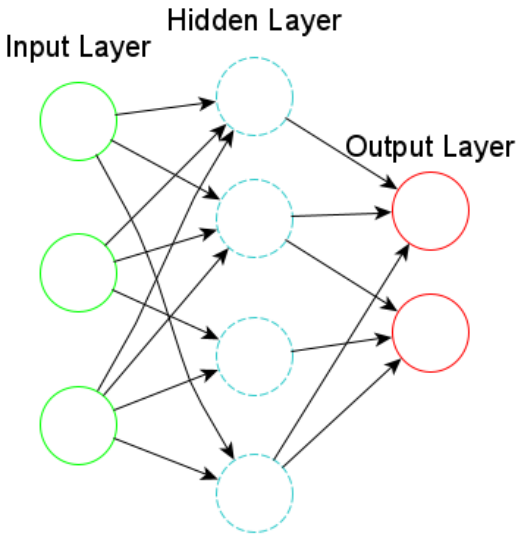


Figure 2.3: Example of a Neural Network

As it can be seen in Figure 2.3, it is always divided in layers in which only the output and input layers are visible from the external world, as consequence the internal layers are called hidden layers. When an input vector is applied, it will propagate from the left side of the network to its right side through the layers and the neurons which compose each layer. It is worth to mention that layers may perform different kind of computation on their inputs. Moreover, the deep neural networks are named after the huge amount of hidden layers.

In the early stages of ANNs the goal was to solve problems as the human brain would do. However, over time, the aim moved to perform specific tasks, leading to a different architecture of the biological brain and brain-inspired networks (Spiking Neural Networks).

Depending on how the edges are connected and the topology, a Neural Network can be classified in several sub-types:

- Feed forward, the data move only from input layer to output layer without cycles in the graph.
- Regulatory feedback, it provides feedback connections back to the same inputs that active them, reducing requirements during learning. It also allows learning and updating much easier.
- Recurrent neural network, it propagates data backward and forward, from later processing stages to earlier stages.
- Modular, several small networks cooperate or compete to solve problems.
- Physical, it is based on electrically adjustable resistance material to simulate artificial synapses.

2.2.1.2 Spiking Neural Networks

Spiking neural networks (SNNs) are artificial neural networks that more closely mimic natural neural networks [14].
In addition to neuronal and synaptic state, in their operational model, SNNs adds the concept of time. The idea is that neurons in the SNN do not activate at each propagation cycle but rather activate only when specific value is reached.
The current activation level is modeled as a differential equation and it is normally considered as neuron's state.
In principle, SNNs can be applied to the same application of Artificial Neural Networks. Moreover, SNNs can model brain of biological organisms without prior knowledge of the environment. Thus, SNNs have been useful in neuroscience for evaluating the reliability of the hypothesis on biological neural circuits but not in engineering.

SSNs are still lagging ANNs in terms of accuracy, but the gap is decreasing and has vanished on some task[15]. However, computer architectures based on SNN have a huge energy footprint compare to other types of architecture [16].

2.3 Machine Learning Quantization

The reduction of computation demand, the increase of power efficiency and the memory footprint of machine learning algorithms can be achieved through the quantization.

Quantization is basically a set of techniques which convert, and map, input values from a large set to output values in a smaller set.
The idea of Quantization is not recent, it has been introduced since the birth of digital electronics. Imagine taking a picture with the phone's camera, the real world is analog and the camera is capturing the analog world and converting it into a digital format. Nevertheless, the high quality of nowadays pictures, quantization is not lossless.

An trivial quantization example for Neural Network model is given in the below figure, where a set of potentially infinite value(floating-point) are mapped to finite values (integer).

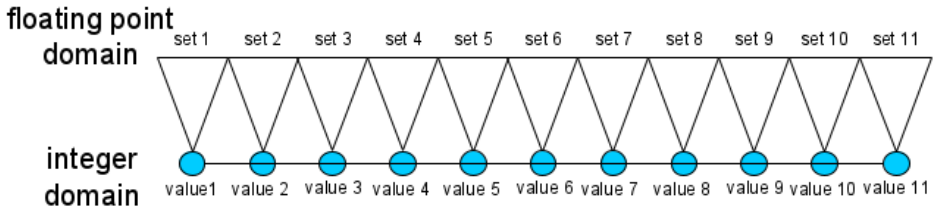


Figure 2.4: Approximation of floating-point values to integer values

It has been proved that even if the model has been quantized, for example from fp32 to integer32, its accuracy is still good and the accuracy drop between the two data representation is negligible [7].
Several quantization techniques can be applied, together or separated, to already trained ML models (post-training quantization):

- Linear quantization: data are directly scaled by taking their maximum value and normalizing them to falling in the desired range.
- Outlier channel splitting [17] : linear quantization is sensitive by large inputs. The idea of OCS is to reduce the value of outliers (for both weights and activations) duplicating the node with halving the output or the weight. This transformation leaves the node functionality equivalent while at the same time it narrows the weight/activation distribution allowing a better linear quantization.
- Analytical Clipping for Integer Quantization [18]: it represents the state-of-the-art for the post-training quantization techniques. It basically consists into apply a clipping function in a given range in order to reduce the quantization noise.

On the other hand, a quantization-aware training can also be done [19].
Quantization has lead to a relief of hardware computation, as it is very well-known floating-point operation are much more expensive than integer operation from a lot of perspectives, and as consequence a reduction into the power consumption of the algorithm. It is also important to mention that the data traffic between the memory and the hardware is reduced due to the compaction of data.

Nowadays, edge devices take advantage of lower precision and quantized operations, including GPUs. Thus, quantization of machine learning algorithms is a defacto standard for edge inference.

2.4 Applications

In principle the AI can be applied to any intellectual tasks [13]. Focusing on machine learning applications, they can spread through a variety of different domains:

- Healthcare, mainly used for classification purposes.
- Automotive, used in self-driving cars.
- Finance and economics, to detect charges or claims outside the norm, flagging these for human investigation. In banks system for organizing operations, maintains book-keeping, investing in stocks and managing properties.
- Cybersecurity, automatically sort the data in networks into high risk and low-risk information.
- Government, for paired with facial recognition systems may be used for mass surveillance.
- Video games, it is routinely used to generate dynamic purposeful behavior in non-player characters.
- Military, enhancing Communications, Sensors, Integration and Interoperability.
- Hospitality, to reduce staff load and increase efficiency.
- Advertising, it is used to predict the behavior of customers from their digital footprint in order to target them with personalized promotions.
- Art, it has inspired numerous creative applications including its usage to produce visual art.

However, all the Machine Learning applications are characterized by the need of a huge amount of data set for the training process.

3

State-of-the-Art

3.1 Overview

The role of Machine Learning has continuously growth in the past few years and a lot of efforts has been done for developing good software APIs in order to address different needs and domains.
In principle, all the machine learning algorithms can be run on the CPU, which already runs the OS and other Application Software, this leads to a lot of overheads, especially in terms memory accesses which are expensive in terms of energy and latency.

Analyzing machine learning algorithms it comes evident that they massively do the same operations and access to data with some kind of patterns. Thus, with the outcome of the new paradigm for the GPU, the General Purpose GPU programming it comes in handy that implementing those algorithms on a GPU, which matches the Machine Learning algorithms requirements regarding the massive operations and the reuse of data, has given a lot of advantages in terms of latency and energy efficiency. However, the capability of GPU of running machine learning algorithms has been pushed almost at the maximum with the increase of computation demands in modern neural networks, therefore other solutions have been explored, such as the development of specific hardware platform.

3.2 GPU

The Moore's law is reaching the end from the point of view of CPUs. However, it seems that the GPUs can still carry on the Moore's law [20].
For this reason, a lot of improvements especially from the companies have been made for developing more and more GPUs with a lower performance per watts.
As already mentioned, with the income of general purpose GPU programming paradigm, more and more machine learning algorithms have been designed for being run on the GPU, gathering the best fruits given by that type of architecture.

As consequences, companies such as Nvidia have started to develop GPU for boosting machine learning applications performance.

3.2.1 Nvidia Ampere A100 Tensor Core GPU

The Nvidia Ampere A100 Tensor Core GPU has been announcer recently and it is one of the most performant GPU. The newly added Tensor Core Unit allows massive increases in throughput and efficiency.
It is able to deliver up to 624 TFPLOPS[1] for training and inference machine learning applications.

The GPU is composed of multiple GPU processing clusters (GPCs), texture processing clusters (TPCs) and streaming multiprocessors (SMs). The core of the GPU is the Streaming Multiprocessor, which is built up from the SM of Volta GPU and Turing one.

Figure 3.1: Streaming Multiprocessor Architecture [21]

Composed of integer, FP32, FP64 units and the Tensor Core Units designed specifically for deep learning. It introduces also new data types in the tensor core for the computation, binary, integer 8 and 4 bits, floating-point 64, 32, 16 and bfp16 (the throughput of the tensor core computation for fp16 and bfp16 is the same). The Ampere SM can achieve such efficient workload on mixed computation and addressing calculations thanks to an independent parallel integer and floating-point data paths.

[1] floating-point operations per second

Matrix-Matrix multiplication operations are at the core of neural network training and inference, and are used to multiply large matrices of input data and weights in the connected layers of the network. The idea is represented into the Figure 3.2 and compared to previous architectures.

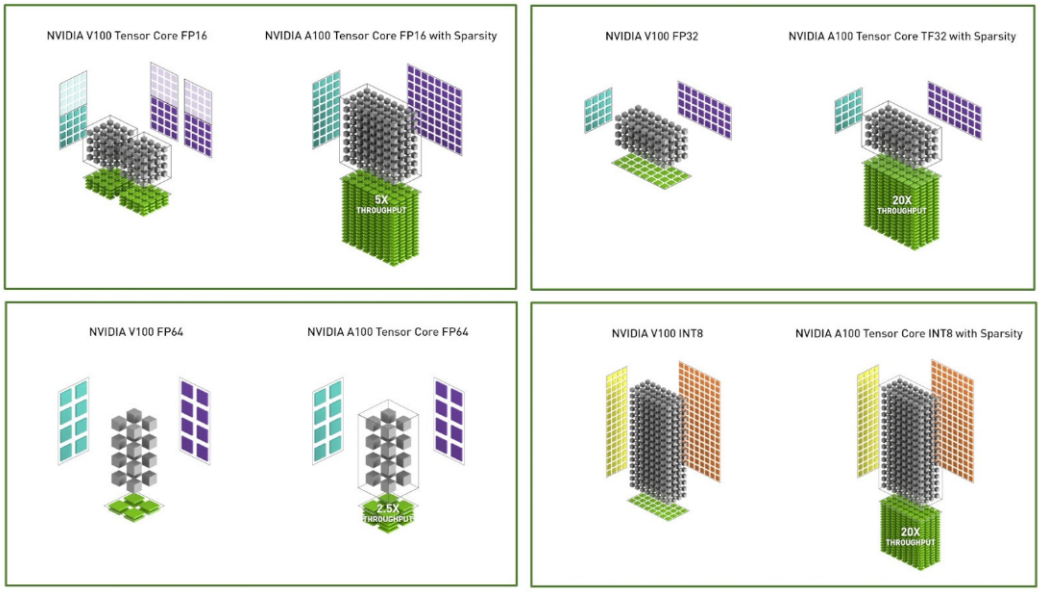


Figure 3.2: Matrix Multiplication in Tensor Core[21]

The Ampere A100 GPU contains 108 Streaming multiprocessor, and 432 third generation Tensor Core. According to Figure 3.3 the Tensor Core Units are able to compute multiplications on FP16 and accumulate on FP32, leading to a further reduction of latency and energy consumption.

