

# FPGA-based Tensor Accelerator for Machine Learning

Erasmus Master thesis in Computer science and engineering

Francesco Angione



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

It is always hard to write this part of a work. I would say it is the hardest part, more than the technical one.

However, let me try to address it anyway. I am apologizing in advance if i will forgot something.

This work is the sum of five years of experiences, from a technical and non technical point of view, and it has been developed during a terrible event, a pandemic, which has literally stopped the entire world and caused death, issues and debates. However, as the human race has always been, we are resilient to everything and we tried as much as we could to not let the world stop, especially thanks to technology, Internet and all the related services. We are just human, but we can do whatever we can image, especially in Computer Science.

First of all, I would like to thank my family for all their support and presence, even when i was going counter current in my life. I would like to thank to both my supervisors Prof. Paolo Bernardi and Prof. Pedro Petersen Moura Trancoso for believing in me without any guarantees on the final work, and their support through this journey.

*To the day in which I learnt how to read, an important pillar of my life.  
To the people who have contribuited, in badness and goodness, to make me the person who i am today.  
To my past and future failures, where i have built and i will build myself.  
To my feelings, which remember us how much we are fragile but at the same time they remind us that we are human being and we gather our strength from them.  
Sapere aude.*

Francesco Angione, Gothenburg, August 2020



# Contents

<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>3</b>
2.1 Overview . . . . .	3
2.2 Machine Learning . . . . .	4
2.2.1 Brain Inspired . . . . .	5
2.2.1.1 Neural Networks . . . . .	6
2.2.1.2 Spiking Neural Networks . . . . .	7
2.3 Machine Learning Quantization . . . . .	8
2.4 Applications . . . . .	9
<b>3 State-of-the-Art</b>	<b>11</b>
3.1 Overview . . . . .	11
3.2 GPU . . . . .	11
3.2.1 Nvidia Ampere A100 Tensor Core GPU . . . . .	12
3.3 Domain Specific Hardware Platform . . . . .	16
3.3.1 NVDLA . . . . .	16
3.3.1.1 NVDLA Software . . . . .	18
3.3.2 Google TPU . . . . .	19
3.3.3 Habana Goya HL-1000 . . . . .	20
<b>4 System Development</b>	<b>23</b>
4.1 Overview . . . . .	23
4.2 Software . . . . .	27
4.3 System Level . . . . .	29
4.4 DTPU, the hardware accelerator . . . . .	31
4.4.1 Real Implementation . . . . .	31
4.4.2 High Level State Machine of Control Unit . . . . .	33
4.4.3 Datapath . . . . .	34
4.4.3.1 Filter&Select and Compact&Select . . . . .	35
4.4.3.2 Matrix Multiplication Unit . . . . .	38
<b>5 Results</b>	<b>41</b>

5.1	Evaluation metrics . . . . .	41
5.2	Utilization Factor . . . . .	42
5.3	Energy and Power Consumption . . . . .	45
5.4	Throughput . . . . .	63
5.5	Latency . . . . .	64
5.6	Accuracy . . . . .	67
<b>6</b>	<b>Conclusion</b>	<b>69</b>
6.1	Discussion . . . . .	69
6.2	Future Works . . . . .	70
	<b>Bibliography</b>	<b>71</b>
	<b>A Accelerator library</b>	<b>I</b>
	<b>B Top level entity of DTPU core</b>	<b>XLVII</b>

# List of Figures

2.1	Classification of AI with emphasis on Machine Learning and its sub-classification . . . . .	4
2.2	A parallelism between a human-brain neuron and a neuron in a Brain Inspired Network . . . . .	5
2.3	Example of a Neural Network . . . . .	6
2.4	Approximation of floating-point values to integer values . . . . .	8
3.1	Streaming Multiprocessor Architecture [21] . . . . .	12
3.2	Matrix Multiplication in Tensor Core [21] . . . . .	13
3.3	Mixed Precision Schema of a FMA unit in Tensor Core Unit [21] . . . . .	13
3.4	Sparsity Optimization of a weight tensor [21] . . . . .	14
3.5	Matrix Multiply Accumulate [21] . . . . .	14
3.6	Software stack [21] . . . . .	15
3.7	Comparsion of two possible NVDLA system [22] . . . . .	16
3.8	Internal architecture of NVDLA small system, Secondary DBB not considered [22] . . . . .	16
3.9	NVDLA Software stack[24] . . . . .	18
3.10	Google TPU architecture[1] . . . . .	19
3.11	Google TPU Software Stack [25] . . . . .	20
3.12	High level view of Goya architecture [4] . . . . .	20
3.13	Habana Goya Software Stack [4] . . . . .	21
4.1	Average execution time divided by type of operations . . . . .	23
4.2	Development workflow . . . . .	25
4.3	Execution Graph . . . . .	27
4.4	Flow Chart with accelerator . . . . .	28
4.5	Zynq 7000 SoC [32] . . . . .	29
4.6	System view hosted in the PL <sup>1</sup> . . . . .	30
4.7	Logical view of DTPU accelerator . . . . .	31
4.8	Real RTL view of DTPU accelerator . . . . .	31
4.9	RTL view of DTPU core . . . . .	32
4.10	A high level view of Control Unit . . . . .	33
4.11	A detailed view of the DTPU core datapath. Enable and resets signals for clocked units has been omitted for improving readability. . . . .	34
4.12	Data Distribution of Filter&Select unit for a MXU size of 8x8 . . . . .	36
4.13	MXU interal structure and weights distribution . . . . .	38
4.14	SMAC and SMUL details . . . . .	39

4.15 DSP Slice Functionality [38] . . . . .	40
5.1 Post Implementation Utilization Factor of integer 8 bit PEs and clock frequency of 30 Mhz . . . . .	42
5.2 Post Implementation Utilization Factor of integer 16 bit PEs and clock frequency of 30 Mhz . . . . .	42
5.3 Post Implementation Utilization Factor of integer 32 bit PEs and clock frequency of 30 Mhz . . . . .	43
5.4 Post Implementation Utilization Factor of integer 64 bit PEs and clock frequency of 30 Mhz . . . . .	43
5.5 Post Implementation Utilization Factor of bfp 16 bit PEs and clock frequency of 30 Mhz . . . . .	43
5.6 Post Implementation Utilization Factor of fp 32 bit PEs and clock frequency of 30 Mhz . . . . .	44
5.7 Post Implementation Power Consumption of Processing System for integer 8 PEs . . . . .	45
5.8 Post Implementation Static Power Consumption Programmable logic for integer 8 PEs . . . . .	45
5.9 Post Implementation Dynamic Power Consumption per Programmable logic with integer 8 PEs . . . . .	46
5.10 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 8 PEs . . . . .	47
5.11 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 8 PEs . . . . .	47
5.12 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 80 MHz and integer 8 PEs . . . . .	48
5.13 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 100 MHz and integer 8 PEs . . . . .	48
5.14 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 120 MHz and integer 8 PEs . . . . .	49
5.15 Post Implementation Power Consumption of Processing System for integer 16 PEs . . . . .	49
5.16 Post Implementation Static Power Consumption Programmable logic for integer 16 PEs . . . . .	50
5.17 Post Implementation Dynamic Power Consumption per Programmable logic with integer 16 PEs . . . . .	50
5.18 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 16 PEs . . . . .	51

5.19 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 16 PEs . . . . .	51
5.20 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 80 MHz and integer 16 PEs . . . . .	52
5.21 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 100 MHz and integer 16 PEs . . . . .	52
5.22 Post Implementation Power Consumption of Processing System for integer 32 PEs . . . . .	53
5.23 Post Implementation Static Power Consumption Programmable logic for integer 32 PEs . . . . .	53
5.24 Post Implementation Dynamic Power Consumption per Programmable logic with integer 32 PEs . . . . .	54
5.25 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 32 PEs . . . . .	54
5.26 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 32 PEs . . . . .	55
5.27 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 80 MHz and integer 32 PEs . . . . .	55
5.28 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 100 MHz and integer 32 PEs . . . . .	56
5.29 Post Implementation Power Consumption of Processing System for integer 64 PEs . . . . .	56
5.30 Post Implementation Static Power Consumption Programmable logic for integer 64 PEs . . . . .	57
5.31 Post Implementation Dynamic Power Consumption per Programmable logic with integer 64 PEs . . . . .	57
5.32 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 64 PEs . . . . .	58
5.33 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 64 PEs . . . . .	58
5.34 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 60 MHz and integer 64 PEs . . . . .	59
5.35 Post Implementation Power Consumption for bfp16 PEs . . . . .	60
5.36 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and bfp16 PEs . . . . .	60
5.37 Post Implementation Power Consumption for fp32 PEs . . . . .	61

5.38 Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and fp32 PEs	61
5.39 Comparison of Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and a MXU 3x3 . . . . .	62
5.40 Roofline model of the accelerator with a MXU size of 8x8 . . . . .	63
5.41 Roofline model of the accelerator with a MXU size of 8x8 and vectorized PEs . . . . .	63
5.42 Total Execution time of Invoke method (left) in the configuration with accelerator and MNIST model . . . . .	65
5.43 Total Execution time of Invoke method (left) in the configuration with accelerator and Cifar10 model . . . . .	65

# List of Tables

5.1	Execution Time for different platform and model, integer 8 . . . . .	64
5.2	Execution Time for different platform and model, integer 16 . . . . .	64
5.3	Execution Time for different platform and model, integer 32 . . . . .	64
5.4	Accuracy Output <sup>1</sup> with Convolution on integer 8 . . . . .	67
5.5	Accuracy Output <sup>1</sup> with Convolution on integer 16 . . . . .	67
5.6	Accuracy Output <sup>1</sup> with Convolution on integer 32 . . . . .	67



# 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 inference, 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.

## 1. Introduction

---

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.

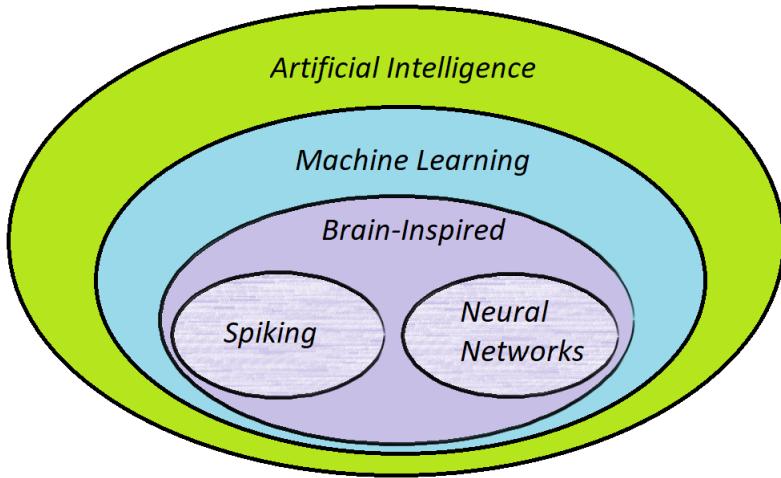
It takes one of its roots from a theoretical paper of *Alan Turing* published by journal *Mind* in the 1950 [12].

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:

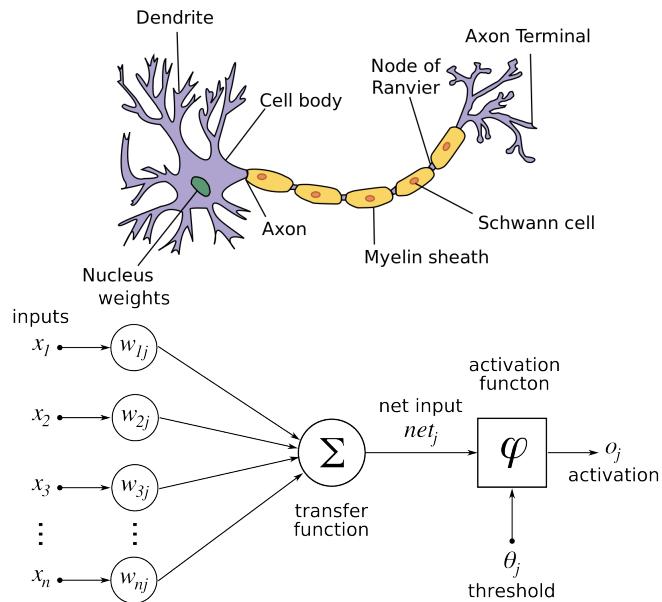
- Supervised Learning, it builds a mathematical model of a set of data that contains both the inputs and the desired outputs.
- Unsupervised Learning, it takes a set of data that contains only inputs and find structure in the data.
- Semi-supervised Learning, it falls between unsupervised learning and 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<sup>1</sup>

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.

