CHALMERS
UNIVERSITY OF TECHNOLOGY

POLITECNICO
DI TORINO

FPGA-based Tensor Accelerator for Machine Learning

Erasmus Master thesis in Computer science 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.

Abstract

Abstract text about your project in Computer Science and Engineering.

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, April 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.
The success of Machine Learning is being driven by the current available hardware which can provide the required demands in terms of storage and compute capacity. But obviously as problems scale so do the demands and thus special hardware is being developed to address these needs.

The use of commodity hardware is not the most effective and efficient way to address this problem, so research is looking at solution that can satisfy the required demands but at lower cost and energy consumption in order to support Machine Learning also on mobile devices. The constant developments in the Machine Learning methods require the design of more flexible hardware accelerators [1] [2].
As it is very well-known, the hardware accelerators are capable, if designed correctly, of delivery a lot of improvements in terms of the latency but also in terms of energy efficiency[3]. Thus, in order to obtain the best solution in every metric a hardware-software co-design is needed, requiring a basic knowledge of Machine Learning algorithms.

In the last years, the number of published papers regarding Machine Learning has growth exponentially, leading to a lot of efforts also from the companies which started to develop, deploy and sell their own hardware platforms, such as Tensor Processing Unit [4] from Google, NVDLA[5] from Nvidia and Gaudi [6] and Goya[7], respectively for training and interference, from Habana (acquired by Intel).

Machine Learning includes two processes, the training and the inference. Mainly, the work is focused on the inference process, exploring different optimizations in terms of hardware.

The aim of the work is to develop an flexible, especially in terms of arithmetic precision, hardware accelerator for Machine Learning applications, since it is a design from scratch the work is carried out on FPGA in order to have the possibility of exploring different solutions and measuring different design metrics of the prototype.

One of the main ideas of those works is that during the inference process, a model does not need the high precision computations [8] [9]for achieving high accuracy into its outputs. Thus, the use of different arithmetic data type can drastically reduce the computations without reducing the final accuracy of the DNN [10] [8]. From a hardware perspective, the use of different arithmetic precision[11] can lead to benefits in terms of area, energy consumption and latency. Moreover, the possibility of decide the current arithmetic precision of the accelerator allows the user, or the high level application, to use different Neural Network models leading to the use of different applications and/or domains of the same hardware, or even at the same time from the user perspective, actually the two models could be multiplexed in time on the accelerator.
According to that, accuracy of operations, reliability, performance and energy efficiency are evaluated, moving through different designs with different precision and/or data gating and compared to the implementation of the same DNN inside a GPU, as well as custom hardware platform.

better problem statement and contribution statement

2

Background

In this chapter a general glance on Artificial Intelligence, and its sub-categories, is given.

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 exponential growth of the last decade, 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) is intelligence demonstrated by machines, any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals.
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. 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, 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.

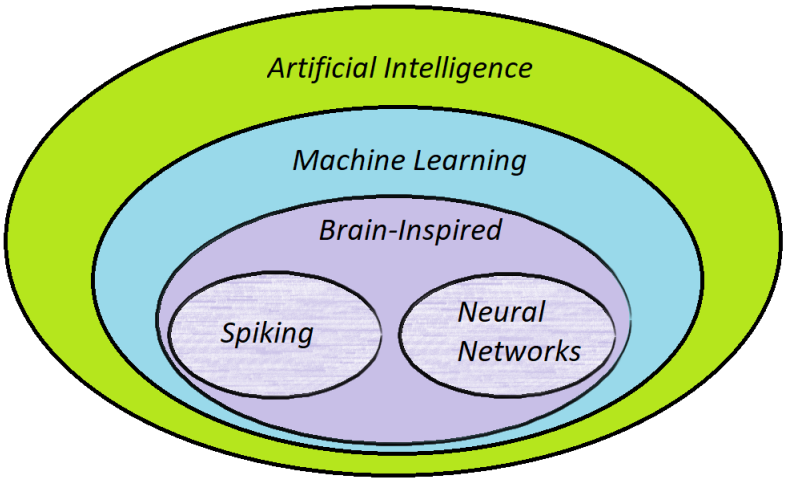

Figure 2.1: Classification of AI with emphasis on Machine Learning and its subclassification

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 suggest, 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 attempt 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, the aim is to identify rare items, events or observarions 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.

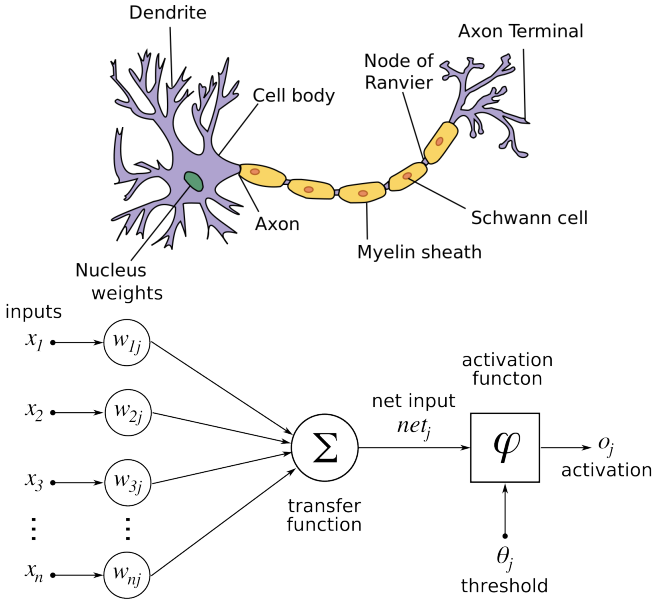

Figure 2.2: A parallelism between a human-brain neuron and a neuron in a Brain Inspired Network[14][15]

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.