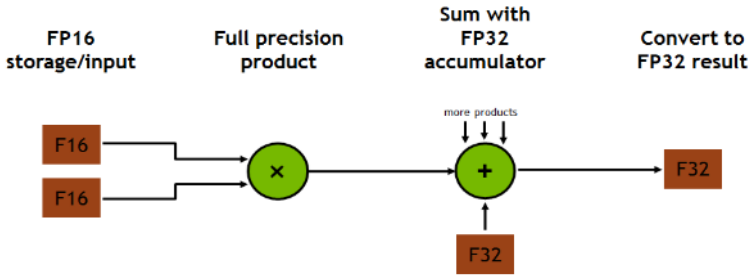


Figure 3.3: Mixed Precision Schema of a FMA unit in Tensor Core Unit[21]

A novel approach for doubling the throughput of deep neural networks has been introduced in this architecture. At the end of training process, only a subset of the total weights are necessary to execute a neural network correctly. As consequence not all the weights are needed, and they can be removed.
Based on training feedback, weights can be adapted at runtime during the training and this does not have any impact on the final accuracy. Thus, thanks to the sparsity of weight tensors., inference process can be accelerated. In addition, also

the training process can be accelerated exploiting the sparsity idea but it has to be introduced at the beginning of the process for achieving some benefits.

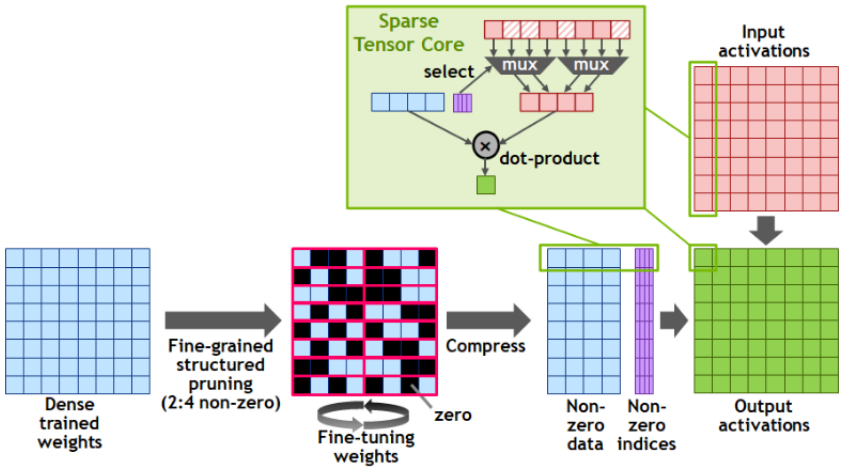


Figure 3.4: Sparsity Optmization of a weight tensor[21]

The apporach in Fig. 3.4 doubles the throughput by skipping the zeros, it also leads to a reduction of memory footprint and an increase into the memory bandwidth.

Following the idea, NVIDIA has introduced a new set of instruction for inference: sparse Matrix Multiply-Accumulate (MMA). Those instructions are able to skip the matrix entries which contain zero values leading to an increase of the Tensor core throughput. An example can be seen in Fig. 3.5, where the light blue matrix has a sparsity of 50%. It is also important to mention that the non-zero entries of the light blue matrix will be matched with the correct entries of the red one.

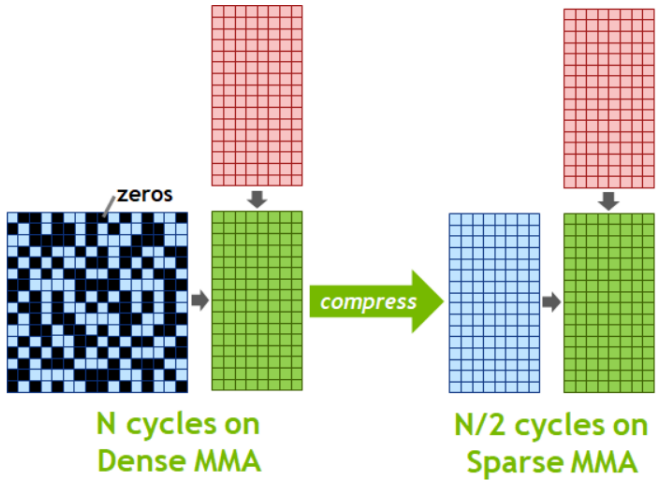


Figure 3.5: Matrix Multiply Accumulate[21]

The deep learning frameworks and the CUDA Toolkit include libraries that have been custom-tuned to provide high multi-GPU performance for each one of the following deep learning frameworks in the Figure 3.6.

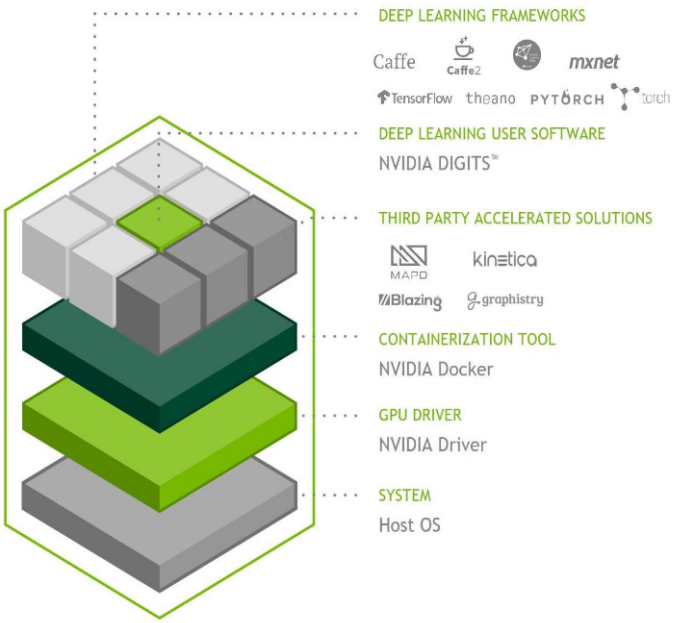


Figure 3.6: Software stack[21]

Combining powerful hardware with software tailored to deep learning, it provides to developers and researchers solutions for high-performance GPU-accelerated deep learning application development, testing, and network training.

3.3 Domain Specific Hardware Platform

Instead of developing GPUs also suitable for Machine Learning applications, the companies have designed and deployed special purpose hardware accelerators.

3.3.1 NVDLA

The Nvidia Deep Learning Accelerator is a free open source hardware platform from Nvidia, highly customizable and modular, which allows to design and deploy deep learning inference hardware.
The architecture comes in two configurations:

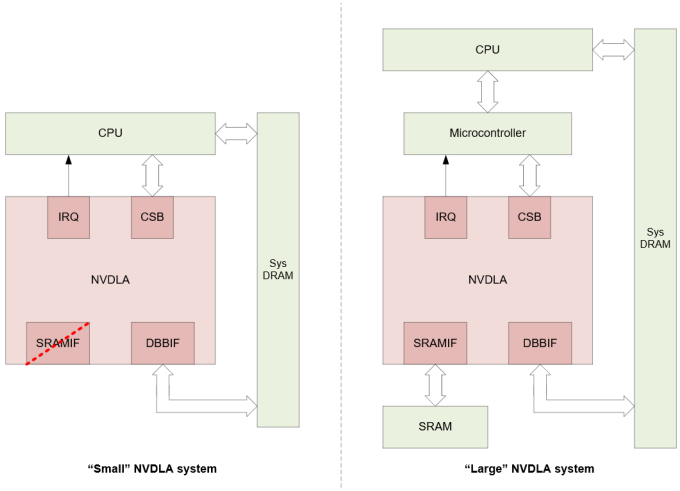


Figure 3.7: Comparsion of two possible NVDLA system[22]

As already mentioned, the aim of the work is to develop a hardware accelerator for machine learning suitable for mobile devices, therefore from now on the NVDLA small system will be considered and analyzed.
The internal architecture of the NVDLA small system is:

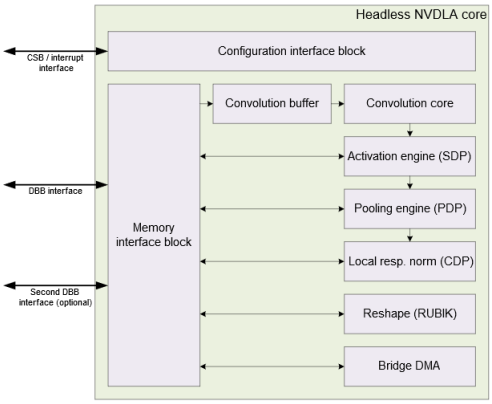


Figure 3.8: Internal architecture of NVDLA small system, Secondary DBB not considered[22]

According to Figure 3.7, for the Small configuration of the accelerator, the processor will be in charge of programming and scheduling the operations on the NVDLA and as consequences handles the start/end of operations and possible interrupts, all of them through the CSB (Configuration Space Bus) interface which is AXI Lite compliant[23].
The data movement to/from memory are handled by the Internal memory controller though the DBB (Data BackBone) interface, which is AXI [23] compliant.

The internal architecture of NVDLA is composed by various engines, each one of them is able to perform specific Machine Learning models:

- Convolution Core: it comes in pair with the Convolution Buffer, its private memory for the data (inputs and weights). It is used to accelerate the convolution algorithms.
- Activation engine (Single Data point Operations): it performs post processing operations at the single data element level such as bias addition, Non-linear function, PReLU (Parametric Rectified Linear Unit) and format conversion when the software requires different precision for different hardware layers.
- Pooling engine (Planar Data Operations): it is designed to accomplish pooling layers, i.e. it executes operation along the width and height plane.
- Local response normalization engine(Cross Channel Data operations): it is designed to address local response normalization layers.
- Reshape(Data memory and reshape operations): it transforms data mapping format without any data calculation.
- Bridge DMA: it is in charge of copying data from the Main Memory to the SRAM of the accelerator, only available into the large configuration of the system.

Another possible configuration which is worth to mention is the possibility to let the engines work separately on independent task or in a fused fashion where all of them are pipelined, working as a single entity.

According to developers the configurability of the cores ranges from arithmetic precision to the theoretical throughput that a single unit can achieve (increasing the number of internal Processing Elements). Moreover, since the engine units are independent of each other, according to the application and the model used they can be safely removed from the design.