<sup>1</sup>Figures under CC license

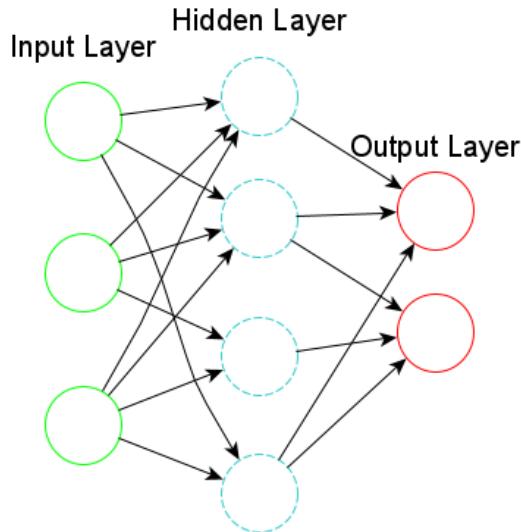
### 2.2.1.1 Neural Networks

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.

SNNs 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].

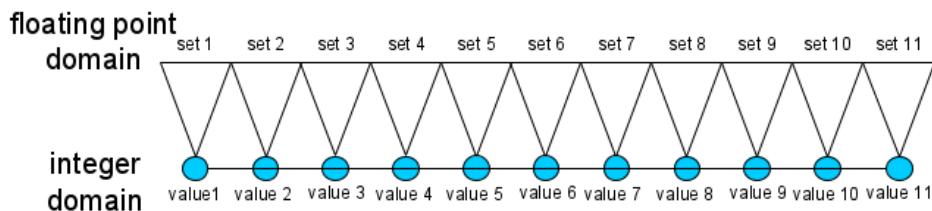
## 2.3 Machine Learning Quantization

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.

## 2. Background

# 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 have 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 overheads, especially in terms memory accesses which are expensive in terms of energy and latency.

Analyzing machine learning algorithms 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 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, improving efforts especially from the companies have been made for developing more and more GPUs with a higher 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 announced 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 TFLOPS<sup>1</sup> 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.



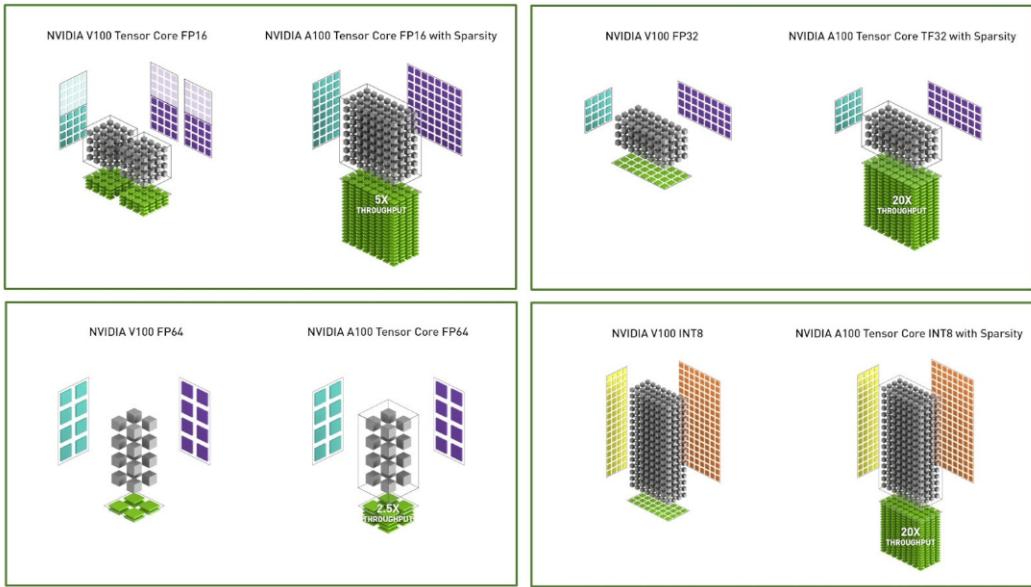
**Figure 3.1:** Streaming Multiprocessor Architecture [21]

Composed of integer, FP32, FP64 units and the Tensor Core Units are designed specifically for deep learning. It introduces also new data types in the tensor core for the computation such as 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.

---

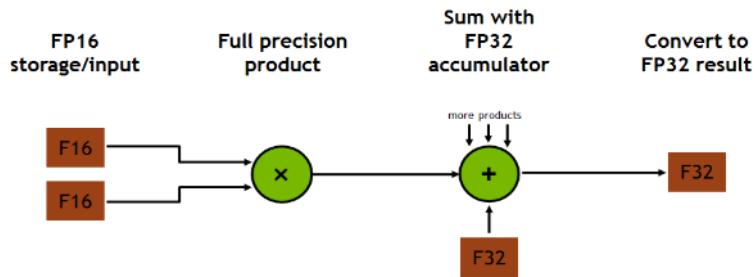
<sup>1</sup>floating-point operations per second

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.

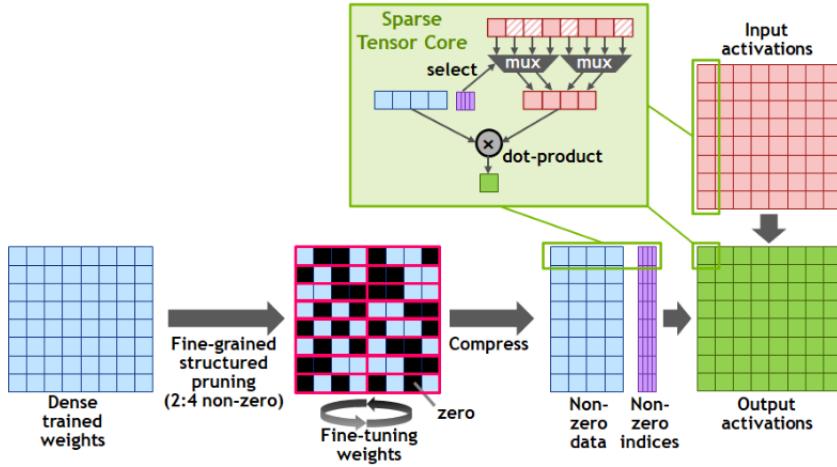


**Figure 3.3:** Mixed Precision Schema of a FMA unit in Tensor Core Unit [21]

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

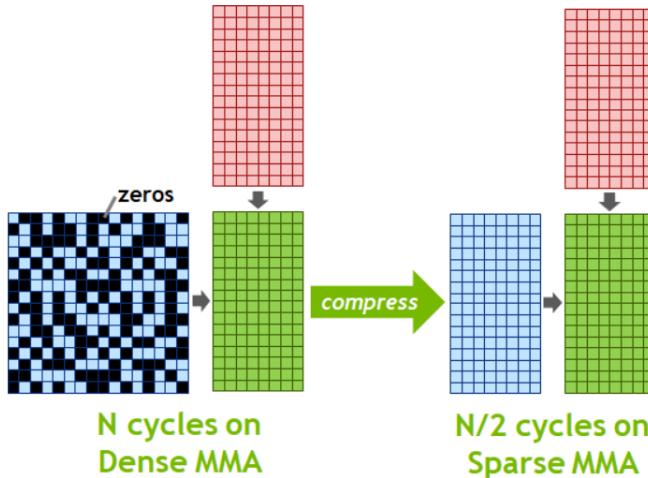
the training process can be accelerated exploiting the sparsity idea but it has to be introduced at the beginning of the process for achieving some benefits.



**Figure 3.4:** Sparsity Optimization of a weight tensor [21]

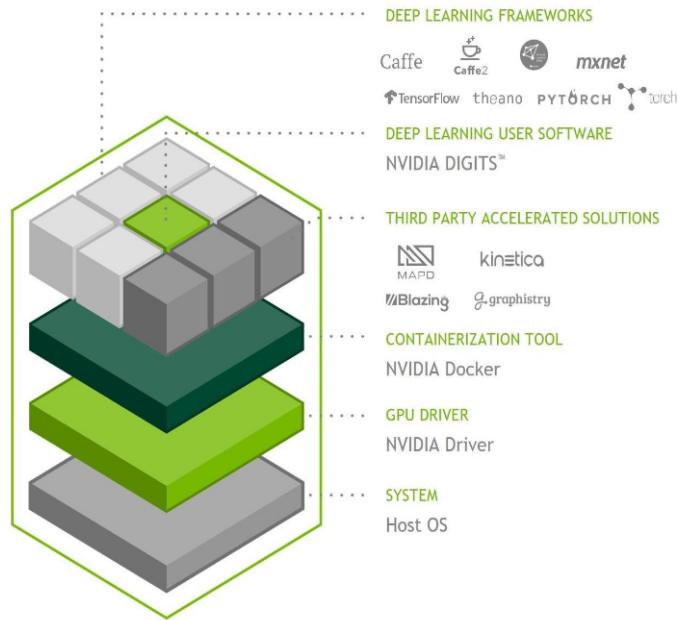
The approach in Figure 3.4 doubles the throughput by skipping the zeros. It also leads to a reduction of memory footprint and an increase into the memory bandwidth.

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 Figure 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.

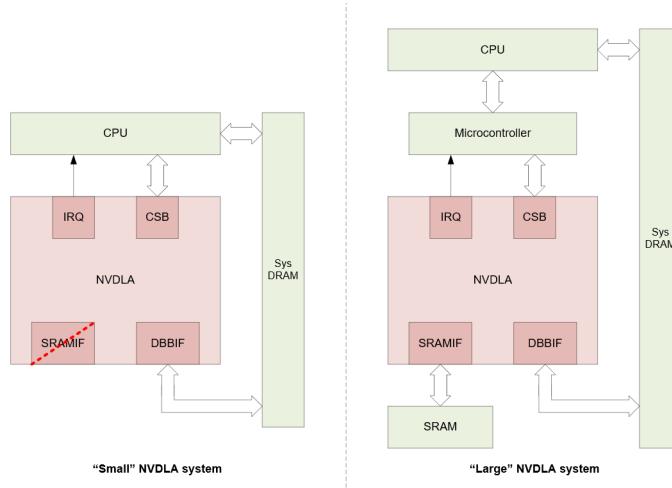
### 3.3 Domain Specific Hardware Platform

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.

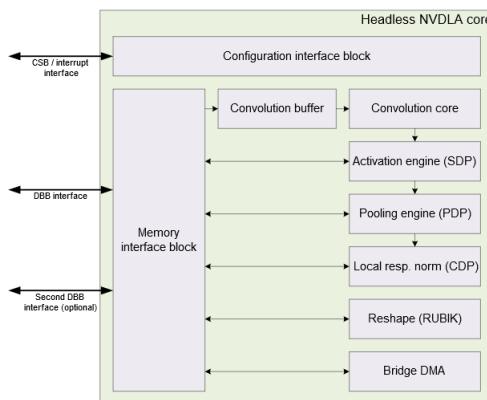
The architecture comes in two configurations:



**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 through 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 operations:

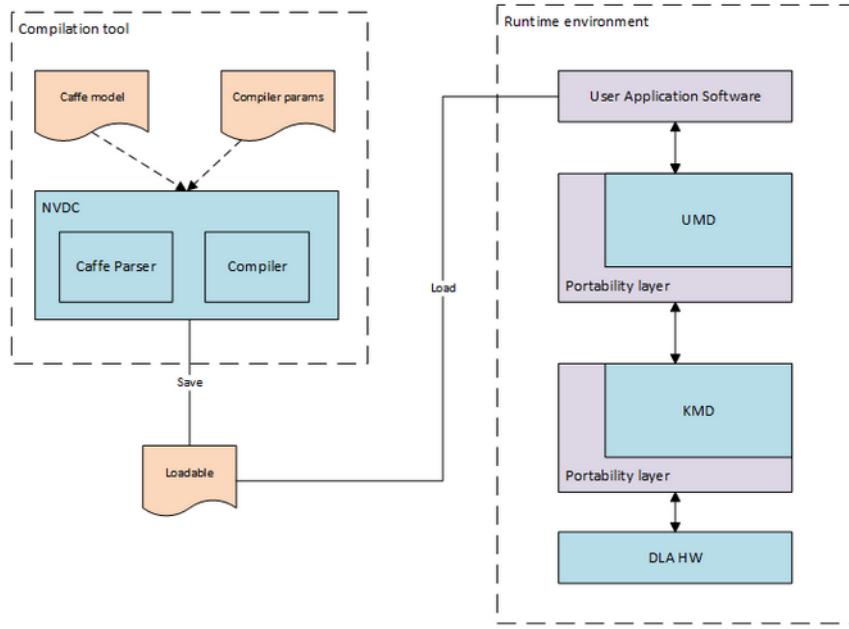
- Convolution Core: it comes in pair with the Convolution Buffer, its private memory for the data (inputs and weights). It is used to accelerate the convolution algorithms.
- Activation engine (Single Data point Operations): it performs post processing operations at the single data element level such 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]

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 submitting operations to the accelerator through low level functions, scheduling the operations and handling the interrupts.