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 b 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.

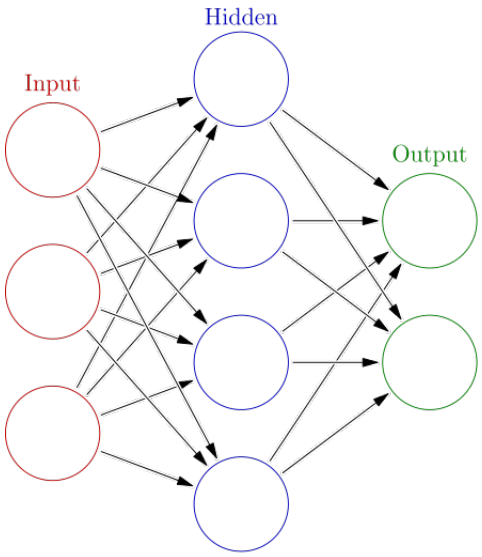

Figure 2.3: Example of a Neural Network[16]

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:

- Feedforward, 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[17].
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[18].

2.3 Machine Learning Quantization

The reduction of computation demand and the increase of power efficiency of Machine Learning algorithms can be achievied 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 to take 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 nowadas pictures, quantization is not lossless.

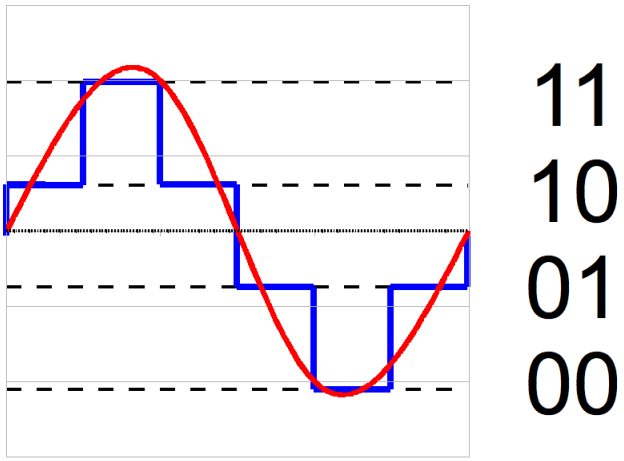

Figure 2.4: Approximation of analog signal to digital signal[19]

Going back to Machine Learning, 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[8]. On the other hand, this 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 consumptiom 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. 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, maintaing 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 C2, 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

In this chapter various solutions available on the market are presented for running the inference process of Machine Learning models.

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 Softwares, 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 Tesla V100

The Nvidia Tesla V100 is one of the most performant GPU at the moment on the market. The newly added Tensor Core Unit allows massive increases in throughput and efficiency.
It is able to deliver up to 125 TFPLOPS[1] for training and inference Machine Learning applications.

The core of the GPU is the Streaming Multiprocessor from the Nvidia Volta GPU.