3.3.1.1 NVDLA Software

It is also worth to mention that the accelerator comes already with a basic software stack:

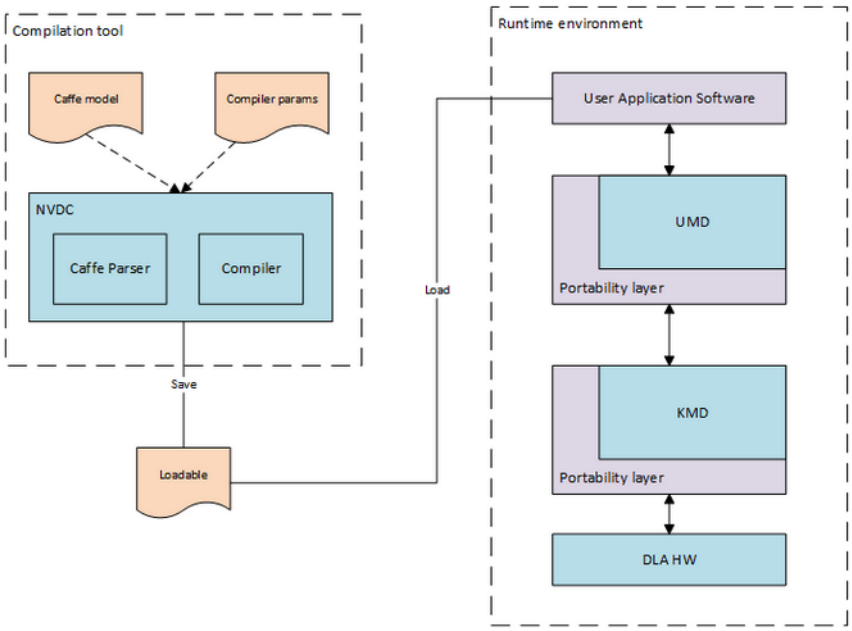


Figure 3.9: NVDLA Software stack[24]

Where the Compilation tools are in charge of converting existing pretrained model into a set of hardware layers (for the desired precision) and programming sequences suitable for the NVDLA, the output of this process is a Nvidia Loadable file suitable for the runtime environment.
Regarding the runtime environment, it has been designed for a system in which is present an OS. It is composed in two parts: the User Mode Driver (UMD) and the Kernel Mode Driver (KMD).
The User Mode Driver loads the loadable file in memory and submits the operation to the KMD, it is also in charge of data movement from/to the accelerator.
The KMD is in charge of submit operations to the accelerator through low level functions, schedules the operations and handles the interrupts.
Both the KMD and the UMD are wrapped into portability layers which are, respectively, hardware dependent and OS dependent. In principle, for migrating the software to another OS or hw plaftorm it is enough to modify only the portability layers.

3.3.2 Google TPU

Google developed its own application-specific integragrated circuit for neural networks, which is tightly integrated with TensorFlow Software. It includes:

- Matrix Multiplier Unit (MXU): 65,536 8-bit multiply-and-add units for matrix operations
- Unified Buffer (UB): 24 MB of SRAM that work as registers
- Activation Unit (AU): Hardwired activation functions

In Figure 3.10 a general view of TPU architecture is presented.

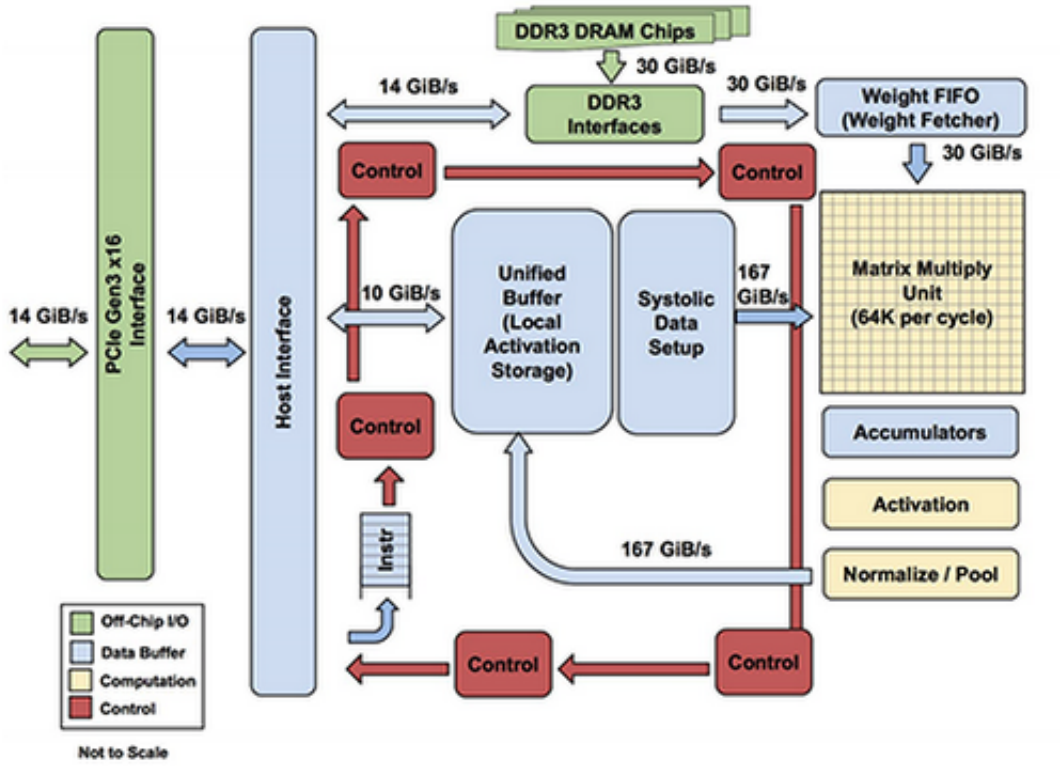


Figure 3.10: Google TPU architecture[1]

Rather than be tightly integrated with a CPU, the TPU is designed to be a coprocessor in which the instruction are sent by the host server rather than fetched.

The matrix multiplication unit reuses both inputs many times as part of producing the output avoiding the overhead of continuously read data from memory. Only spatial adjacent ALU are connected together, which makes wires shorter and energy-efficient. The ALUs only perform computations in fixed pattern.

As far as concern the software stack, the TPU can be programmed for a wide variety of neural network models. To program it, API calls from TensorFlow graph are converted into TPU instructions.

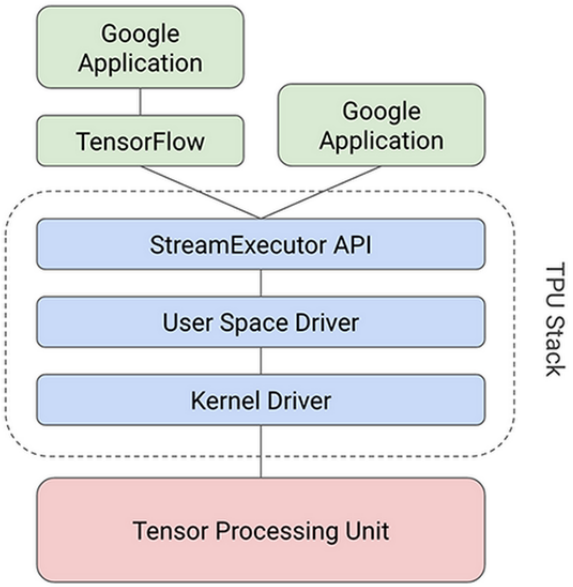


Figure 3.11: Google TPU Software Stack[25]

3.3.3 Habana Goya HL-1000

Habana's Goya is a processor dedicated to inference workloads. It is designed to deliver superior performance, power efficiency and cost savings for data centers and other emerging applications.
It allows the adoption of different deep learning models and is not limited to specific domains. Moreover, the performance requirements and accuracy can be user-defined.

In the Figure 3.12 a high level view of the Goya architecture can be appreciated.

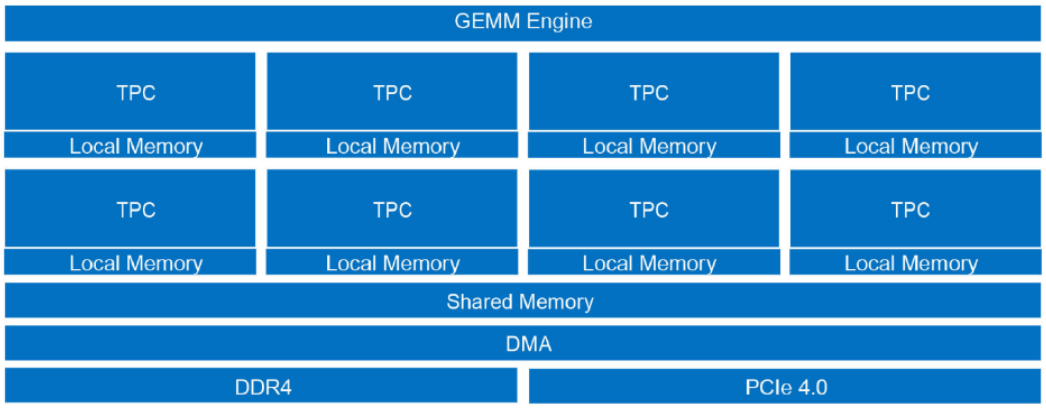


Figure 3.12: High level view of Goya architecture[4]

It is based on scalable, fully programmable Tensor Processing Cores, specifically designed for deep learning workloads.
It also provides other flexible features such as GEMM operation acceleration, special functions dedicated hardware, tensor addressing, latency hiding capabilities and different data types support in TPC (FP32, INT32, INT16, INT8, UINT32, UINT16, UINT8).

Regarding the software stack, it can be interfaced with all deep learning frameworks. However, a model has to be first converted into an internal representation, as it can be seen in Figure 3.13.

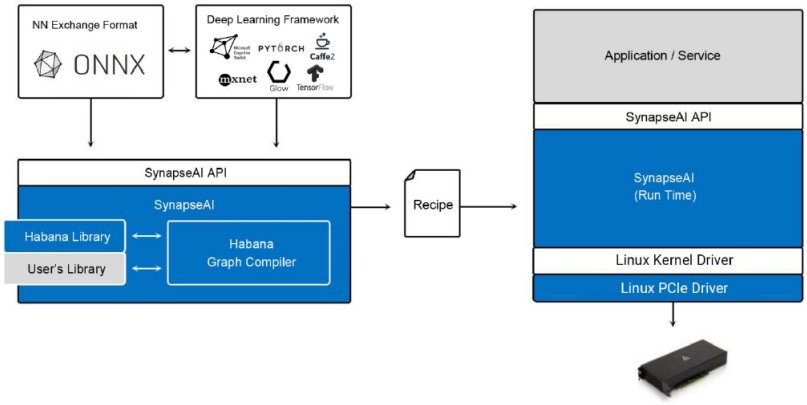


Figure 3.13: Habana Goya Software Stack[4]

It also supports quantization of models trained in floating-point format with near-zero accuracy loss.