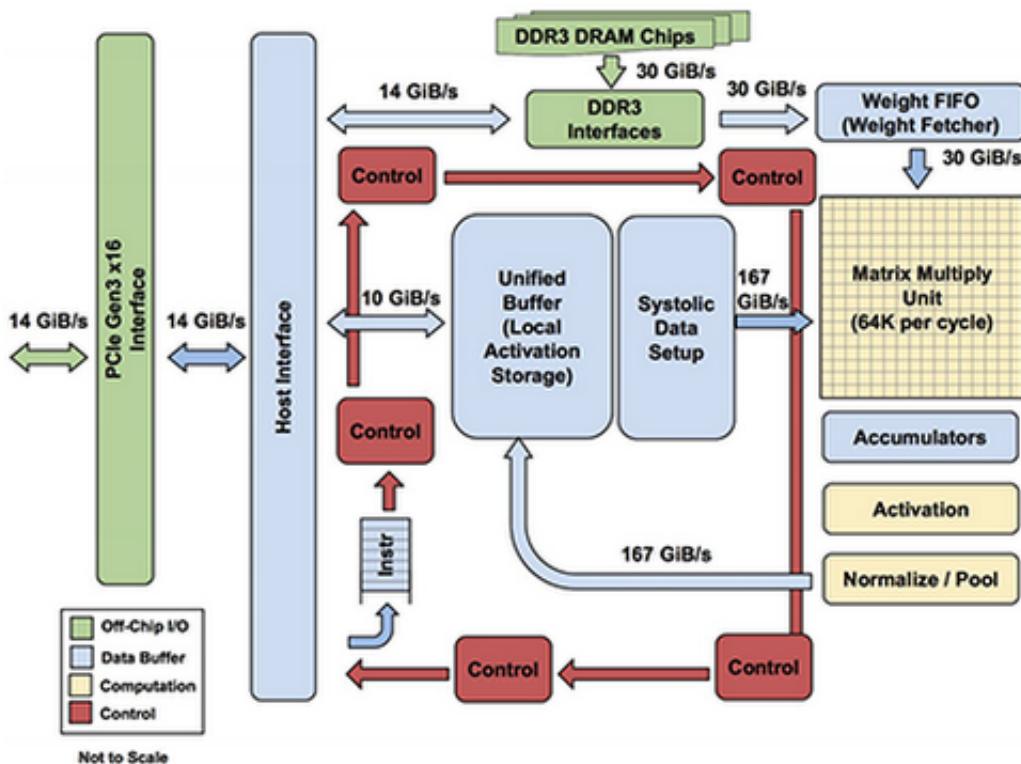
Both the KMD and the UMD are wrapped into portability layers which are, respectively, hardware dependent and OS dependent. In principle, for migrating the software to another OS or hardware platform it is enough to modify only the portability layers.

### 3.3.2 Google TPU

Google developed its own application-specific integrated 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.

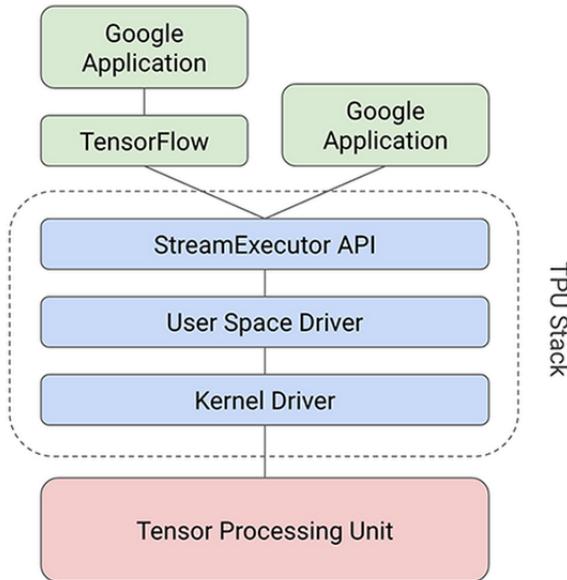


**Figure 3.10:** Google TPU architecture[1]

Rather than be tightly integrated with a CPU, the TPU is designed to be a coprocessor in which the instructions 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 concerned 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.



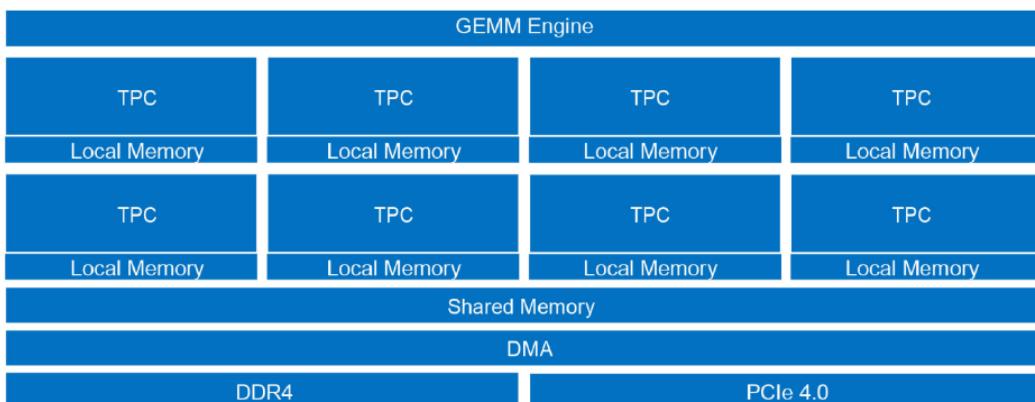
**Figure 3.11:** Google TPU Software Stack [25]

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

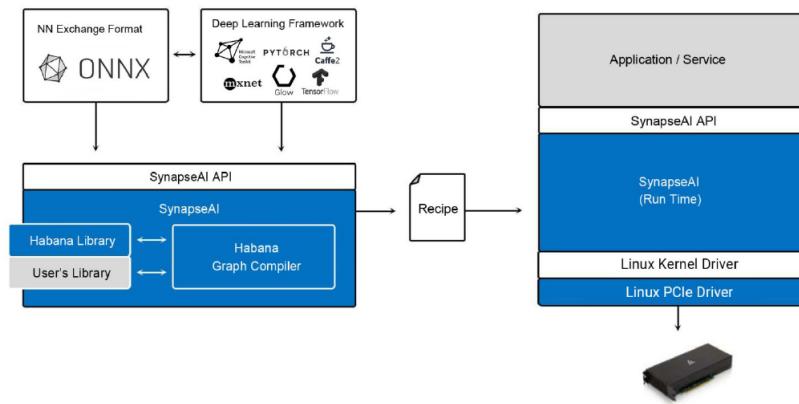


**Figure 3.12:** High level view of Goya architecture [4]

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.

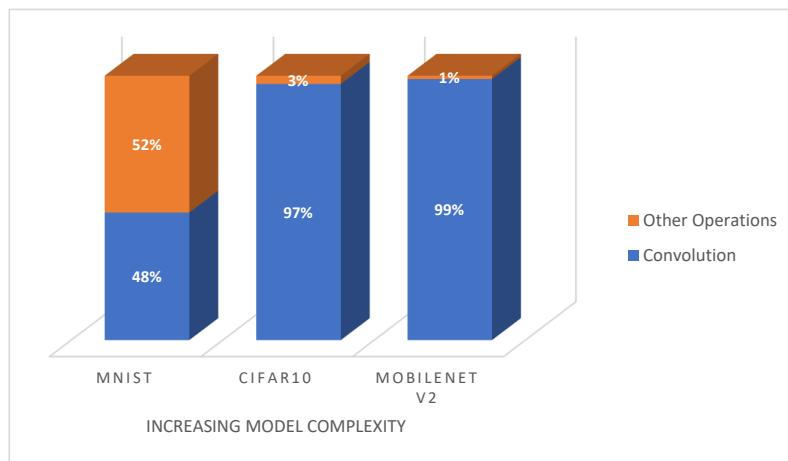


# 4

## 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 that those data, in particular weight data, are reused for several executions 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, as it can be also observed in the following Figure:



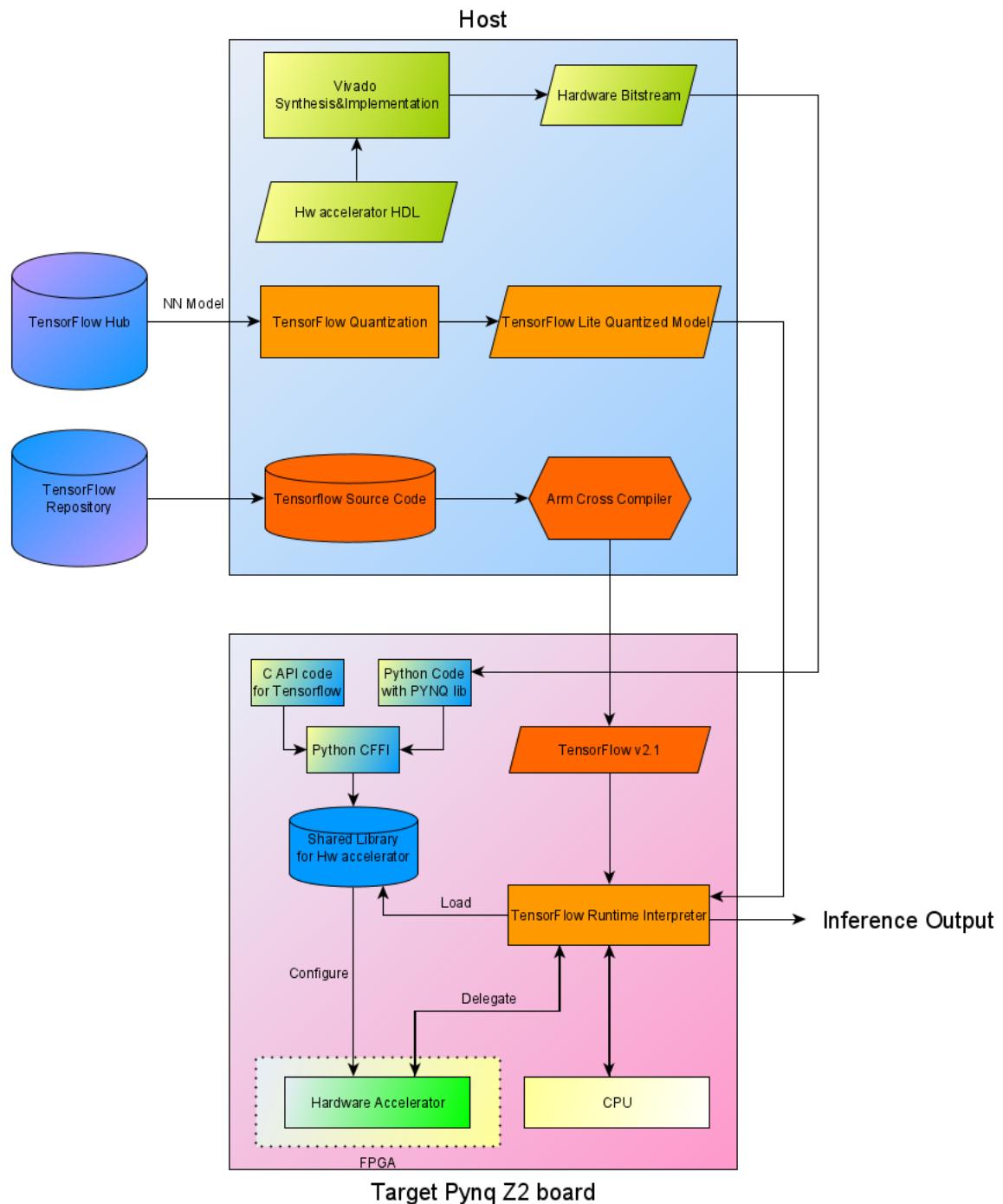
**Figure 4.1:** Average execution time divided by type of operations