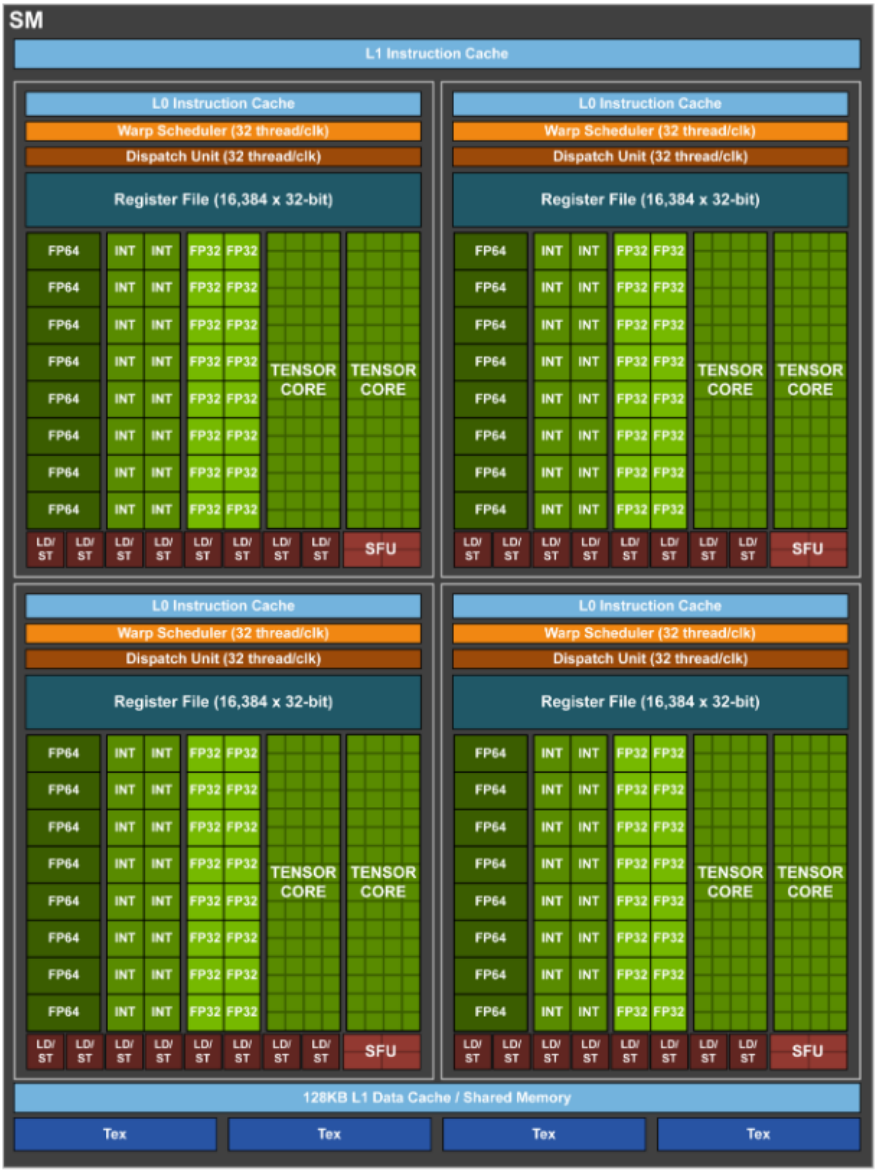

Figure 3.1: Streaming Multiprocessor Architecture [21]

Composed of integer, FP32, FP64 units and the Tensor Core Units designed specif-

[1] floating point operations per second

ically for deep learning. The Volta SM can achieve such efficient workload on mixed computation and addressing calculations thanks to an independent parallel integer and floating-point data paths.

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.

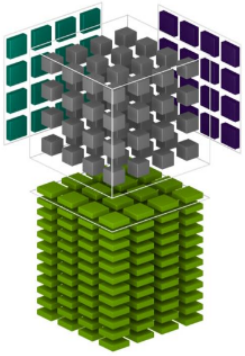

Figure 3.2: Matrix Multiplication in Tensor Core[21]

The Tesla V100 GPU contains 640 Tensor Cores: eight per Streaming Multiprocessor. Each one of the cores is able to perform 64 floating point Fused Multiply and Accumulate operations per clock. 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.

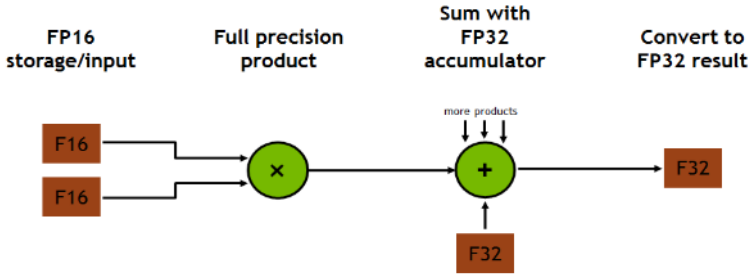

Figure 3.3: Mixed Precision Schema of a FMA unit in Tensor Core Unit[21]

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 3.4.

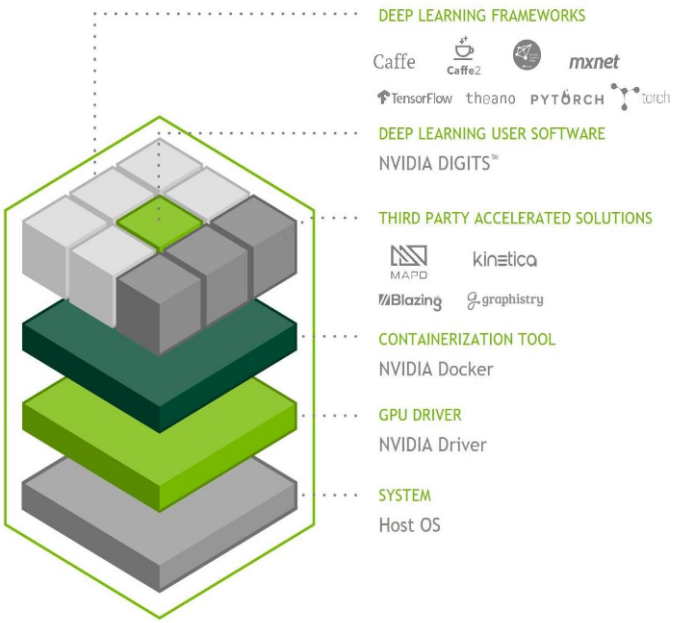

Figure 3.4: Software stack[21]

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:

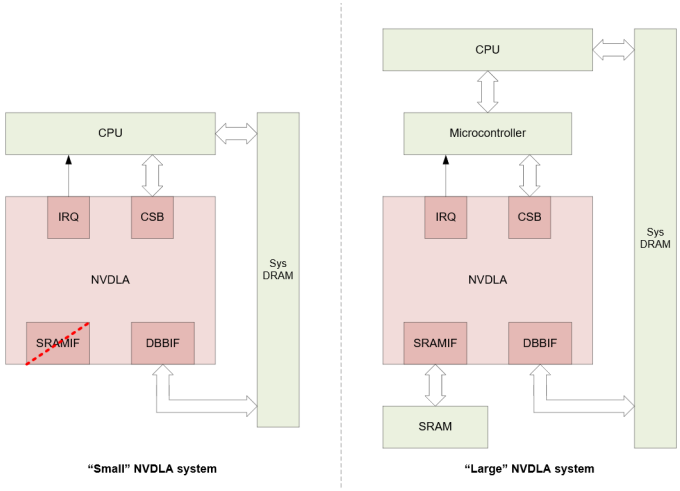

Figure 3.5: Comparsion of two possible NVDLA system[22]

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:

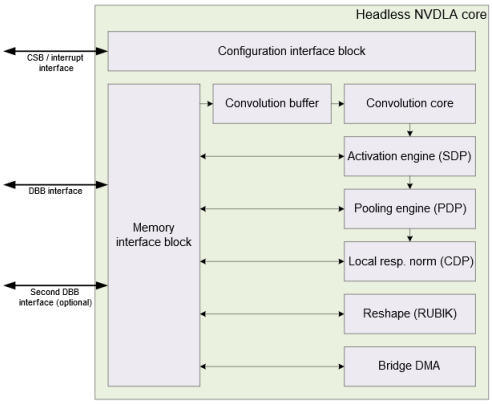

Figure 3.6: Internal architecture of NVDLA small system, Secondary DBB not considered[22]

According to Figure 3.5, 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 from 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:

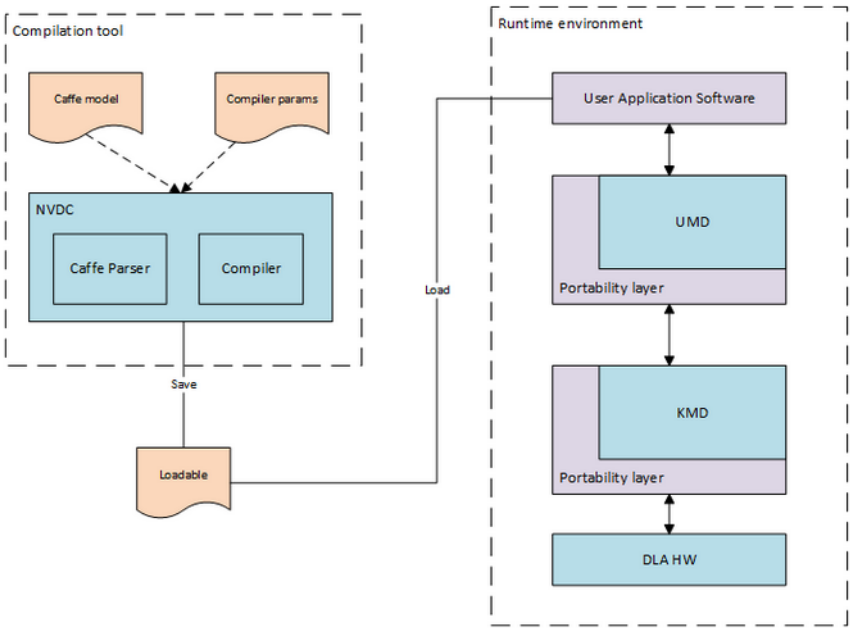

Figure 3.7: NVDLA Software stack[24]

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): 24MB of SRAM that work as registers
- Activation Unit (AU): Hardwired activation functions

In Figure 3.8 a general view of TPU architecture is presented.