4
System Development

4.1 Overview

As already mentioned, the use of custom hardware for a specific application can have big benefits especially in terms of energy consumption and latency. The inference process of Neural Network is mainly characterized by massive multiply and addition operations. Fetch of data from main memory follows patterns and it has been proved those data, in particular weight data, are reused for several execution of the Neural Network model. As consequence, executing a Neural Network model on a Von-Neumann based architecture machine leads to performance degradation, even in a cache-based system, since the CPU has to request the data from the main memory, execute the operation on those data and then save back to main memory before moving to the next data. The introduction of vectored instruction in the modern processors can have a slight impact in the performance benefits. However, the drastically increase of layers in the Neural Network has made them suitable for several applications, this it can be translated into a massive increase of operations for executing them. Following the fast demands of operations into a Neural Network, it becomes evident that executing them on a CPU could not meet anymore real-time application requirements.
Instead, the designed accelerator has a Dataflow architecture, with emphasis on weight data reuse, and it is able to execute a tensor convolution. The basic idea is a computation matrix composed in every entry of processing elements which are able to perform operation between the incoming data and the weights, which have been already loaded for exploiting a data reuse approach.

The custom hardware accelerator is not useful as it is, it has to be integrated into a ML-Framework in order to appreciate its benefits. After a preliminary research on which ML-Framework would allow to integrate a custom hardware accelerator minimizing the efforts to change the model code and its definitions, it has been evident that the TensorFlow Framework, an end-to-end open source Machine Learning platform [26], suits the needs.

The workflow of the Hardware-Software development is illustrated in the following:

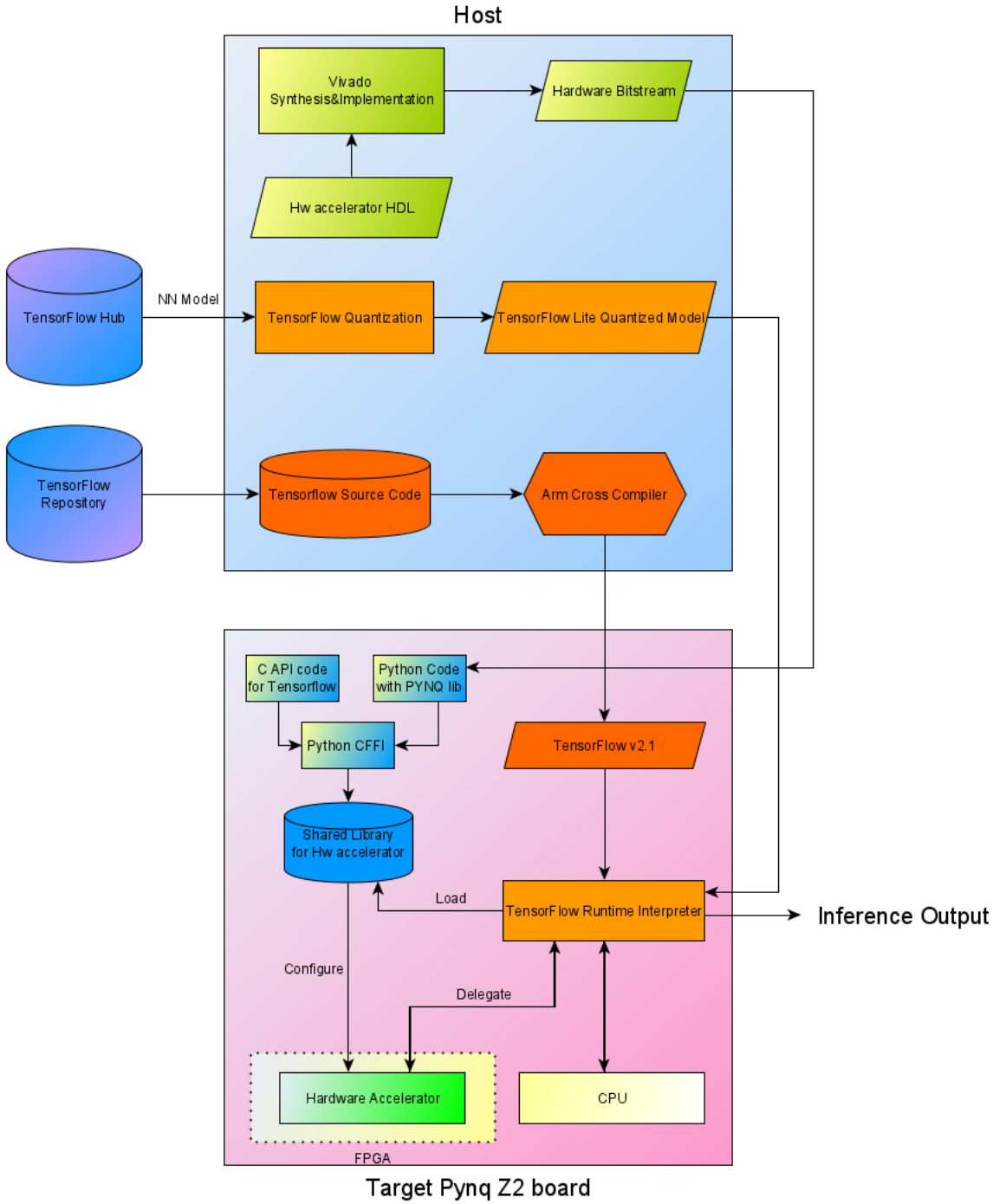


Figure 4.1: Development workflow

The entire work is implemented on a PYNQ Z2 board from TUL, based on a Zynq-7000 SoC [27]. In order to speed-up the development process and use built-in library for the AXI protocol and the DMA transfers, the software is partially carried out through the PYNQ environment of the board [28] based on Python which has become a de facto standard [29].
The usage of Python as basic software allows to easily integrate it with high level Machine Learning Framework, such as TensorFlow in this case.

4.2 Software

The focus of the work is the inference process, pre-trained models are needed and TensorFlow Hub [30] comes in handy for this purpose. It provides already pre-trained Machine Learning models for different domains. Moreover, TensorFlow has the feature of quantize a post-trained model for different arithmetic precision. In the Fig. 4.1 it can be seen that the quantization process has been done offline.
The choice of using the stable release 2.1 of TensorFlow is dictated from the possibility of using Delegates (aka hardware accelerators or GPUs) in its Neural Network model. A delegate is a way to delegate part or all graph execution to another executor. Every model is represented, internally, as a graph (with its relative order of execution for the nodes) and every node of the graph is described as a set of operation that has to be applied to the node's input. As every node is described by a set of operations, it is easy to understand which part of the graph can be executed on the accelerator in advance, and this operation is done at the beginning when both the model and the accelerator library is loaded. It is worth to mention that TensorFlow is open-source and since no binary installations for its 2.1 release are provided for Arm processor, it has been cross-compiled from scratch for the PYNQ-Z2 board.

TensorFlow demands as library for the accelerator a C Python-API compatible shared library. In addition, the code for using the accelerator was already written using the PYNQ environment in Python. Therefore, for allowing code reuse and decreasing the development time the Python code has been embedded in the C code (from a TensorFlow example of the delegate library), adding callbacks to Python code[1]. This has been possible thanks to the Python library *CFFI* (C Foreign Function Interface)[31], which is also able to provide a shared library Python-API compatible as output.

[1] See Appendix 1

4.3 System Level

As it can be seen from 4.2, it is divided in two big blocks:

- Processing System: The processing system (in Fig. 4.3 referred as *processing system7*) is in charge of running the OS and the Machine Learning application, as consequence it also runs the necessary software for programming the accelerator registers and the data movement to/from main memory from/to the accelerator.
- Programmable Logic: The programmable logic (PL) hosts the entire design, from the accelerator itself to the DMAs and the AXI interconnections.

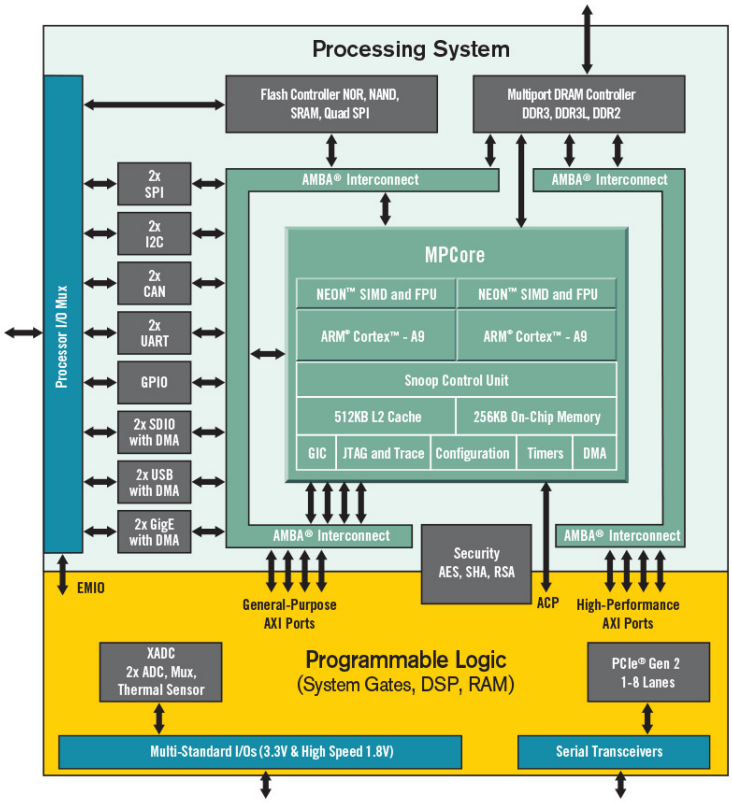


Figure 4.2: Zynq 7000 SoC[32]

Furthermore, the Programmable Logic in Fig. 4.3 is hosting:

- AXI interconnections: IP cores from Xilinx[33][34] in order to connect and correctly address entities in the Programmable Logic.
- AXI DMA: IP core from Xilinx [35] which allows data movement between main memory and accelerator memories. Several single channel DMA have been used instead of using a single DMA with multiple channels, the reason is that in the PYNQ environment only the drivers for the single channel DMA are provided.
- DTPU: the actual hardware accelerator.
- XADC: IP core from Xilinx [36] which allows to measure the temperature of the SoC, the voltages and the currents at run time.

In the following figure, the schematic of the overall design in the PL is presented.