Following the fast demands of operations into a Neural Network, it becomes evident

that executing them on a CPU could not meet 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 4.2:** Development workflow

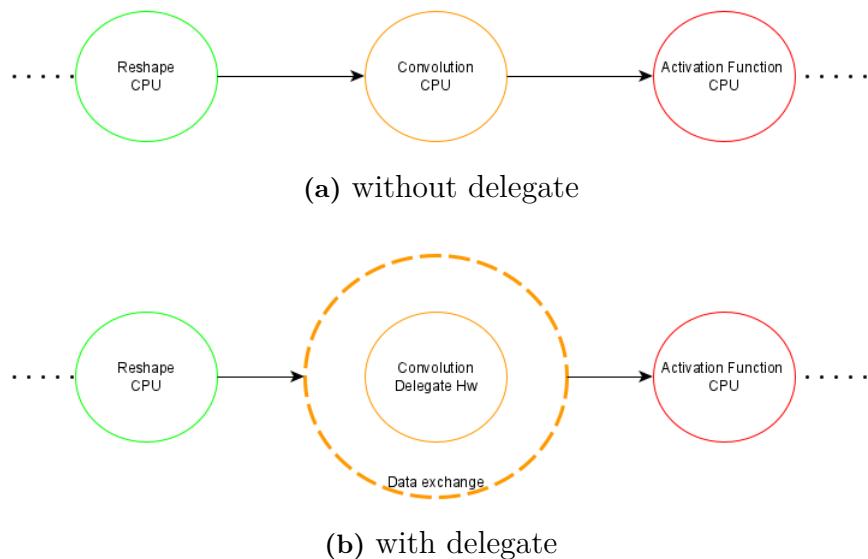
The entire work is implemented on a PYNQ Z2 board from TUL, based on a Zynq-7000 SoC [27]. In order to speed-up the development process and use built-in library for the AXI protocol and the DMA transfers, the software is partially carried out through the PYNQ environment of the board [28] based on Python which has became 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 quantizing a post-trained model for different arithmetic precision. In the Fig. 4.2 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 as it is represented in the following Figures: It is worth to mention that TensorFlow is open-source and since



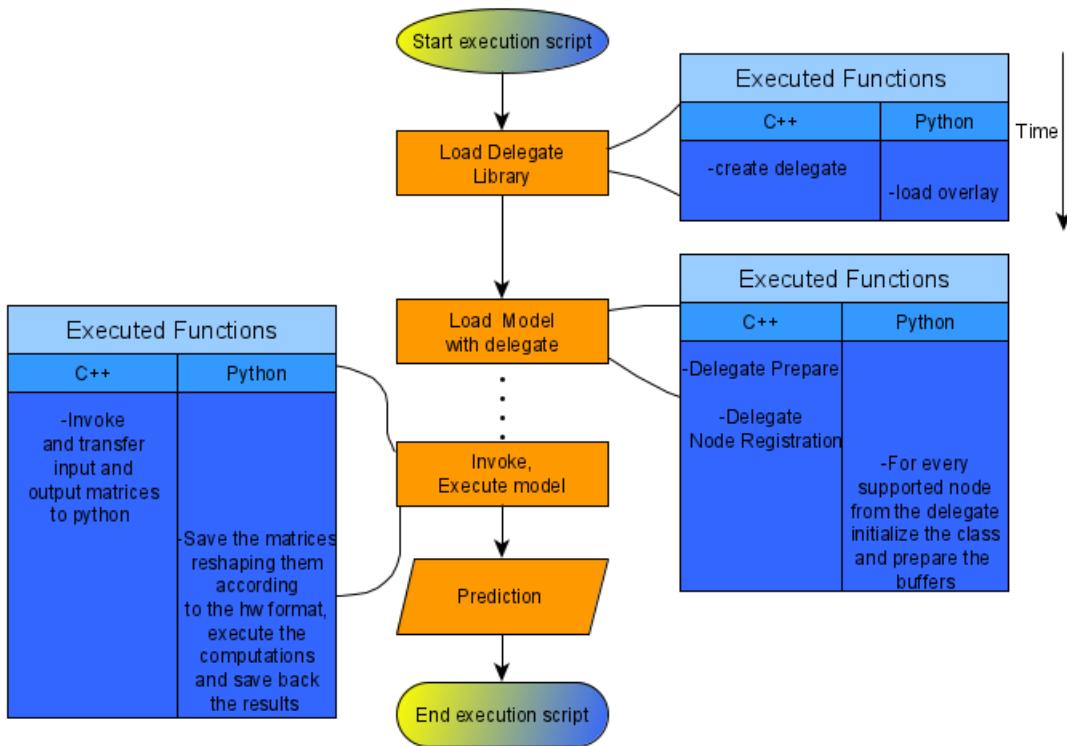
**Figure 4.3:** Execution Graph

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.

## 4. System Development

---

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<sup>1</sup>. This has been possible thanks to the Python library *CFI* (C Foreign Function Interface) [31], which is also able to provide a shared library Python-API compatible as output. In the following Figure the flow chart between Tensorflow Lite and the accelerator library can be seen:



**Figure 4.4:** Flow Chart with accelerator

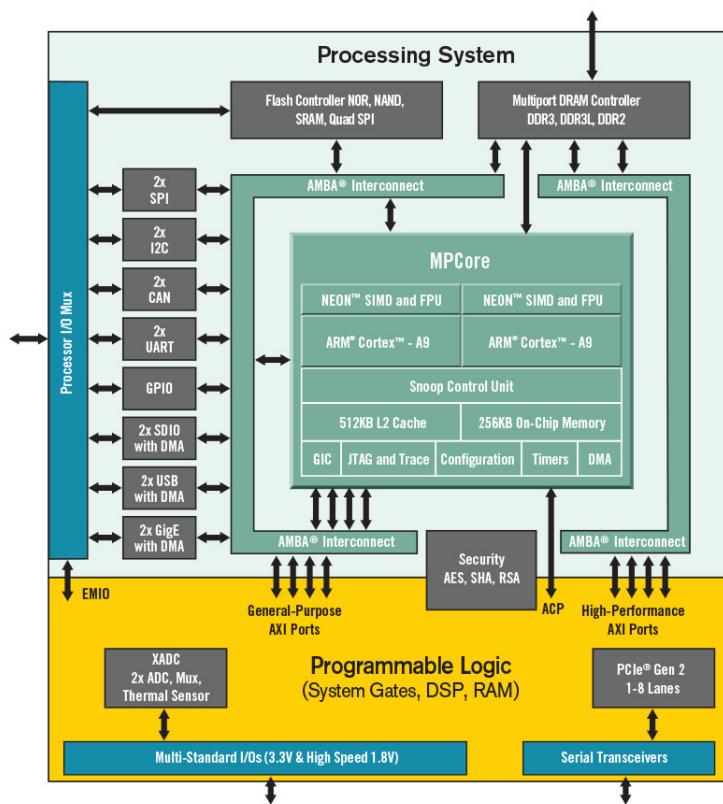
---

<sup>1</sup>See Appendix A

## 4.3 System Level

As it can be seen from Figure 4.5, it is divided in two big blocks:

- Processing System: The processing system (in Figure 4.6 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.



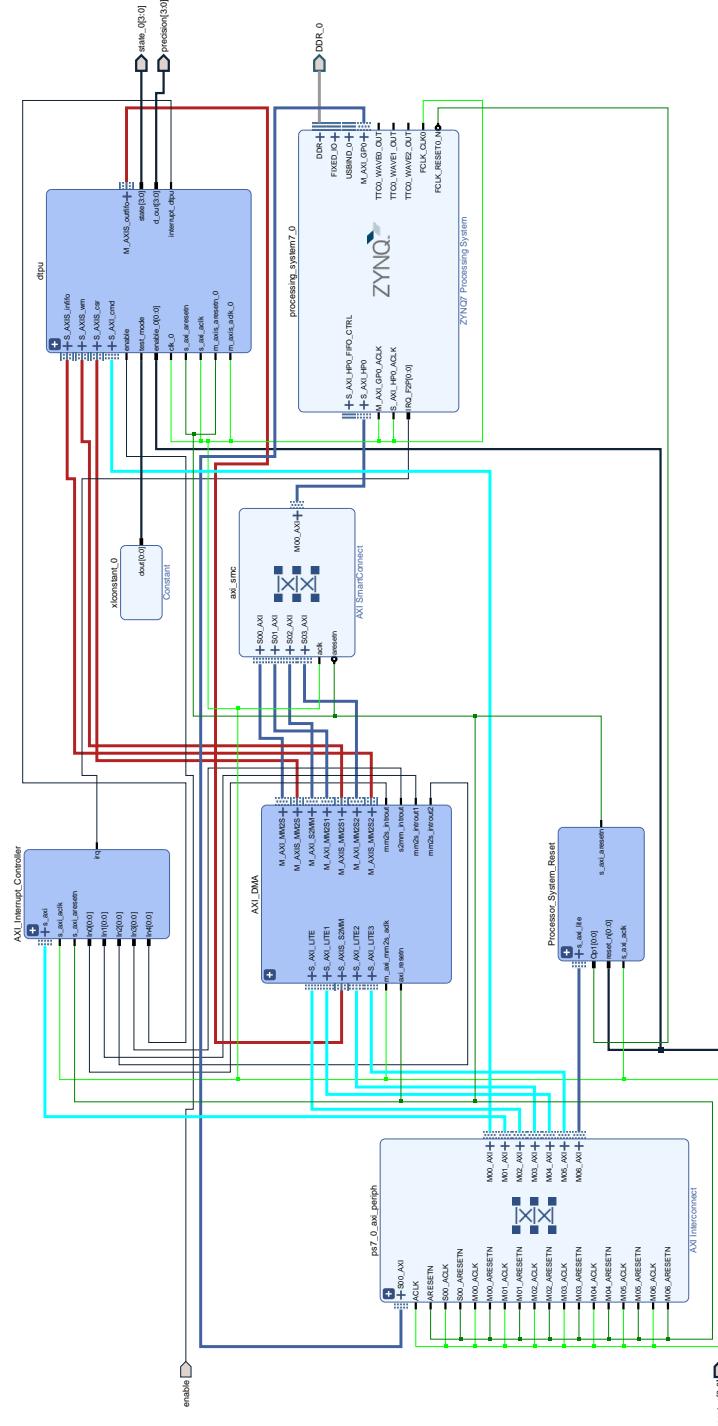
**Figure 4.5:** Zynq 7000 SoC [32]

Furthermore, the Programmable Logic in Figure 4.6 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.

## 4. System Development

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



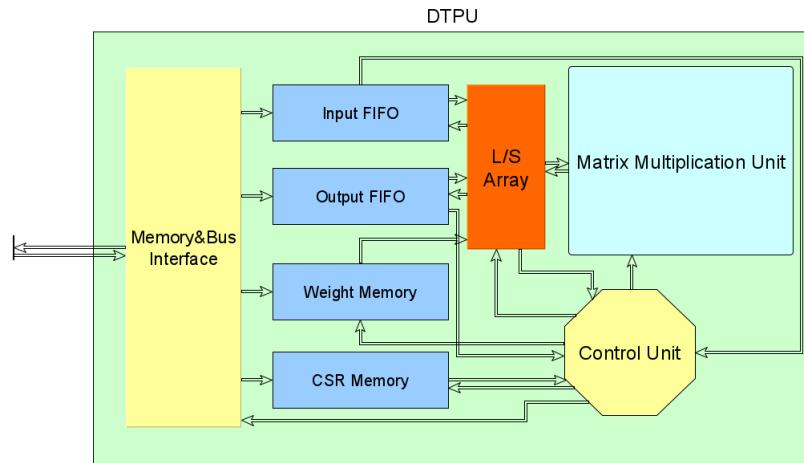
**Figure 4.6:** System view hosted in the PL <sup>2</sup>

<sup>2</sup>Except for the Zynq Processing system

## 4.4 DTPU, the hardware accelerator

The hardware accelerator, named *Cogitantium*<sup>3</sup>, *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.