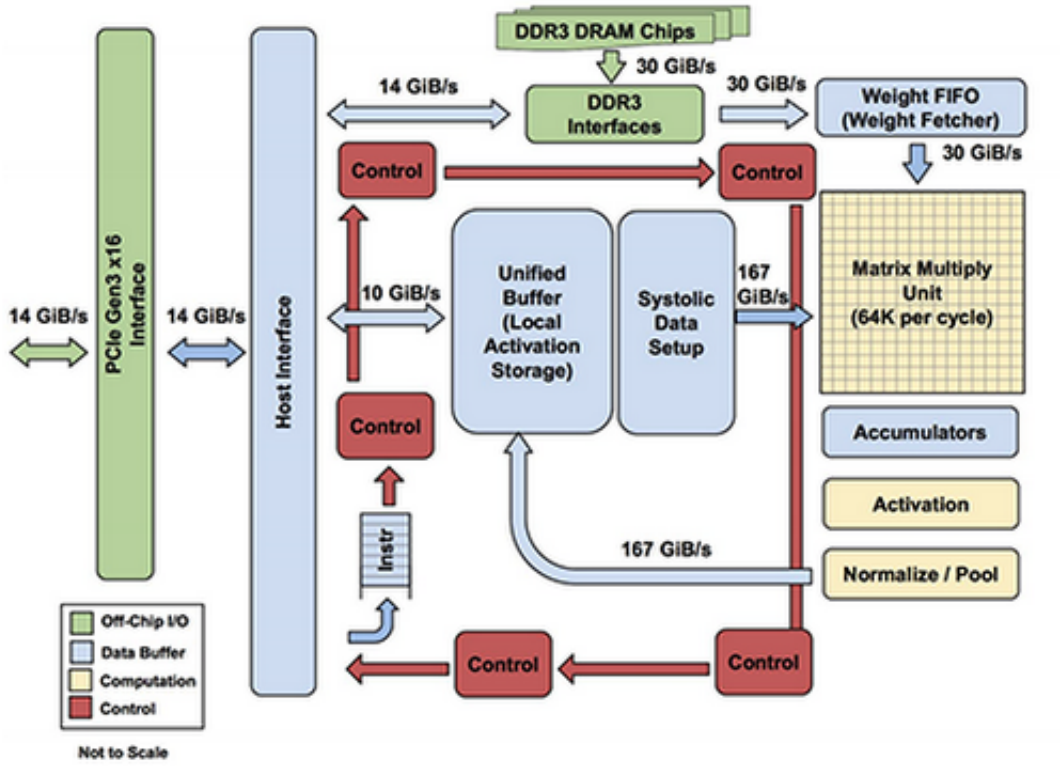

Figure 3.8: Google TPU architecture[4]

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.

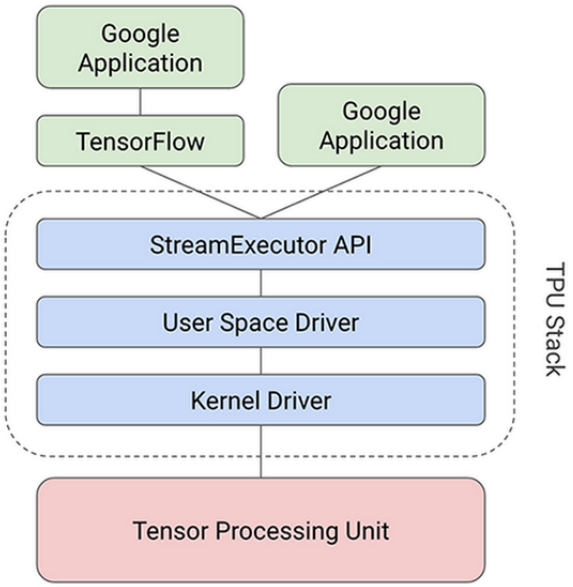

Figure 3.9: Google TPU Software Stack[25]

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 3.10 a high level view of the Goya architecture can be appreciated.

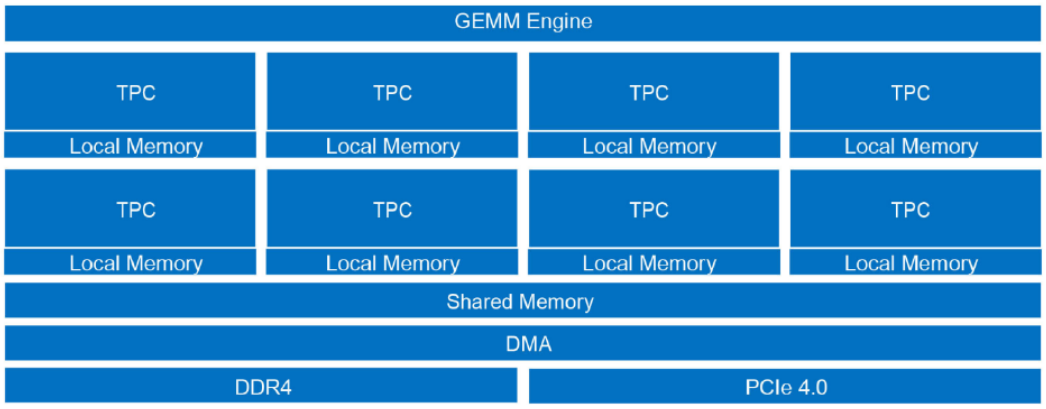

Figure 3.10: High level view of Goya architecture[7]

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 3.11.

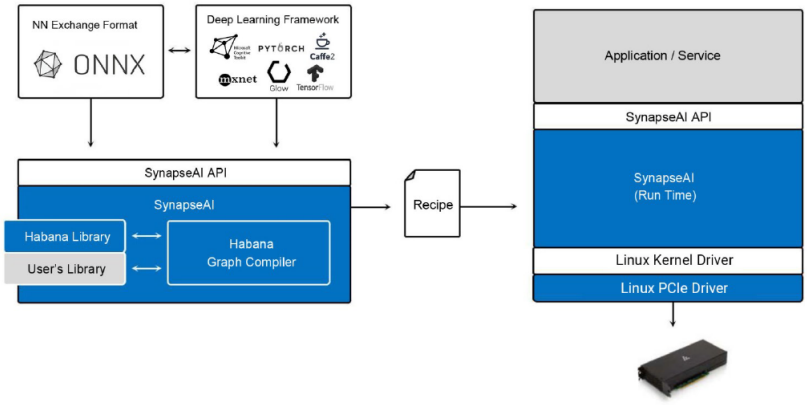

Figure 3.11: Habana Goya Software Stack[7]

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

4

System Development

In the following chapter a Top Down approach of the System development is described, with particular emphasis on the hardware development part.

4.1 Overview

it should start with a justification and overview of the proposed accelerator

The entire work is implemented on a PYNQ Z2 board from TUL, based on a Zynq-7000 SoC [26].