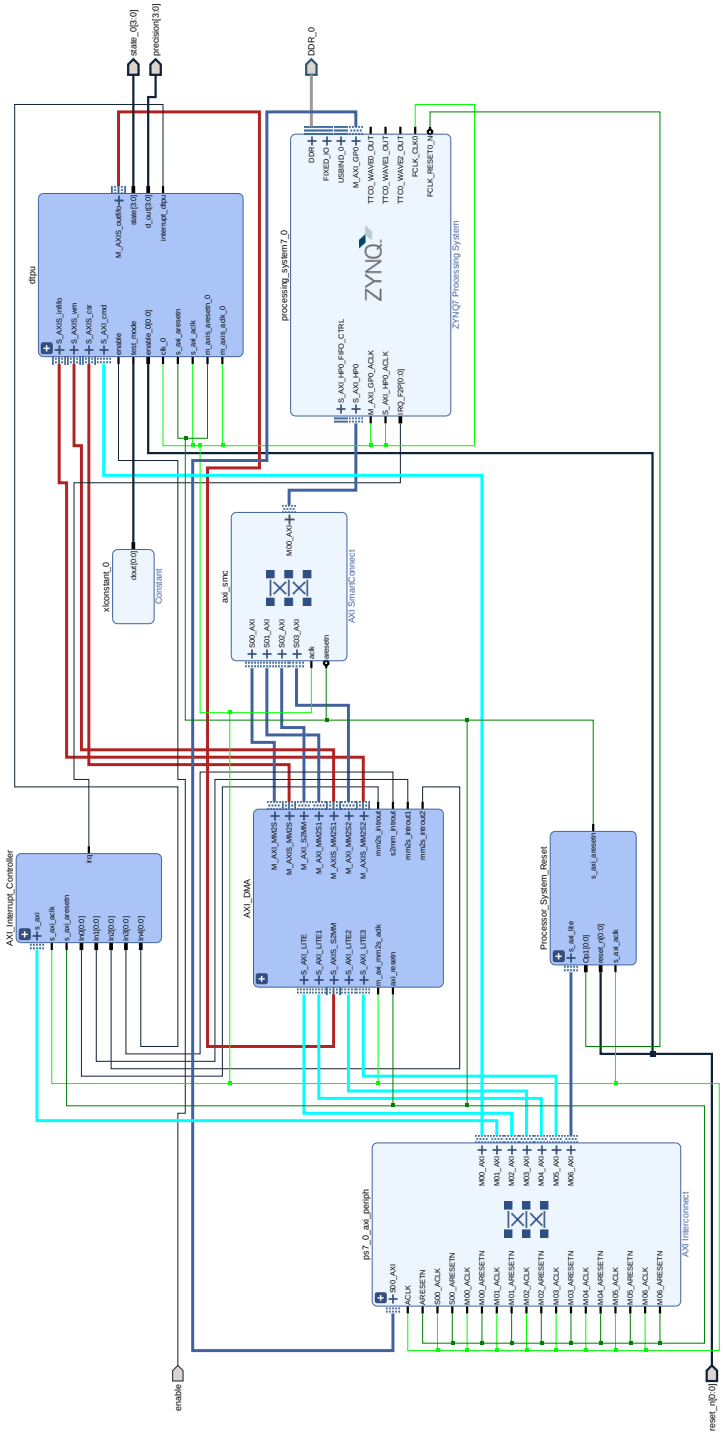


Figure 4.3: System view hosted in the PL [2]

[2]Except for the Zynq Processing system

4.4 DTPU, the hardware accelerator

The hardware accelerator, named *Cogitantium*[3], *The Dumb Tensor Processing Unit*, is in charge of carrying out the tensor convolution of the neural network model, exploiting a data-flow architecture on the input data and a data reuse for the weight data.
In Figure 4.4 is presented the Logical block diagram of the accelerator.

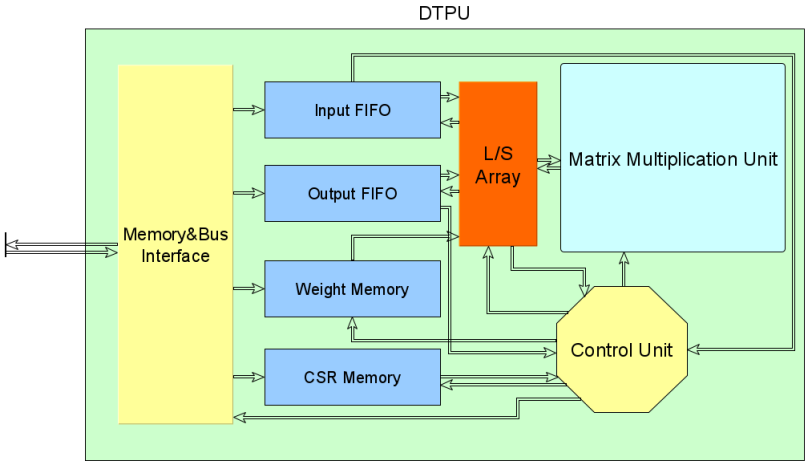


Figure 4.4: Logical view of DTPU accelerator

4.4.1 Real Implementation

The work is not focused on developing embedded memories and AXI interfaces, therefore a Xilinx's IP core, which includes all those necessary sub components, has been used[37] leading to the actual block diagram which can be observed in the Figure 4.5.

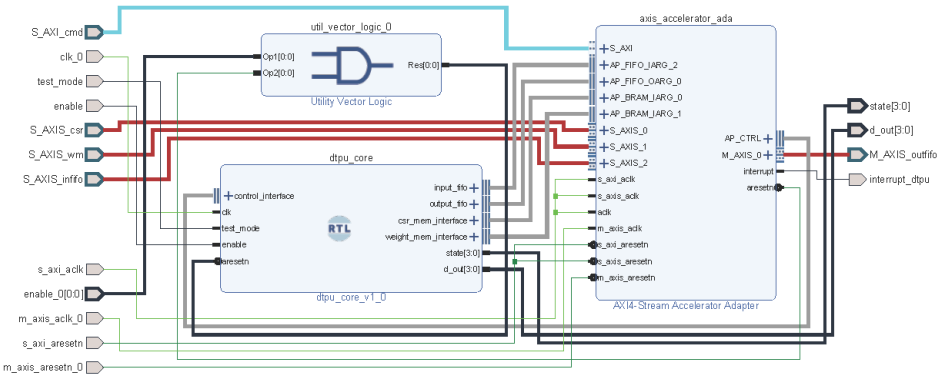


Figure 4.5: Real RTL view of DTPU accelerator

[3]Thoughtful

The latter has allowed to completely focus the work on the DTPU core, which has become:

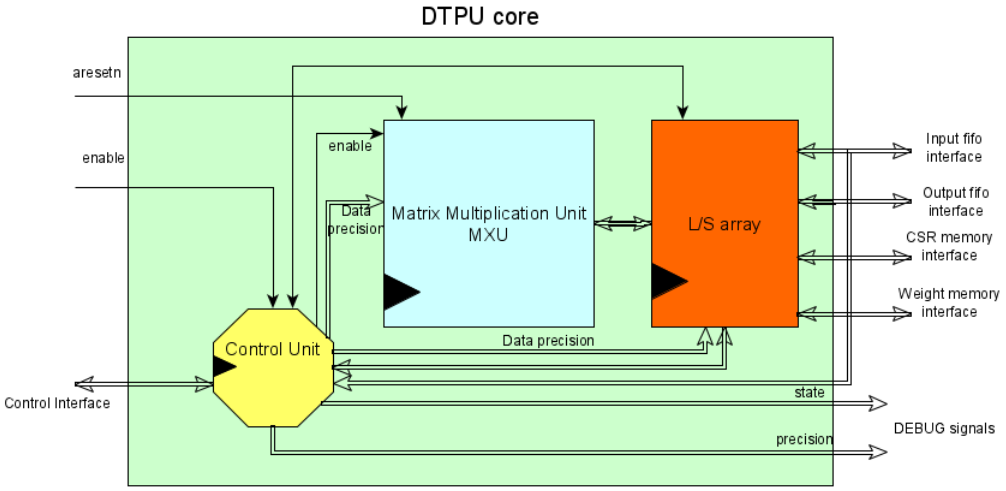


Figure 4.6: RTL view of DTPU core

Where the sub-units:

- L/S array provides the data for the Matrix Multiplication Unit, especially the weight data are reused across several executions and therefore loaded once.
- Control Unit is in charge of handling handshake signals for transferring the ownership of the data (data transferred by the DMA from the Main Memory), load the weights and activation in the respectively units and save the results to the output FIFO. Since it is a Data flow architecture, there is no control flow of the data in the core and this has allowed to keep the Control Unit as simple as possible.
- Matrix Multiplication Unit (Mxu) is the computation unit of the hardware accelerator, it executes the tensor convolution for different arithmetic precision.

4.4.2 High Level State Machine of Control Unit

The Dataflow architecture has allowed to design a Control unit as much simple as possible, presented in the below figure:

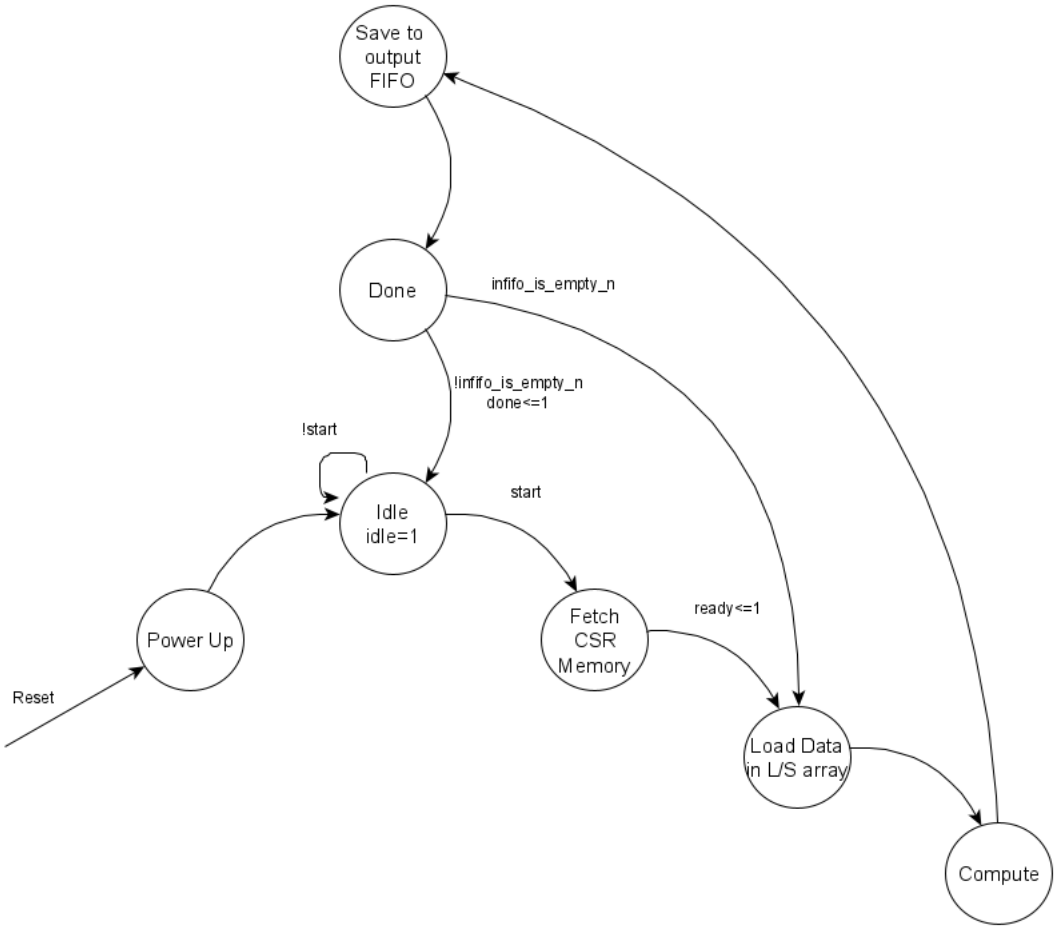


Figure 4.7: A high level view of Control Unit

In which:

- *Idle* state is waiting for the start signal from the *axis accelerator adapter* (generated when all the data have been transferred[4]).
- *Fetch CSR Memory* state is in charge of retrieve from the CSR memory the desired data precision for the computation and the starting address of the weight memory. It also notifies to the *axis accelerator adapter* that it is ready[5].
- *Load data in L/S array* state loads the correct weight values (retrieved from the weight memory) and the activation data into the correct L/S unit. The number of active L/S unit is computed at run time, it depends on from the current required data precision and the fixed number of rows and columns in the MXU.