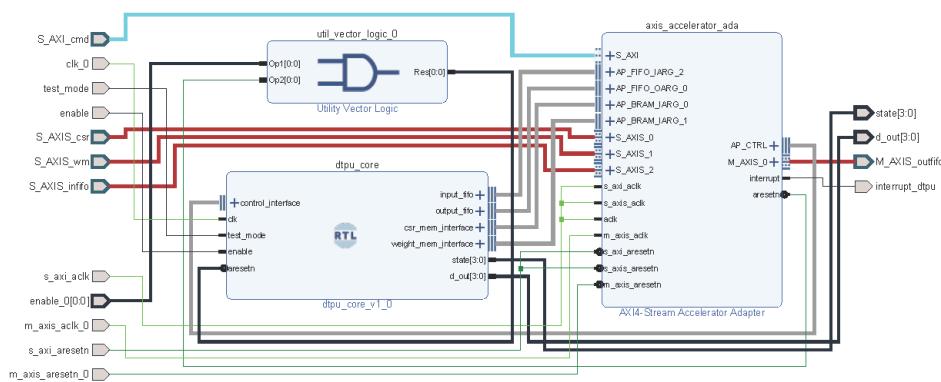
Figure 4.7 presents the Logical block diagram of the accelerator.



**Figure 4.7:** 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 4.8.



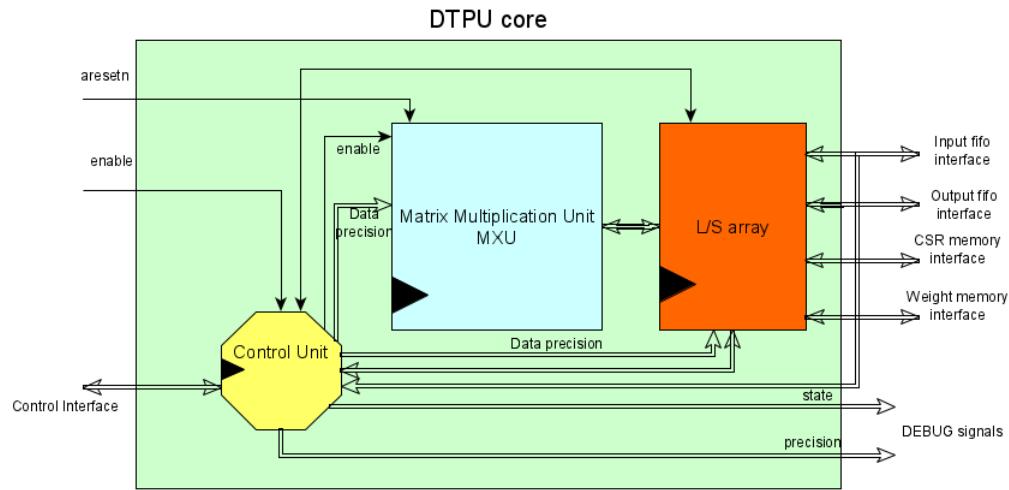
**Figure 4.8:** Real RTL view of DTPU accelerator

<sup>3</sup>Thoughtful

## 4. System Development

---

The latter has allowed to completely focus the work on the DTPU core<sup>4</sup>, which has become:



**Figure 4.9:** RTL view of DTPU core

Where the sub-units:

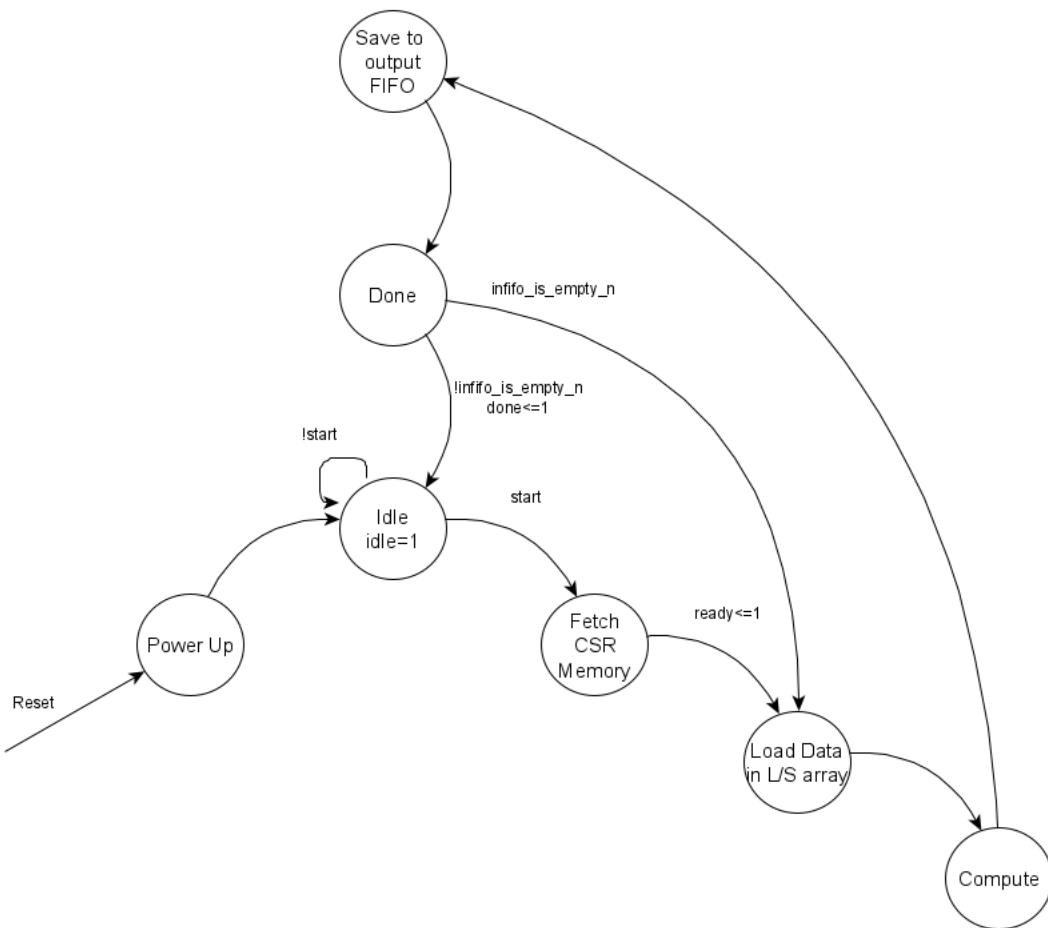
- L/S array provides the data for the Matrix Multiplication Unit, especially the weight data are reused across several executions and therefore loaded once.
- Control Unit is in charge of handling handshake signals for transferring the ownership of the data (data transferred by the DMA from the Main 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.

---

<sup>4</sup>See Appendix B

#### 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 4.10:** 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<sup>5</sup>).
- *Fetch CSR Memory* state is in charge of retrieving 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<sup>6</sup>.
- *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.

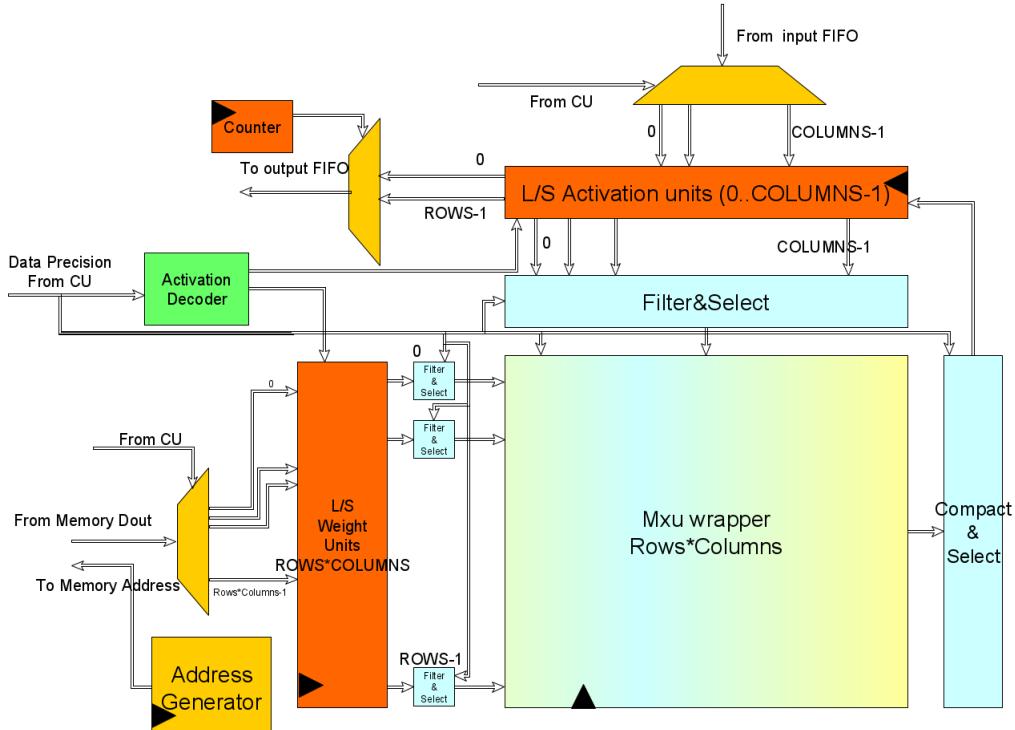
<sup>5</sup>Input Data, Weight data and CSR data

<sup>6</sup>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<sup>7</sup>.

#### 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 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 4.11 as block diagram.



**Figure 4.11:** A detailed view of the DTPU core datapath. Enable and resets signals for clocked units has been omitted for improving readability.

<sup>7</sup>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 Figure 4.11, the brawn of the accelerator is the MXU wrapper, which contains the symmetric matrix of MACs with variable precision. Regarding the other blocks:

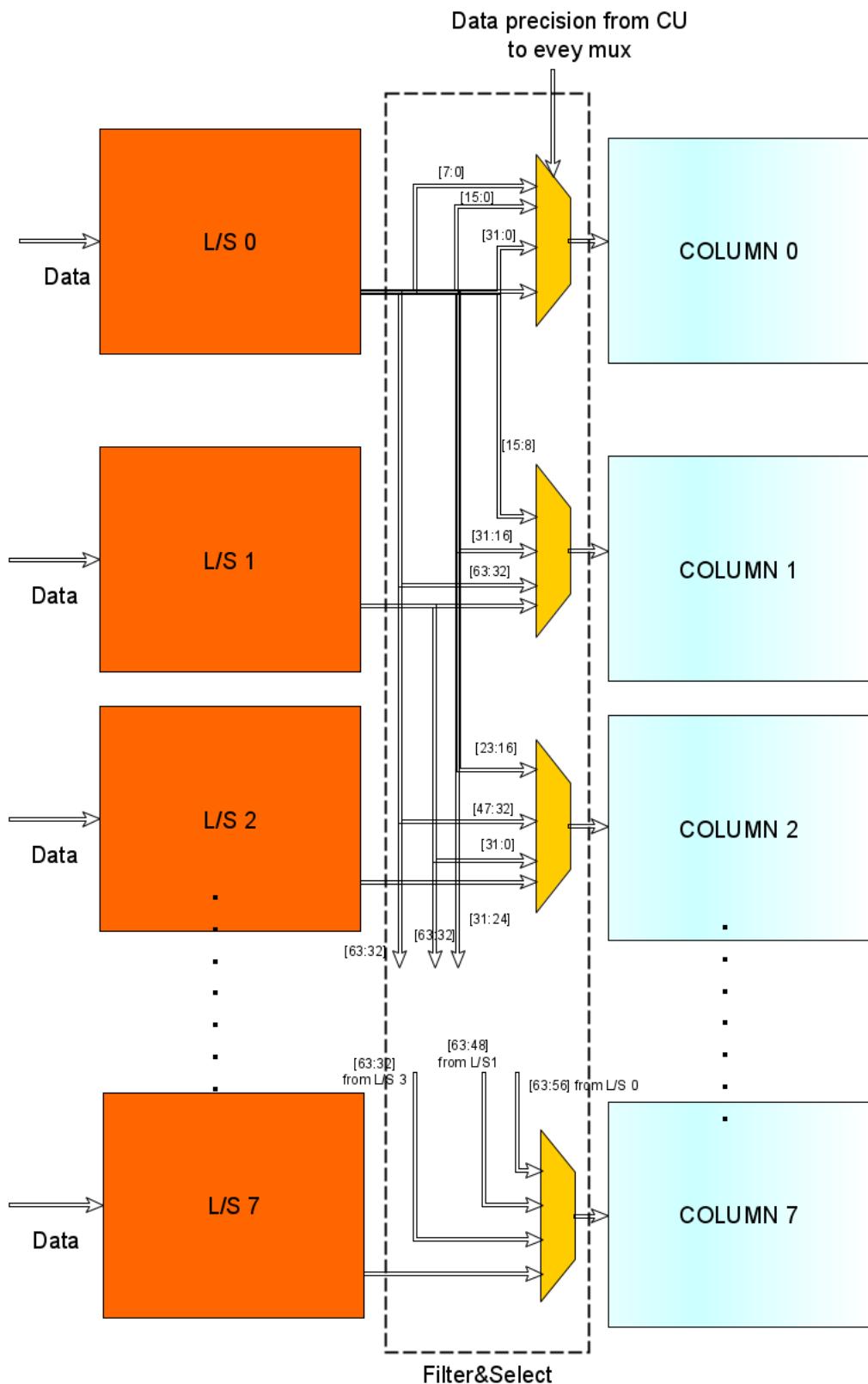
- Activation Decoder: It is able to generate the right activation signals for the L/S units, depending on from the current data precision and MXU size.
- Muxes and DeMuxes: Their purpose is to feed the right data from/to memory to/from the right units. The counter (from 0 to ROWS-1) in the Mux for the 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 Figure 4.12. The same can be applied for each row of L/S Weight units.



**Figure 4.12:** Data Distribution of Filter&Select unit for a MXU size of 8x8

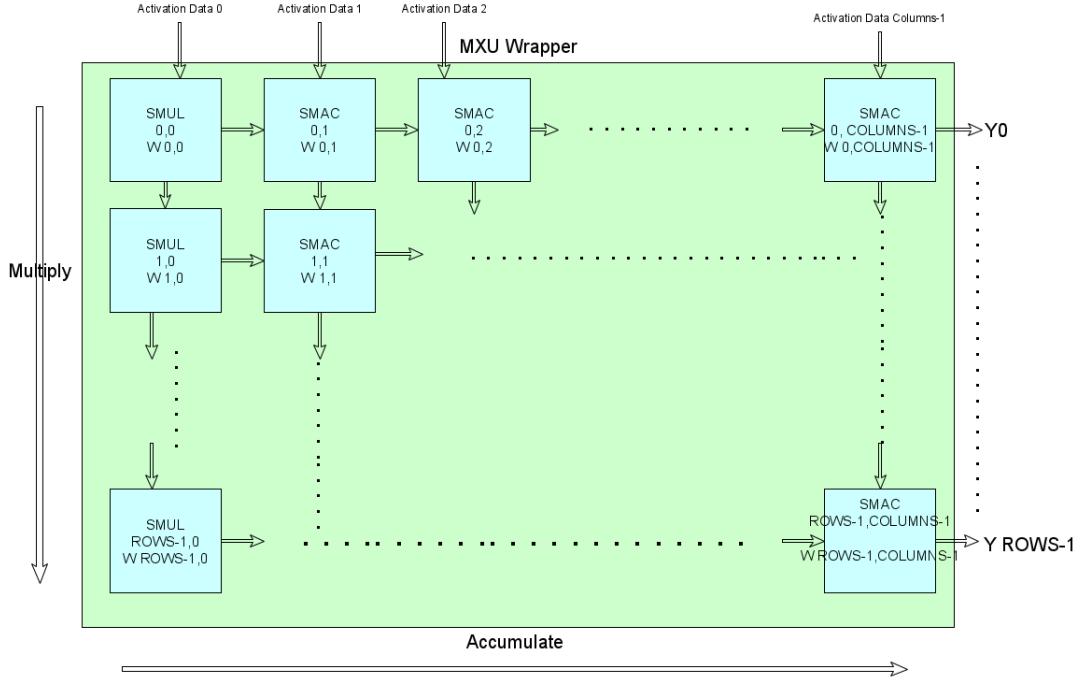
In case the required precision is on 16 bit, with the same MXU size, two L/S units for activation are activated ( $2^*ROWS$  for the L/S weight units) and will feed the respective Columns. The reason behind the two active L/S units is that in 64 bit, only 4 16-bit values can be packed. Increasing the 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 is 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 4.13:** MXU internal 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 Figure 4.11). 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:

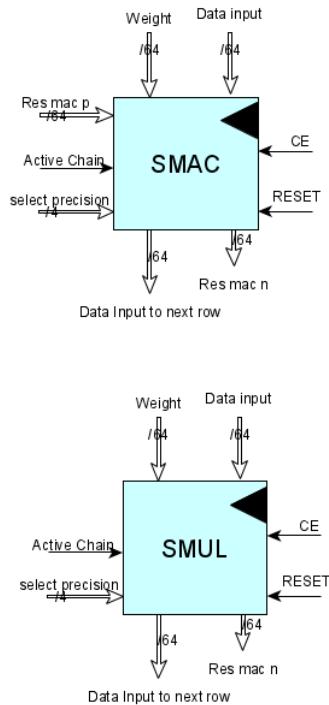
$$MACOPS = \text{ROWS} * \text{COLUMNS} \text{ per } \# \text{ clock cycle required for a single unit}$$

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 Figure 4.14.

A detailed view of SMAC (Sub unit Multiply and Accumulate) and SMUL(Sub unit Multiply), the Processing Elements, is given in Figure 4.14.



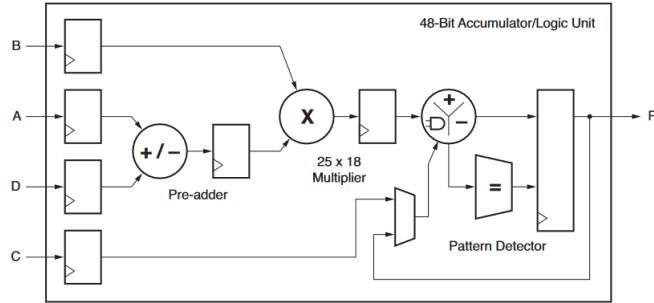
**Figure 4.14:** SMAC and SMUL details

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) the Processing Elements can compute vectorized operations. With the help of *active chain* signal (active low, otherwise it is a 64 bit computation) and data width fixed to 64 bit, 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.

#### 4. System Development

---

SMAC and SMUL units have been designed, internally, using Vivado DSP primitives [38], which a general schema can be appreciated in Figure 4.15:



**Figure 4.15:** DSP Slice Functionality [38]

Allowing fitting two computation (referring to SMAC) in one single unit<sup>8</sup> 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.

---

<sup>8</sup>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.

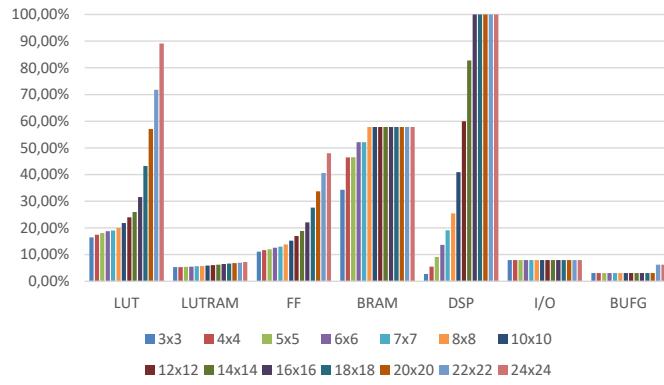
## 5.2 Utilization Factor

An important aspect of an embedded system is the on-die utilization area. Those kinds of system are usually deployed on tight 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.

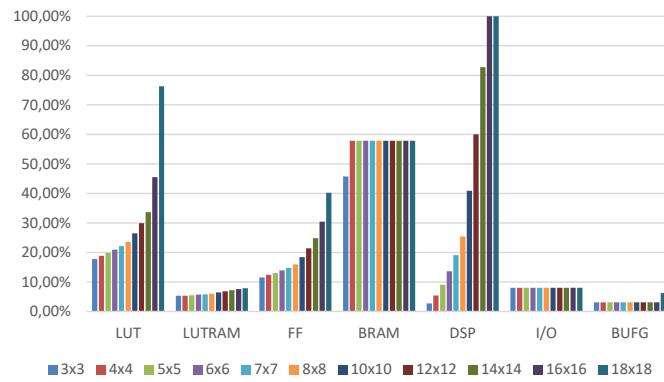
In the following Figures, utilization results are presented for each data type, where the Matrix Multiplication Unit sizes are pushed as much as the timing requirements are meet:

- Integer 8 bit:



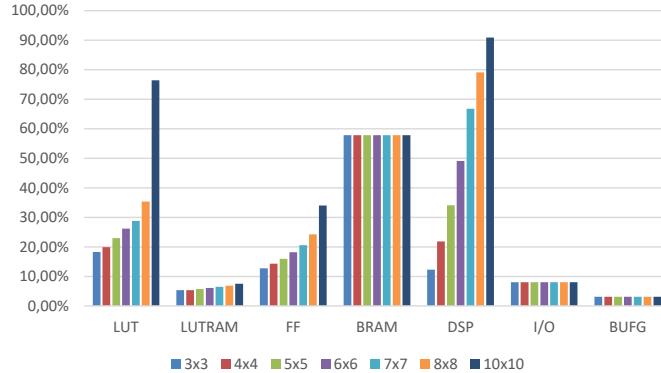
**Figure 5.1:** Post Implementation Utilization Factor of integer 8 bit PEs and clock frequency of 30 Mhz

- Integer 16 bit:



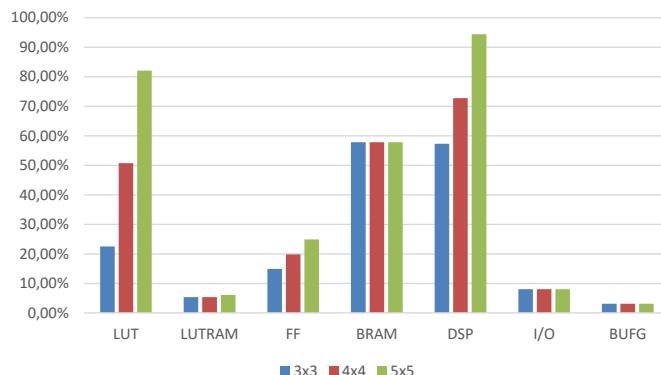
**Figure 5.2:** Post Implementation Utilization Factor of integer 16 bit PEs and clock frequency of 30 Mhz

- Integer 32 bit:



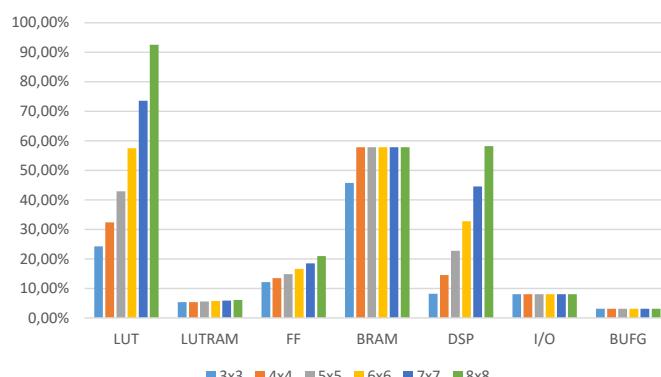
**Figure 5.3:** Post Implementation Utilization Factor of integer 32 bit PEs and clock frequency of 30 Mhz

- Integer 64 bit:



**Figure 5.4:** Post Implementation Utilization Factor of integer 64 bit PEs and clock frequency of 30 Mhz

- Brain Floating point 16:

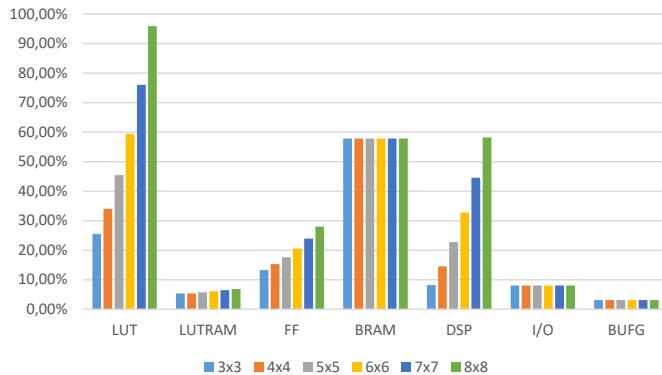


**Figure 5.5:** Post Implementation Utilization Factor of bfp 16 bit PEs and clock frequency of 30 Mhz

## 5. Results

---

- Floating point 32:



**Figure 5.6:** Post Implementation Utilization Factor of fp 32 bit PEs and clock frequency of 30 Mhz

It can be seen that the trend for integer 8 and 16 is similar (Figure 5.1 and 5.2). It is a 1 to 1 mapping between the PE and the DSP entity on the board. Actually, the DSP entities are on 16 bit and using the 8 bit units the high 8 bits are gated to zeros.