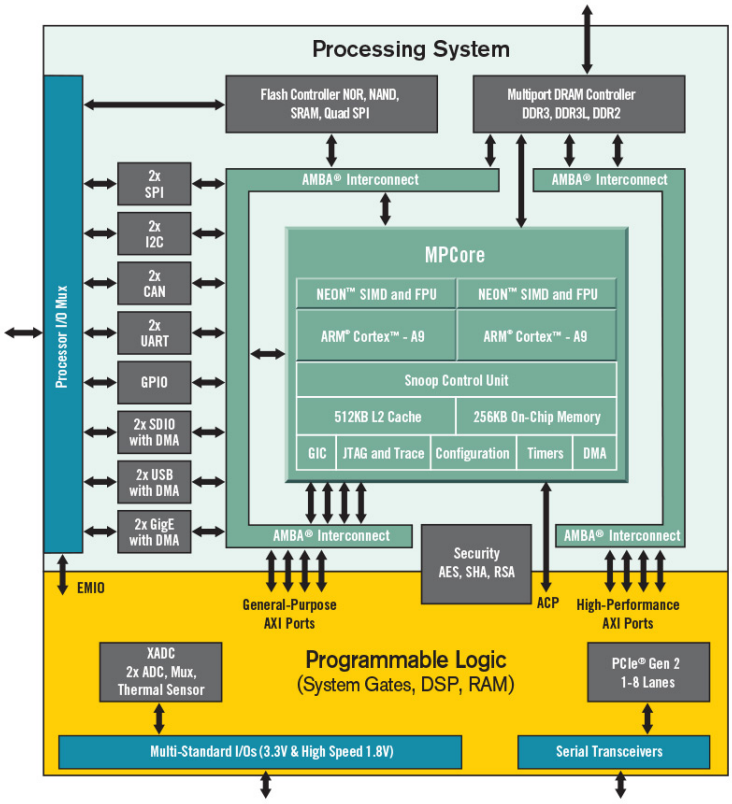

Figure 4.1: Zynq 7000 SoC[27]

The latter is composed of a programmable logic and a processing system, the Programmable logic hosts the accelerator and the relative AXI interconnections while the processing system is in charge of running the software for programming and scheduling correctly the operation on the accelerator.

In order to speed-up the development process and use built-in library for the AXI protocol and the DMA transfers, the software is carried out through the PYNQ enviroment of the board [28] which is based on Python, another reason is that Python has becoma a de facto standard [29].
The usage of Python as basic software allows to easily integrate it with high level Machine Learning API, such as TensorFlow in this case, an end-to-end open source Machine Learning platform [30]. It has the feature of quantize a post-trained model for different arithmetic precision.
Since the aim of the work is focused on the inference process, pre-trained models are needed and TensorFlow Hub [31] comes in handy for this purpose. It provides already pre-trained Machine Learning models for different domains.

4.2 Software

todo -> driver for accelerator and model from tensorflow (also offline quantization)

4.3 System Level

As it can be seen from 4.1 , it is divided in two big block:

- Processing System: The processing system in Figure 4.2 referred as *ps7* 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.
  Several single channel DMA have been used instead of using a single DMA with multiple channels, the reason is that the in the PYNQ enviroment only the drivers for the single channel DMA are provided. Furthermore, the Programmable Logic can be divided into:
  - AXI interconnections: IP cores from Xilinx[32][33] in order to connect and correctly address entities in the Programmable Logic.
  - AXI DMA: IP core from Xilinx [34] which allows data movement between main memory and accelerator memories.
  - DTPU: the actual hardware accelerator.
  - XADC: IP core from Xilinx [35] which allows to measure the temperature of the SoC, the voltages and the currents at runtime.