[4]Input Data, Weight data and CSR data

[5]The ready signal is used as handshake between the core and the axis accelerator adapter for transferring the ownership of the data

- *Compute* state activates the MXU and it waits the end of computation before committing the results to the output FIFO.
- *Save to output FIFO* state saves the data stored in the active L/S units to the output FIFO.
- *Done* state, depending on the input FIFO if it is empty or not, continues the computation for the next activation data or it returns to the idle state, notifying to the axis accelerator adapter the end of the computation[6].

4.4.3 Datapath

As it is well-known, the execution of ML models is memory intensive and it consists in massive multiplication and accumulation operations. In addition, it can be seen also that during execution of ML models some memory location are accessed frequently. Therefore, it is evident that a DataFlow architecture which could exploit local data reuse and compute, massively, in parallel multiplications and additions could boost the performance. The DTPU core has been designed according to the previously mentioned ideas. The datapath of the core is presented in Figure 4.8 as block diagram.

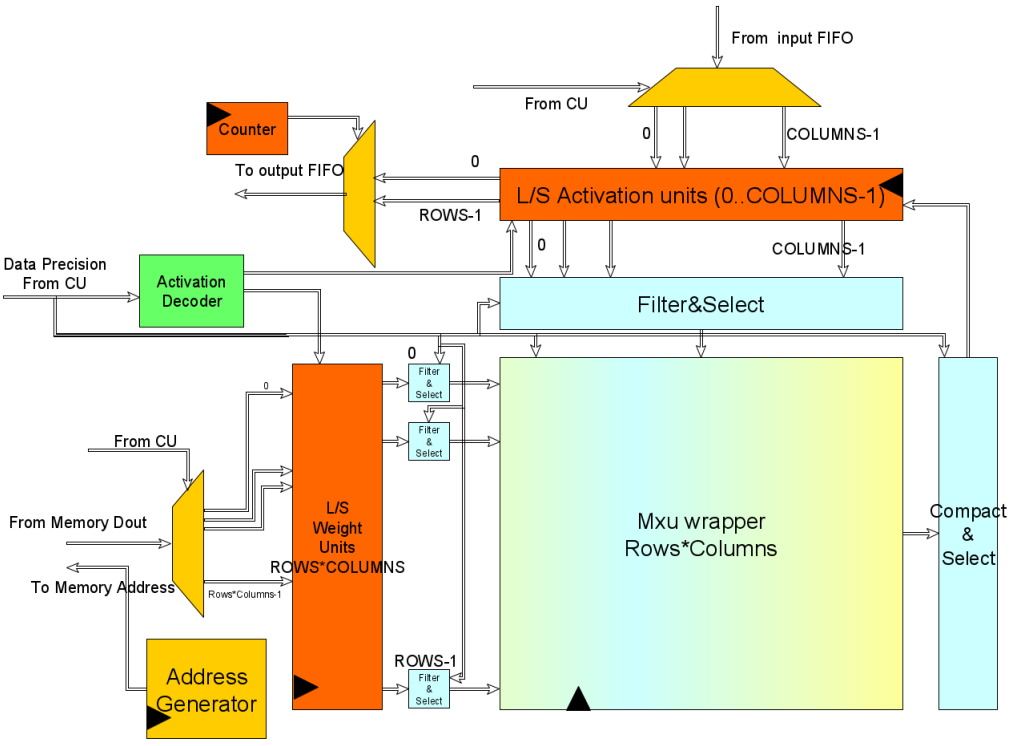


Figure 4.8: A detailed view of the DTPU core datapath. Enable and Resets signals for clocked units has been omitted for improving readability.

[6]the notification for the end of computation allows the axis accelerator adapter to put the results on the output master axi stream interface in order to be transferred by the DMA

In Fig. 4.8, the brawn of the accelerator is the MXU wrapper, which contains the symmetric matrix of MACs with variable precision. Regarding the other blocks:

- Activation Decoder: It is able to generate the right activation signals for the L/S units, depending on from the current data precision and MXU size.
- Muxes and DeMuxes: Their purpose is to feed the right data from/to memory to/from the right units. The counter (from 0 to ROWS-1) in the Mux for the output FIFO is for saving at every clock cycle a data in the FIFO.
- Filter&Select: depending on the precision it provides the correct data to the correct computation units.
- Compact&Select: it is the complement of the Filter&Select unit, it is able to compact the output data from the MXU wrapper and feed the store registers.
- L/S weight Units: the name L/S has been kept for consistency even if it does not have any store process since the weight are only loaded once (stationary weights) and kept until a next full execution.
- L/S Activation Units: they are in charge of loading the data from the input FIFO into batteries of Flip-Flops while at the same time they can save the results to submit late in the output FIFO.

4.4.3.1 Filter&Select and Compact&Select

In principle, for each Processing element in the MXU wrapper a weight and an activation has to be provided (and as consequence it has to be provided from its relative Load Units). However, since the data width of memories and FIFO has been fixed to its maximum, 64 bits, it comes evident that during a computation with 8 bit integer it will fetch(and save for the output FIFO), in case of a 8x8 Mxu Size, 8 values from FIFO and 64 values from the weight memory. In this scenario all the Flip-Flops of the L/S units (both activation and weights) would sample values where the 56 upper bits are always unused leading to a waste of time for the memory accesses and energy for unused data.

A clever solution is to pack data before sending them to the accelerator. Nevertheless, the pack of data requires to internally unpack and, before committing to the output FIFO, pack the results. Unpacking and packing are done, respectively, by Filter&Select and Compact&Select units. Retrieving the previous example (computation on 8 bit integer, MXU size of 8x8 and 64 bit memory data) and using the approach of unpacking and packing, this leads to use only one L/S unit for activations (8 for the L/S weight units) for both the load and store operation. With one single L/S active unit and 8 bit integer computation, an 8 bit activation data has to be distributed for each column of the MXU, and this is done by the Filter&Select unit. For committing to output FIFO, results on 8 bit will be compacted in one single data of 64 bit by the Compact&select.

A visual distribution of the data can be seen in Fig. 4.9, the same can be applied for each row of L/S Weigth units.

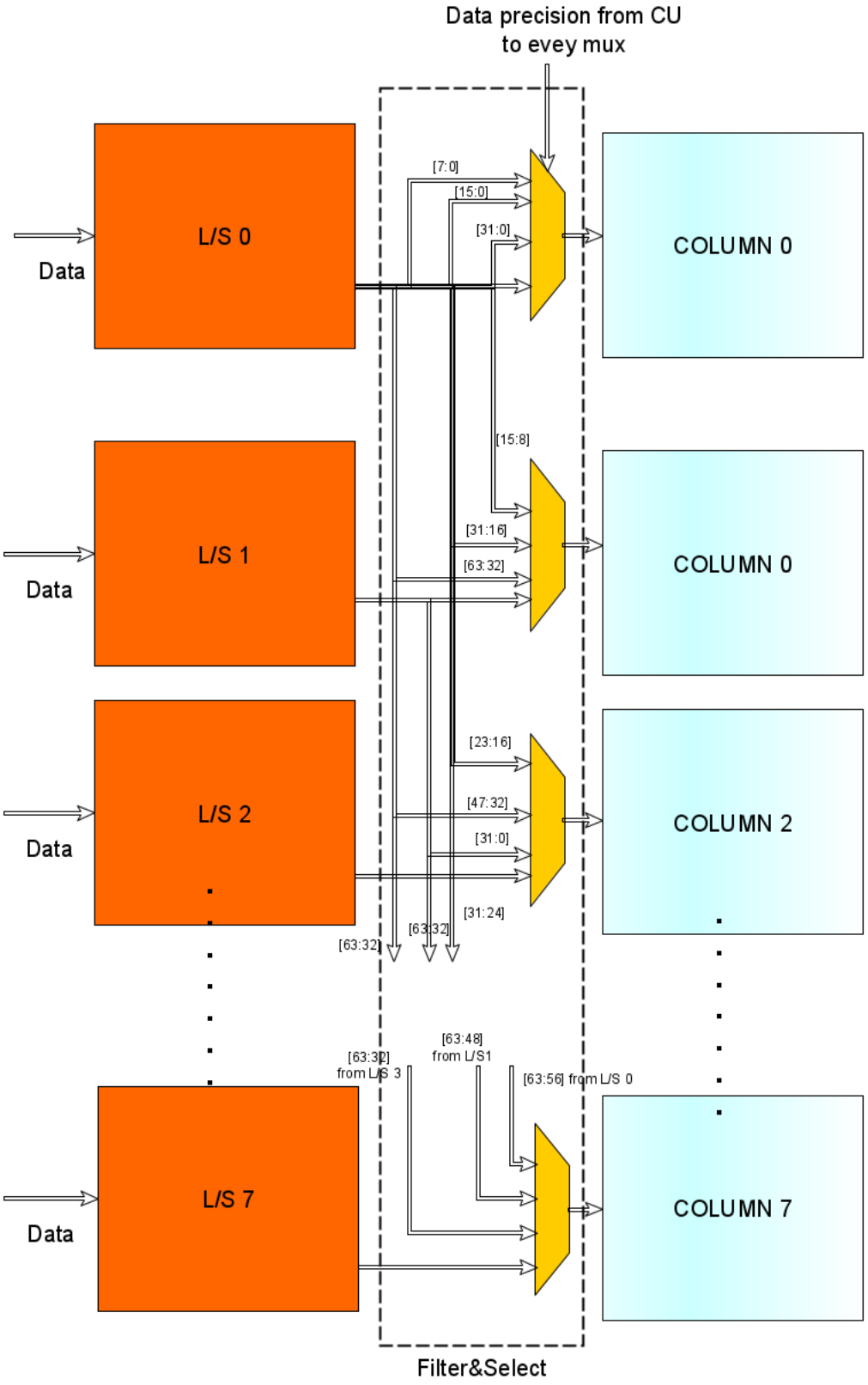


Figure 4.9: Data Distribution of Filter&Select unit for a MXU size of 8x8

In case the required precision is on 16 bit, with the same MXU size, two L/S units for activation are activated (2*ROWS for the L/S weight units) and will feed the respective Columns. The reason behind the two active L/S units is that in 64 bit, only 4 16-bit values can be packed. Increasing the MXU size, the L/S units are activated accordingly. For example, in case of a MXU size of 16x16 and integer 8 bit, two L/S units are activated (in case of integer 16 computation,4 units are activated).