As soon as the DSP are used the utilization of LUT and FF is linear in the sizes of Matrix Multiplication unit, while the DSP utilization is quadratic. At a full utilization of DSP entities the PEs start to be implemented in logic and it can be seen, in all the previous graphs, a sudden rise in the LUT utilization.

It is also worth to mention that the PEs on 64 bit integer are a special case of FPGA's utilization, they reach sooner than the other designs the full utilization. Every PEs in this configuration is using a 14 DSP entities for taking into account also the possibility of computing vectorized operations as previously mentioned.

Regarding the floating point units, it has been used the same hardware unit for both the fp32 and bfp16, since they have the same exponent bit length but different mantissa length (this also allows to have the same numerical stability). Relying on the synthesis process to properly optimize the different units and discard, where necessary, the unused hardware. In fact, comparing Figure 5.5 with Figure 5.6, there is a slight different in the utilization of the LUT and a more remarkable difference in FF utilization.

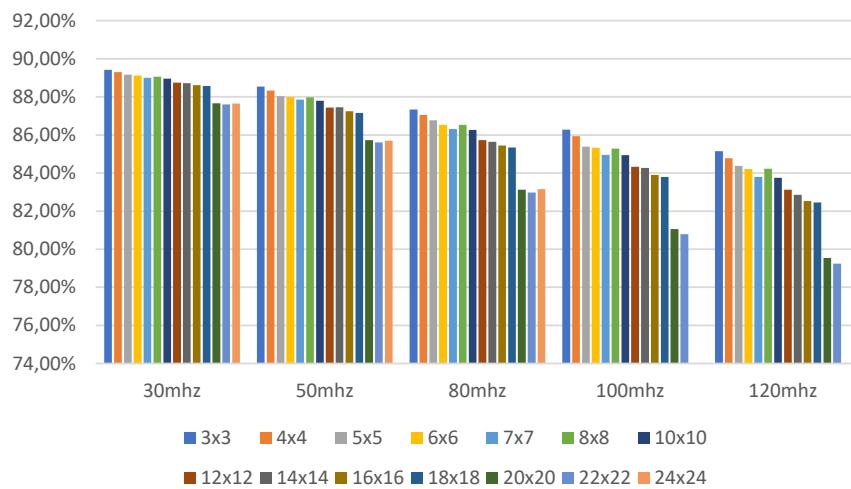
Increasing the clock frequency, the FPGA's utilization is reduced since with an increase of the Matrix Multiplication Unit sizes the design is not able anymore to meet the timing requirements, especially for the floating point units.

### 5.3 Energy and Power Consumption

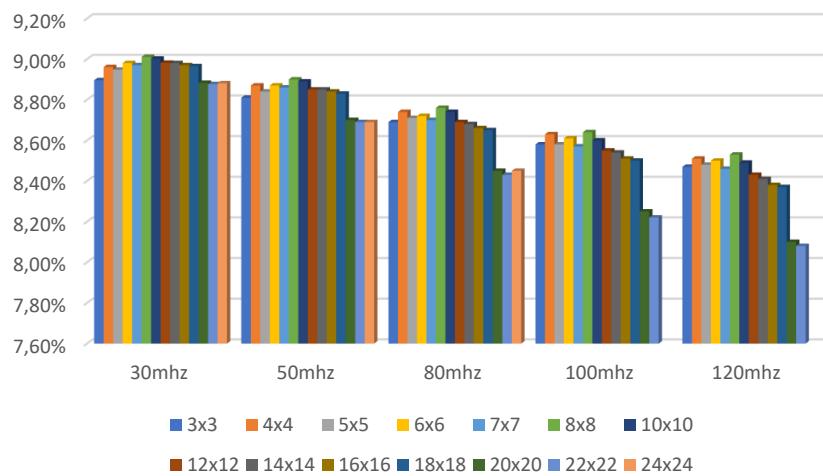
Energy and Power consumption are important factor, for a mobile device in which there is a limited battery capacity meanwhile for data centers stringent power ceilings due to cooling costs.

According to the Vivado Power estimation manual[39], the static power is calculated over all the FPGA resources. This is due to the hard estimation of the static power per single design. In the following Figures, estimations of power consumption from Vivado are presented for each data type and different clock frequencies:

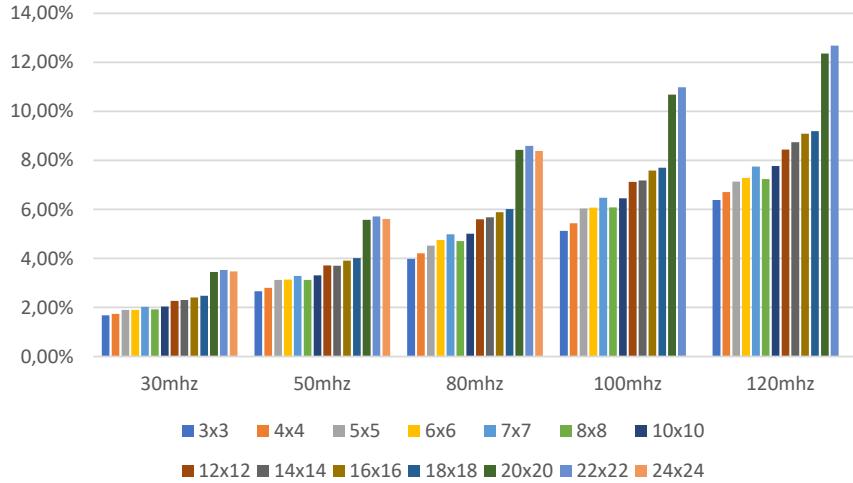
- Integer 8:



**Figure 5.7:** Post Implementation Power Consumption of Processing System for integer 8 PEs



**Figure 5.8:** Post Implementation Static Power Consumption Programmable logic for integer 8 PEs



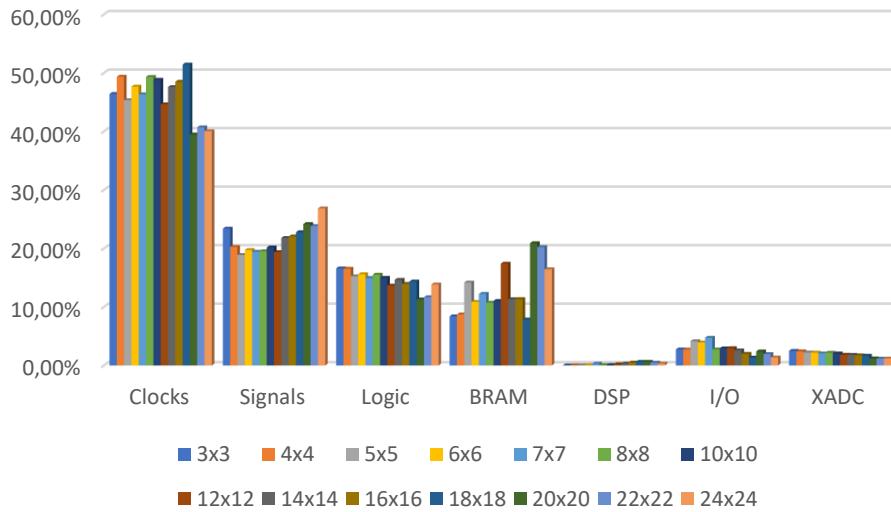
**Figure 5.9:** Post Implementation Dynamic Power Consumption per Programmable logic with integer 8 PEs

The previous Figures represent the behaviour of the power consumption with different Matrix Multiplicatio Unit and for different clock frequency. It is expressed as percentage with reference to the total power consumption of the SoC ( processing system and programmable logic). In fact it can be seen that the power consumption of the processing system and the static power consumption have a less impact on the total power consumption with an increase of the Matrix Multiplication Unit and the frequency. On the other hand, the dynamic power consumption 5.9, as expected, grows with a growing Matrix Multiplication Unit and the design frequency.

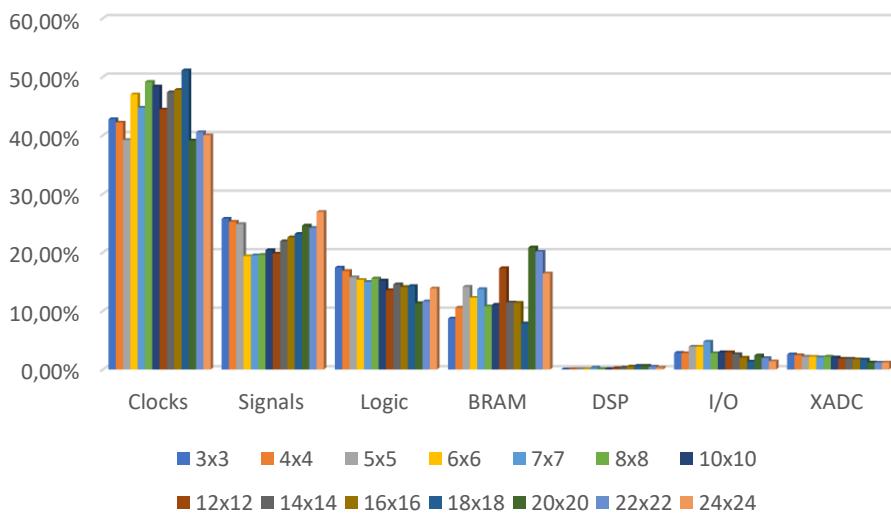
In the following Figures, the dynamic power consumption for each entities (in percentage, wrt the dynamic power in Figure 5.9) in the FPGA is analyzed for different clock frequencies.

As it is very well known, the clock distribution is one of the main source of power consumption, and it is confirmed from all the previos Figures. Also the interconnections, called *signals* in the figures, are power hungry (a bigger MXU leads to many, and longer, interconnections between PEs). In fact the clock distribution networks and the interconnections are the predominant entities of the dynamic power consumption of the programmable logic. The logic entity is containing all the power consumed by the FFs and LUTs, it looks like their power consumption is decreasing but it is only the percentage of the total dynamic power which is decreasing. It is worth to mention that the PEs (at least the majority of them) are implemented using the DSP entities. However, the power consumed by those entities is almost negligible, the DSPs are low power entities in the FGPA according to its datasheet [32].

Regarding the BRAM, I/O and XADC, with an increase or a decrease of the other entities impact they have a slightly modification of their impact on the power consumption.