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

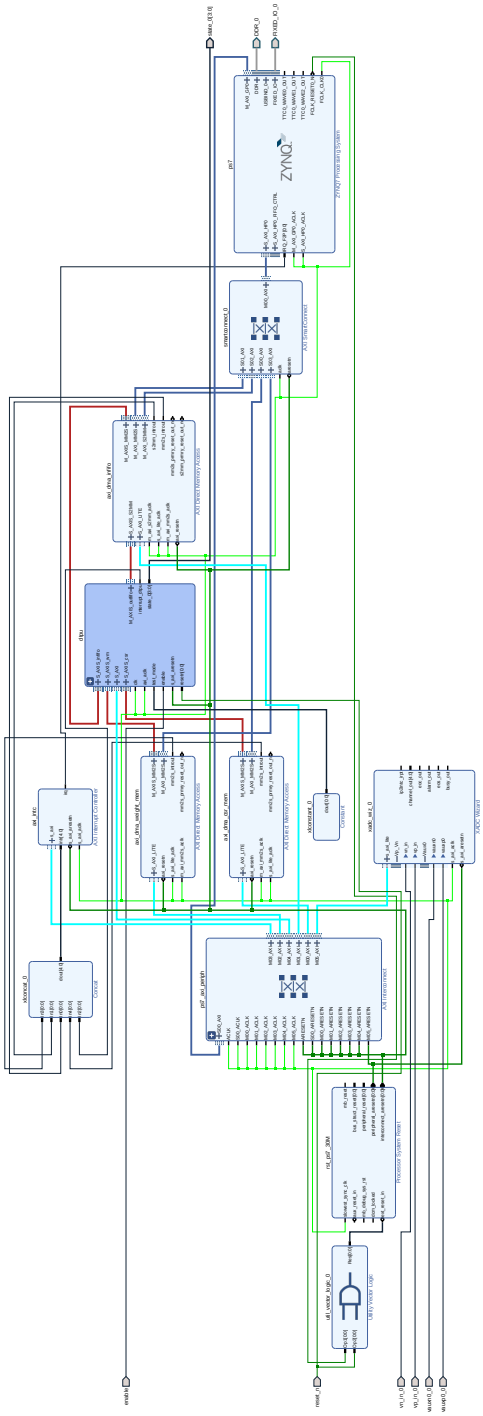

Figure 4.2: System

4.4 DTPU, the hardware accelerator

The hardware accelerator, named *Dumb Tensor Processing Unit*, is in charge of carring out the computation of the neural network model, exploiting a data-flow architecture on the input data applyng a Fused Multiply Add approach between each unit of the Matrix Multiplication Unit (MXU).

In Figure 4.3 is presented the Logical block diagram of the accelerator.

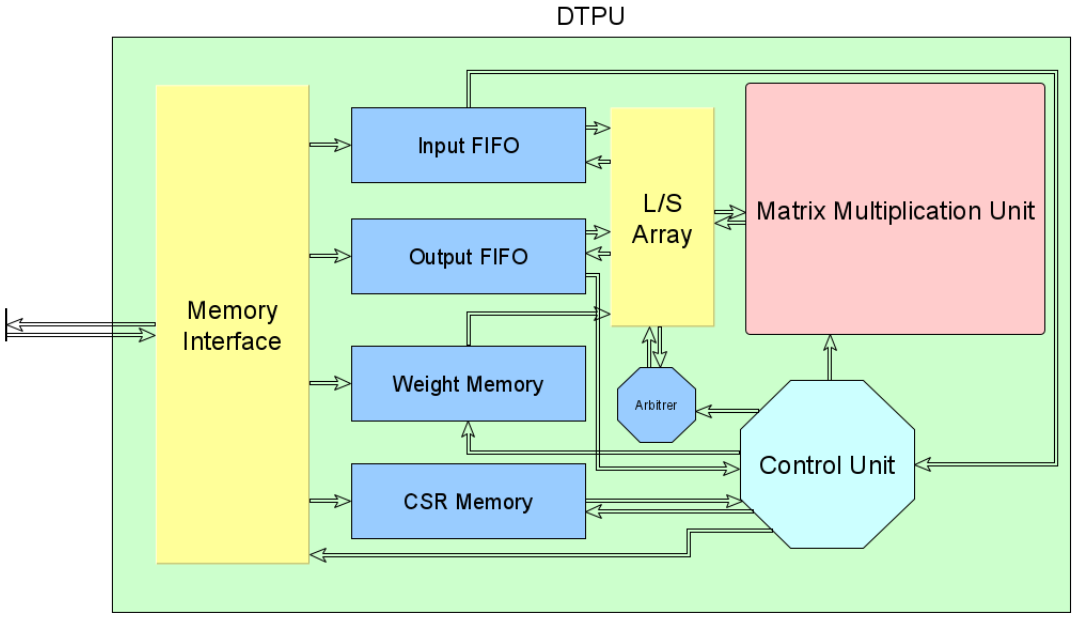

Figure 4.3: Logical view of the accelerator

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 the necessary subcomponents, has been used[36] leading to the actual block diagram which can be observed in the Figure 4.4.