This approach comes also with the overhead of packing and unpacking the data on the CPU but, on the other hand, the memory data movement are reduced and bandwidth increased, with a reduction in the energy consumption (thanks also to the reduced active L/S units).
It is also worth to mention that using size for the MXU which are power of two would maximize the memory bandwidth.

4.4.3.2 Matrix Multiplication Unit

The Matrix Multiplication Units (referred as MXU) is the muscle part of the accelerator, where the convolution is done. As the name suggest, it is organized as a Matrix:

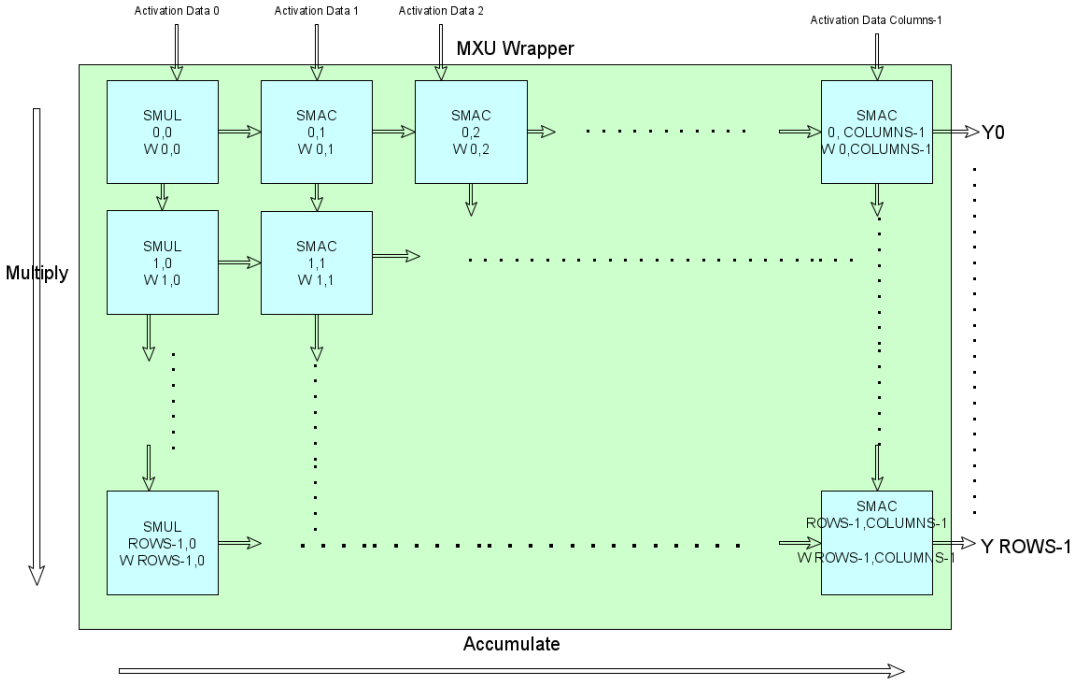


Figure 4.10: MXU interal structure and weights distribution

Every sub units has its own weight value (distributed thanks to the L/S weight combined with Filter&Select units, see Fig. 4.8). It is a homogeneous unit, except for the first column, which does not accumulate. In addition, as it can be seen from the block diagram, there is no control flow between every Processing units, there is only data exchange from the previous unit to the next one (for both axis). This matrix configuration of the hardware allows to massive multiply and accumulate at the same time, in particular it can compute:

$$MAC_{OPS} = ROWS * COLUMNS \text{ per } \# \text{ clock cycle required for a single unit}$$
$$\text{with a } Throughput = Rows$$

The MXU can be synthesized with different criteria.
In particular, the Processing Elements can be independently generated for a single data precision, from integer 8/16/32/64 to floating-point 32 or brain floating-point 16, or with some precision at the same time. Then data precision is decided, via software, and properly controlled using signal in 4.11.

A detailed view of SMAC (Sub unit Multiply and Accumulate) and SMUL(Sub unit Multiply), the Processing Elements, is given in 4.11.

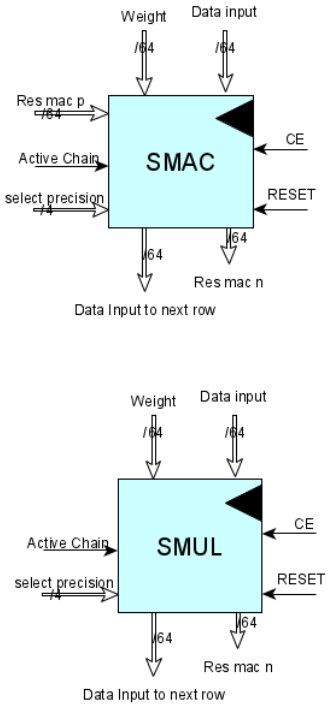


Figure 4.11: SMAC and SMUL details

It is important to mention that the sub units are always receiving data on 64 bits even if internally they may use all of them or not, depending on the value of *select precision* and *active chain* signals. For the full integer configuration (64 bit width operations) beside the possibility of computing for different data width (i.e. choose between 8/16/32/64) with the help of *active chain* signal (active low, otherwise it is a 64 bit computation) and data width fixed to 64 bit, the Processing Elements can compute vectorized operations. Therefore, it is able to compute at the same time two 8-bit, one 16-bit and one 32-bit operations (multiplication for SMUL and multiplication and addition for SMAC). However, this comes with the overhead of correctly packing and unpacking the data on the CPU before transferring them to the accelerator.

SMAC and SMUL units have been designed, internally, using Vivado DSP primitives [38], which a general schema can be appreciated in Fig. 4.12:

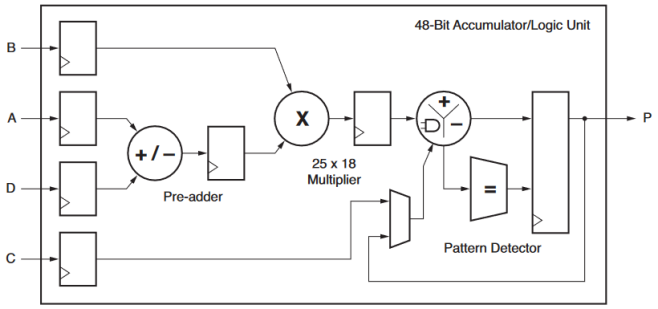


Figure 4.12: DSP Slice Functionality[38]

Allowing fitting two computation (referring to SMAC) in one single unit[7] and maximize the resource utilization.

As soon as the Synthesis process reach the maximum value of DSP utilization, it does not switch automatically to use Fabric for those primitives. For maximizing the resource usage of the FPGA, the DSP primitives have been regenerated for both Fabric and DSP blocks. In this way, during the generation algorithm for the MXU, it uses primitives for DSP up to the maximum allowed value for the given board and then it starts to utilize Fabric. This approach has allowed almost a full utilization of the FPGA resources.

[7]Only for integer 8 and 16

5
Results

If you can not measure something, you can not improve it.
— William Thomson Kelvin

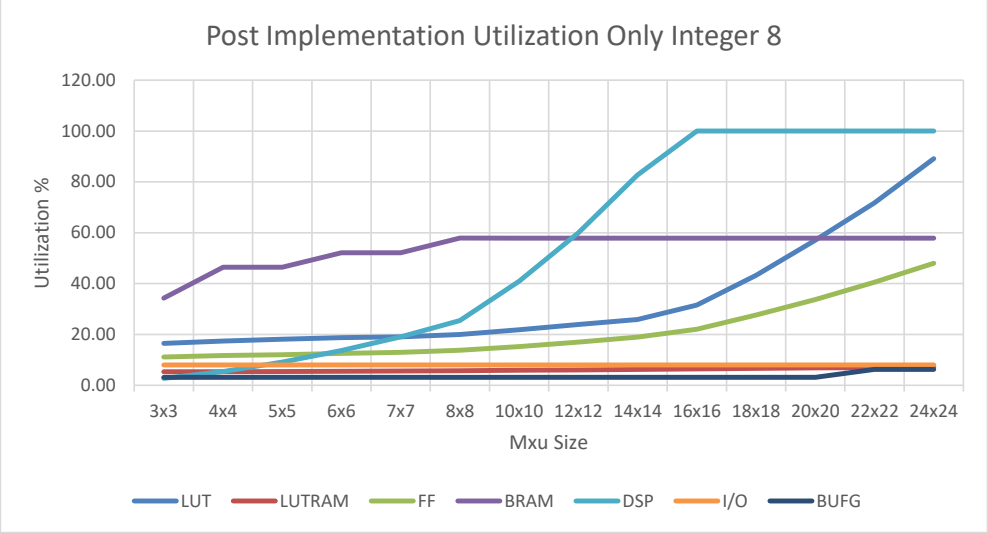
5.1 Evaluation metrics

Generally speaking in Computer Science, every domain and application could have different evaluation metrics, for example the energy efficient of a CPU is a heavy metrics in embedded systems while in a high performant CPU latency and throughput are dominant metrics. As said that, evaluation metrics strongly depend on the end-users, therefore the designers have to make assumption on the end-user intentions and applications.
In this work the assumptions are that the accelerator will be deployed into an embedded system and at the same time it should give to the user a certain degree of flexibility for running Neural Network models. Thus, as it is suggested [5] the following metrics are used:

- Accuracy, quality of the final result of inference process.
- Throughput, for measuring real time performance. It depends on the number of internal computation cores.
- Latency, for interactive applications.
- Energy and Power.
- Hardware cost (Utilization Factor in case of an FPGA) of chip area and process technology.

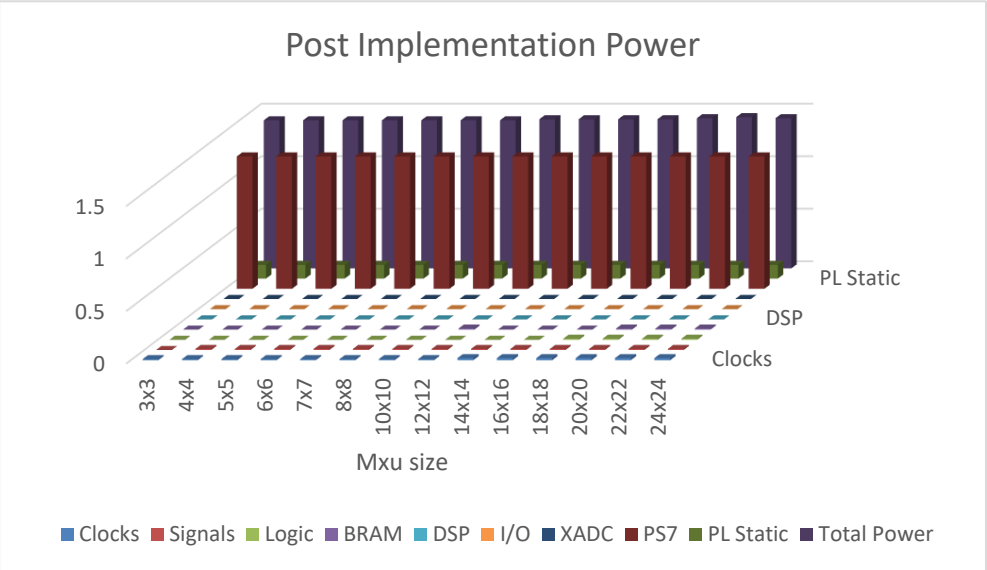
5.2 Utilization Factor

An important aspect of an embedded system is the on-die utilization area. Those kinds of system are usually deployed on tightly area constrained chips for hiding their presence to the user. Therefore, it is important to measure and understand the behavior on the Utilization of the FPGA (used as area measurement in this case) of the design as the size of Matrix Multiplication Unit increases and in parallel the throughput.
The Utilization Factor, composed of Look-up-Table, Flip Flops and Digital Signal Processor usage, is expected to increase as the size of Multiplication Matrix increase and the bit width of Computation Unit.
In the following, utilization results are presented for each data type:



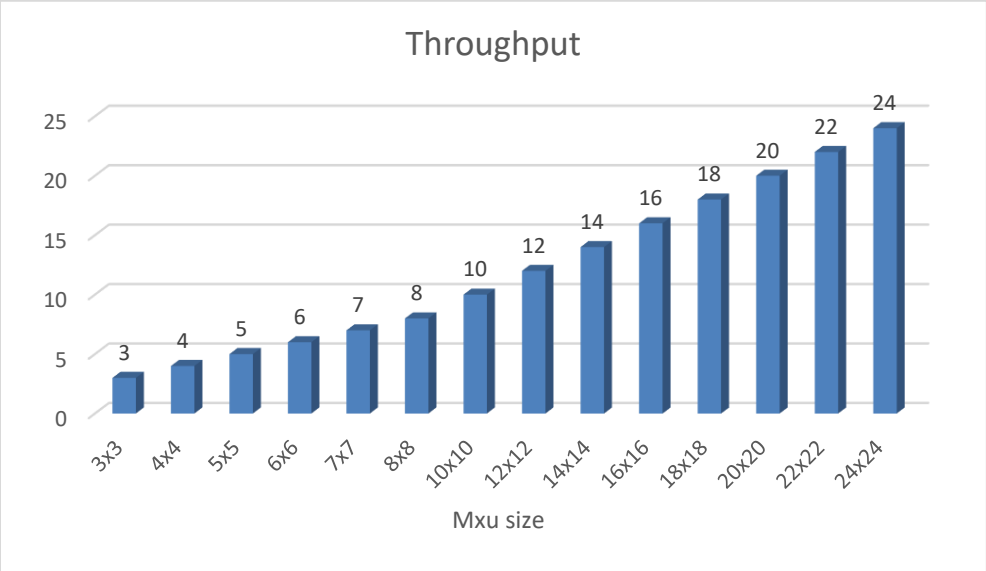
5.3 Energy and Power Consumption

Energy and Power consumption are important factor, for a mobile device in which there is a limited battery capacity meanwhile for data centers stringent power ceilings due to cooling costs.
In the following, estimation of power consumption from Vivado are presented for each data type:



5.4 Throughput

According to the definition, Throughput is the amount of units of information a system can process in a given time. As said that, for the designed accelerator it results to be equal to the number of rows into the Matrix Multiplication Unit. Normalizing this value with the clock frequency, it results to be constant for all the data type and frequencies:



5.5 Latency

In a real time application, the most important factor is the latency, the execution time of a task. In this case the latency is measured as average of the execution time of a Neural Network model for different platforms. In addition, the execution of the models, on the target, in the configuration *CPU+accelerator* is done with different clock frequencies and data type in the Programmable Logic, and as consequence a different overall latency (and power consumption).
In the following tables, the execution type for different data type and model is presented (with a fixed clock frequency of the accelerator at 80 MHz).

Model	CPU (host)[1]	GPU(host)[1]	CPU(Pynq Z2 board)[3]	CPU(Pynq Z2 board) + accelerator
MNIST	0.3 ms	5.7 ms	3.8 ms	
Mobile Net V2		62 ms		
Cifar 10	20 ms	22 ms	343 ms	

Table 5.1: Execution Time for different platform and model, integer 8

On the other hand, the clock frequency of the accelerator can be tuned. Therefore, in the following, the behavior of the latency for every model with changing in the clock frequency is presented.

graph of how the execution time on the target changes with the frequency for every model

a possible estimation of the speedup

[1]Intel i7-6700HQ, 2.60 Ghz
[2]NVIDIA, GeForce GTX960M, 1.176 Ghz
[3]Arm dual-core Cortex-A9, 650 MHz

5.6 Accuracy

list of NNs with the accuracy of the reference model(the one running on the board's cpu) with the cpu+accelerator run

Model	Reference Output [1]	Actual Output[2]	Relative Error[3]
MNIST			
Mobile Net V1 integer8			
Mobile Net V1 fp32			
Cifar 10			

Table 5.2: Accuracy Output of the and its relative error

5.7 Memory footprint

So having a chart or table with weight sizes (assuming this sis what is interesting to you since is what determines the local store of the FPGA engine) is enough. You should present the values for different bit length but no more.

[1] Inference output of the model on the target, CPU only

[2] Target CPU + accelerator with MXU size of 8x8 and clock frequency at 80Mhz

[3] $\epsilon_r = \left| \frac{OuputReference - ActualOutput}{OutputReference} \right| * 100\%$

6
Conclusion

6.1 Conclusion

6.2 Future Works

Bibliography

[1] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, et all, "In-Datacenter Performance Analysis of a Tensor Processing Unit", *CoRR* **2017**, *abs/1704.04760*.

[2] Nvidia, NVDLA, `http://nvdla.org/index.html#`.

[3] Habana, "Gaudi™ Training PlatformWhite Paper", **2019**.

[4] Habana, "Goya™ Inference Platform White Paper", **2019**.

[5] V. Sze, Y. H. Chen, T. Yang, J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey", *IEEE vol. 105 no. 12 pp. 2295-2329* **Dec. 2017**.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, "A domain-specific architecture for deep neural networks", *ACM 61 pag. 50-59* **August 2018**.

[7] Y. Cai, C. Liang, Z. Tang, H. Li, S. Gong, "Deep Neural Network with Limited Numerical Precision", **2018**, (Eds.: J. Abawajy, K.-K. R. Choo, R. Islam), 42–50.

[8] J. Johnson, "Rethinking floating point for deep learning", *Facebook AI Research* **2018**.

[9] A. Rahman, S. Oh, J. Lee, K. Choi, "Design space exploration of FPGA accelerators for convolutional neural networks", **1996**.

[10] J. T. Johnston, S. R. Young, C. D. Schuman, J. Chae, D. D. March, R. M. Patton, T. E. Potok, "Fine-Grained Exploitation of Mixed Precision for Faster CNN Training", **2019**, 9–18.

[11] H. J. L. Hao Zhang, S.-B. Ko, "Efficient Fixed/Floating-Point Merged Mixed-Precision Multiply-Accumulate Unit for Deep Learning Processors", **2018**.

[12] A. Turing, "Computing machinery and intelligence", *Mind* **1950**.

[13] S. Russell, P. Norvig, *Artificial intelligence : a modern approach.* Pearson Education Limited, **2016**.

[14] W. Maass, "Networks of spiking neurons: The third generation of neural network models", *Neural Networks* **1997**, *10*, 1659 –1671.

[15] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, A. Maida, "Deep learning in spiking neural networks", *Neural Networks* **2019**, *111*, 47 –63.

[16] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, D. Modha, "A Low Power, Fully Event-Based Gesture Recognition System", *IBM research* **2017**.

[17] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, Z. Zhang, "Improving Neural Network Quantization without Retraining using Outlier Channel Splitting", **2019**.

[18] R. Banner, Y. Nahshan, D. Soudry, “Post training 4-bit quantization of convolutionalnetworks for rapid-deploymen”, **2019**.
[19] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, “DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients”, **2018**.
[20] Chien-Ping Lu in Proceedings of 2010 International Symposium on VLSI Design, Automation and Test, **2010**, pp. 5–5.
[21] Nvidia, “NVIDIA A100 Tensor Core GPU Architecture, Unprecedent acceleration at every scale”, **2020**.
[22] Nvidia, NVDLA Hardware Architectural Specification, `http://nvdla.org/hw/v1/hwarch.html`.
[23] Arm, “AMBA® AXI™ and ACE™ ProtocolSpecification”, **2011**.
[24] Nvidia, NVDLA Software Manual, `http://nvdla.org/sw/contents.html`.
[25] Google, An in-depth look at Google’s first Tensor Processing Unit (TPU), `https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu`.
[26] TensorFlow, `https://www.tensorflow.org/overview`.
[27] Xilinx, “Zynq-7000 SoC, Technical Reference Manual”, **2018**.
[28] Xilinx, PYNQ, `http://www.pynq.io/board`.
[29] G. Corradi, “The value of Python Productivity: exteme edge analytics on Xilinx zynq portfolio”, *Xilinx* **2018**.
[30] TensorFlow Hub, `https://www.tensorflow.org/hub/overview`.
[31] C Foreign Function Interface for Python, `https://cffi.readthedocs.io/en/latest/`.
[32] Xilinx, “Zynq-7000 SoC Data Sheet: Overview”, **2018**.
[33] Xilinx, “AXI Interconnect v2.1LogiCORE IP Product Guide”, **2017**.
[34] Xilinx, “Vivado Design Suite, AXI Reference Guide”, **2017**.
[35] Xilinx, “AXI DMA v7.1LogiCORE IP Product Guide”, **2019**.
[36] Xilinx, “7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter , User Guide”, **2018**.
[37] Xilinx, “AXI4-Stream Accelerator Adapter v2.1LogiCORE IP Product Guide”, **2015**.
[38] Xilinx, “7 Series DSP48E1 Slice,User Guide”, **2018**.

A

Appendix 1

add code for generating the shared library for the delegate