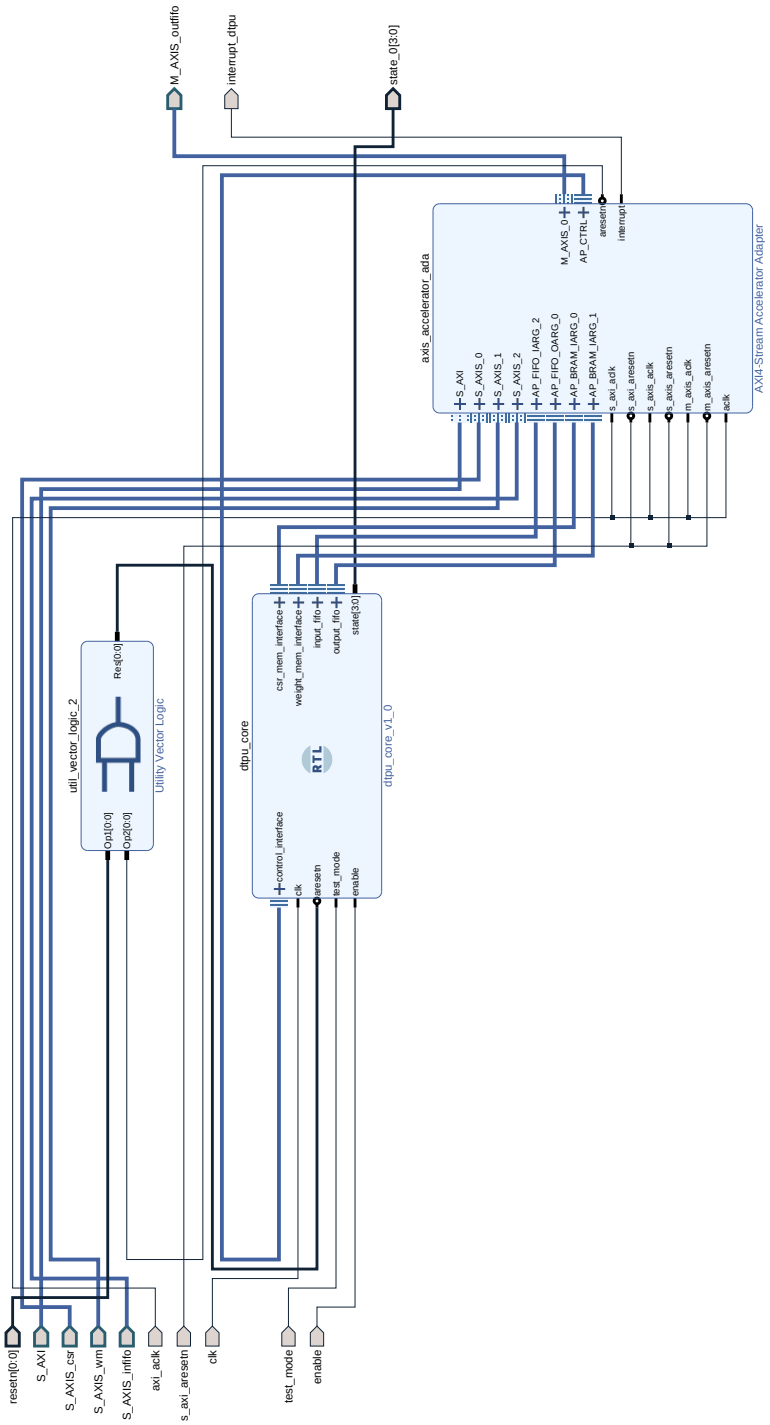

Figure 4.4: Real RTL view of the accelerator

The latter has allowed to completely focus the work on the DTPU core.

4.4.2 Axi accelerator adapter

4.4.3 DTPU core

4.4.3.1 Matrix Multiplication Unit

5
Results

In this chapter, area results on the FPGA of different designs are presented. Following that, on different hardware designs Neural Network TensorFlow Lite models are executed and perfomances measured.

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 [1] 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, for a mobile device in which there is a limited battery capacity meanwhile for data centers stringent power ceilings due to cooling costs.
- 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.

Post Synthesis FPGA Utilization Computation on 4 bit integer			
Size of Matrix Multiplication Unit	LUT	FF	DSP
3x3 4x4 5x5 6x6 7x7 8x8 10x10 12x12 15x15 16x16 20x20			

Post Synthesis FPGA Utilization Computation on 8 bit integer			
Size of Matrix Multiplication Unit	LUT	FF	DSP
3x3 4x4 5x5 6x6 7x7 8x8 10x10 12x12 15x15 16x16 20x20			

6
Conclusion

You may consider to instead divide this chapter into discussion of the results and a summary.

6.1 Discussion

6.2 Conclusion

6.3 Future Works

Bibliography

[1] 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**.

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

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

[4] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, R. C. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit", *CoRR* **2017**, *abs/1704.04760*.

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

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

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

[8] 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.

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

[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] Wikipedia, 'Neuron, `https://simple.wikipedia.org/wiki/Neuron`.

[15] Wikipedia, 'Rosenblatt's Perceptron Machine Learning Neuron, https://commons.wikimedia.org/wiki/File:Rosenblattperceptron.png.
[16] Wikipedia, Artificial Neural Networks, https://en.wikipedia.org/wiki/Artificial_neural_network.
[17] W. Maass, "Networks of spiking neurons: The third generation of neural network models", *Neural Networks* **1997**, *10*, 1659 –1671.
[18] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, A. Maida, "Deep learning in spiking neural networks", *Neural Networks* **2019**, *111*, 47 –63.
[19] Wikipedia, Quantization (Signal Processing), https://en.wikipedia.org/wiki/Quantization_(signal_processing)#cite_note-3.
[20] Chien-Ping Lu in Proceedings of 2010 International Symposium on VLSI Design, Automation and Test, **2010**, pp. 5–5.
[21] Nvidia, "NVIDIA Tesla V100 GPU architecture, The world's most advanced data center GPU", **2017**.
[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] Xilinx, "Zynq-7000 SoC, Technical Reference Manual", **2018**.
[27] Xilinx, "Zynq-7000 SoC Data Sheet: Overview", **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, https://www.tensorflow.org/overview.
[31] TensorFlow Hub, https://www.tensorflow.org/hub/overview.
[32] Xilinx, "AXI Interconnect v2.1LogiCORE IP Product Guide", **2017**.
[33] Xilinx, "Vivado Design Suite, AXI Reference Guide", **2017**.
[34] Xilinx, "AXI DMA v7.1LogiCORE IP Product Guide", **2019**.
[35] Xilinx, "7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter , User Guide", **2018**.
[36] Xilinx, "AXI4-Stream Accelerator Adapter v2.1LogiCORE IP Product Guide", **2015**.

recheck references

add the xilinx ip core manuals

A

Appendix 1