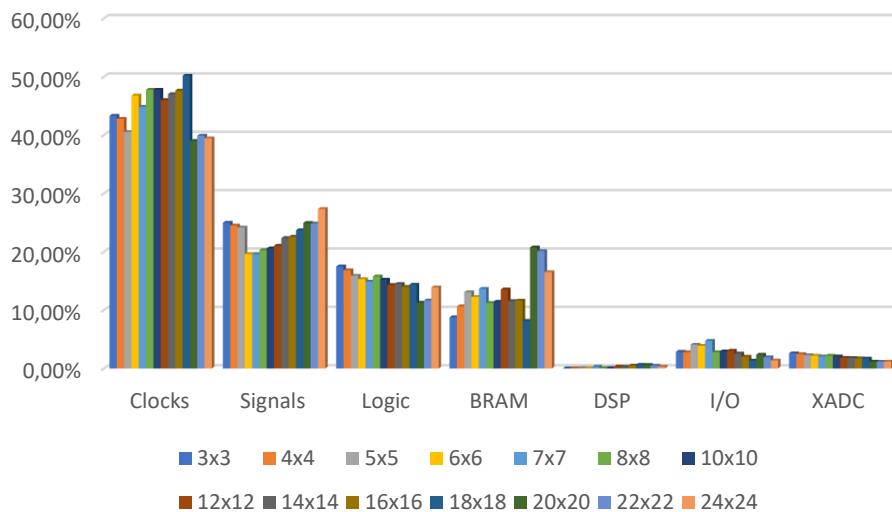
**Figure 5.10:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 8 PEs



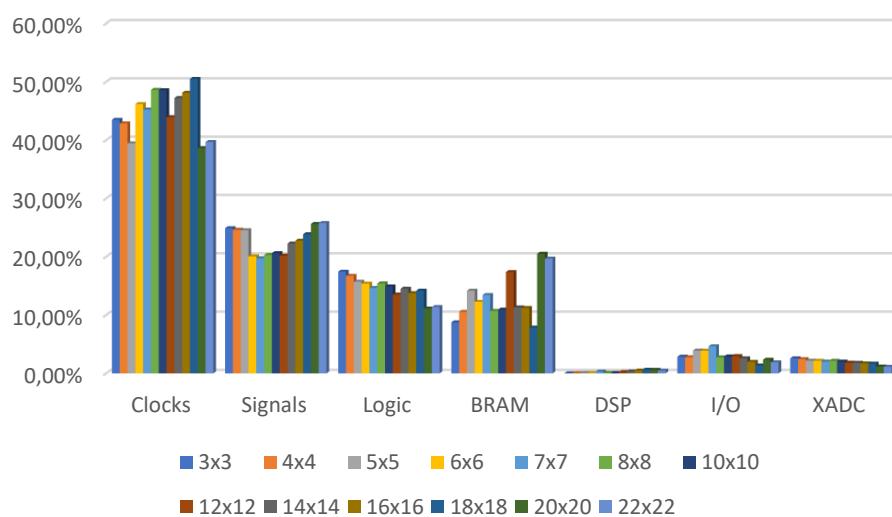
**Figure 5.11:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 8 PEs

## 5. Results

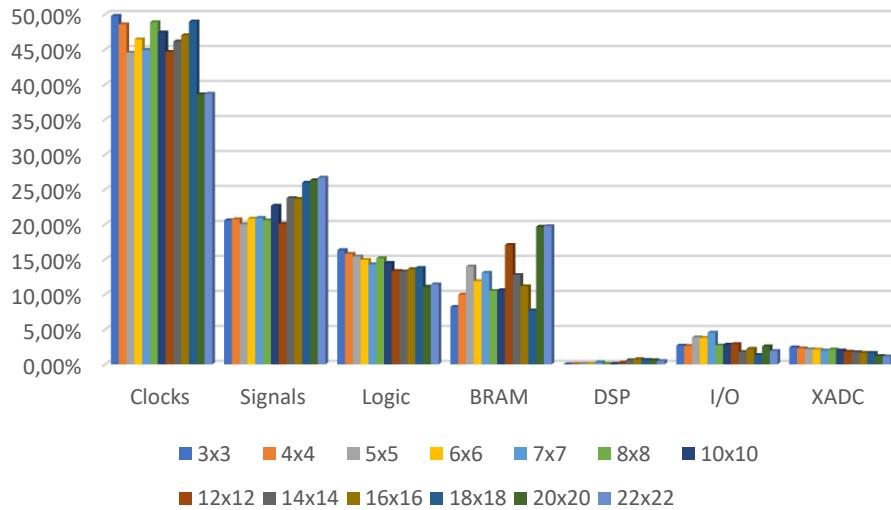
---



**Figure 5.12:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 80 MHz and integer 8 PEs



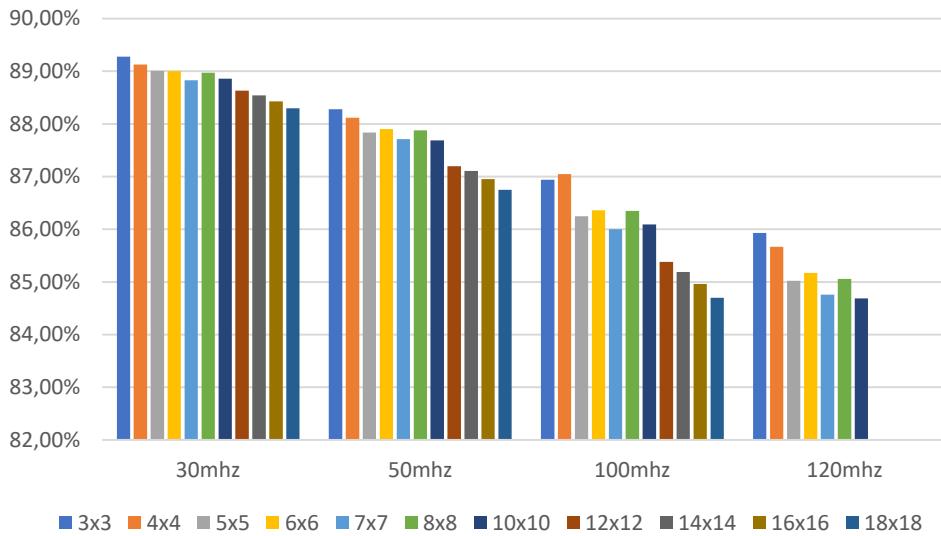
**Figure 5.13:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 100 MHz and integer 8 PEs



**Figure 5.14:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 120 MHz and integer 8 PEs

- Integer 16:

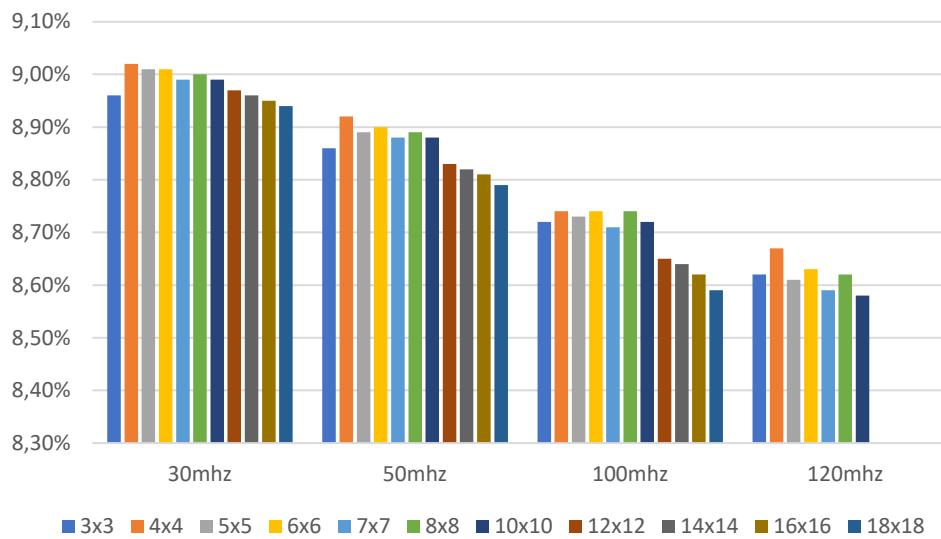
In the following Figures, the same considerations for the Integer 8 are still valid since the PEs are always implemented on the same DSP entity but without the higher 8 bits of the input values gated to zeros.



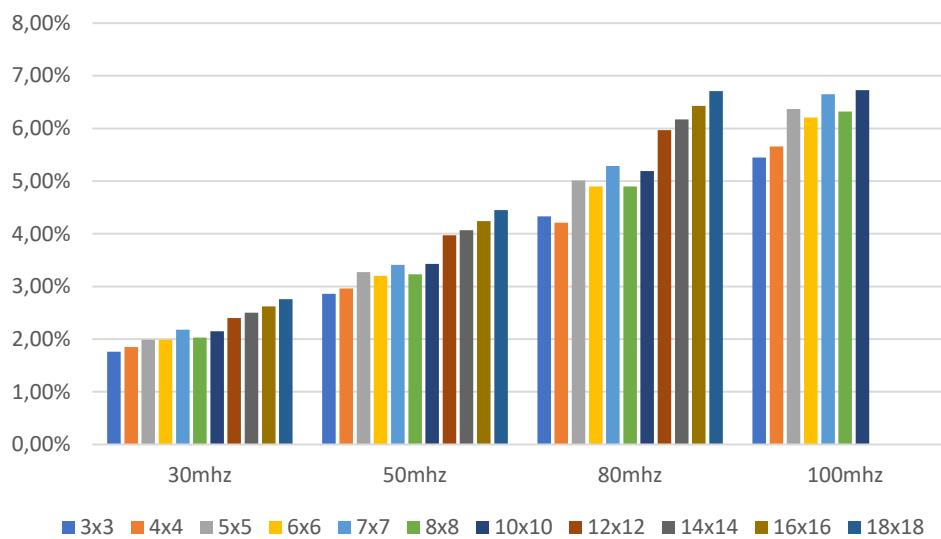
**Figure 5.15:** Post Implementation Power Consumption of Processing System for integer 16 PEs

## 5. Results

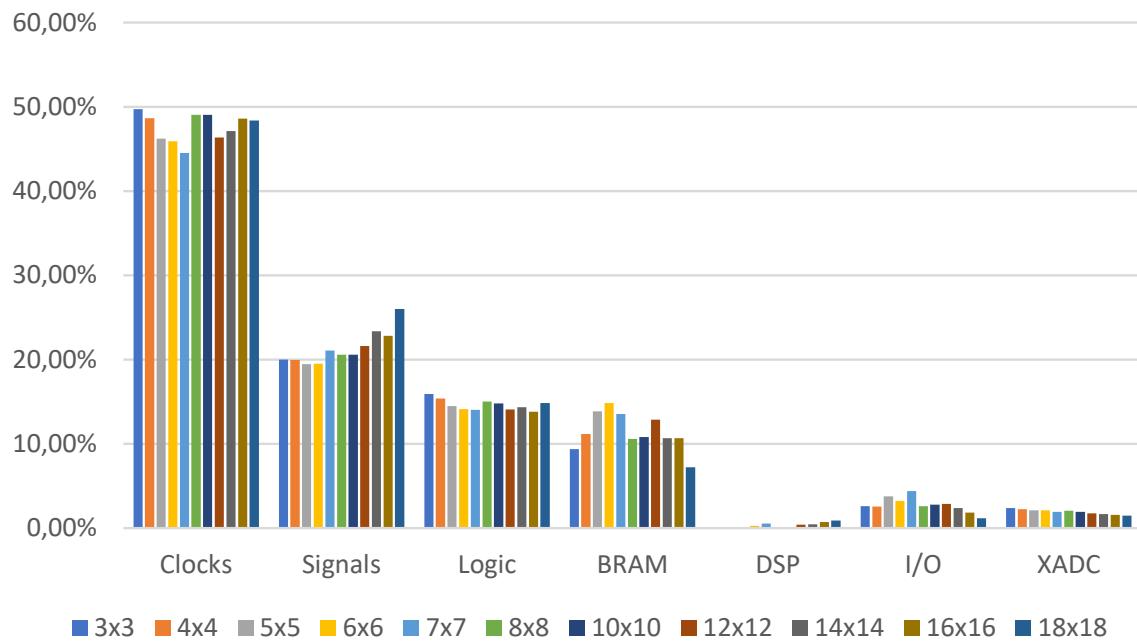
---



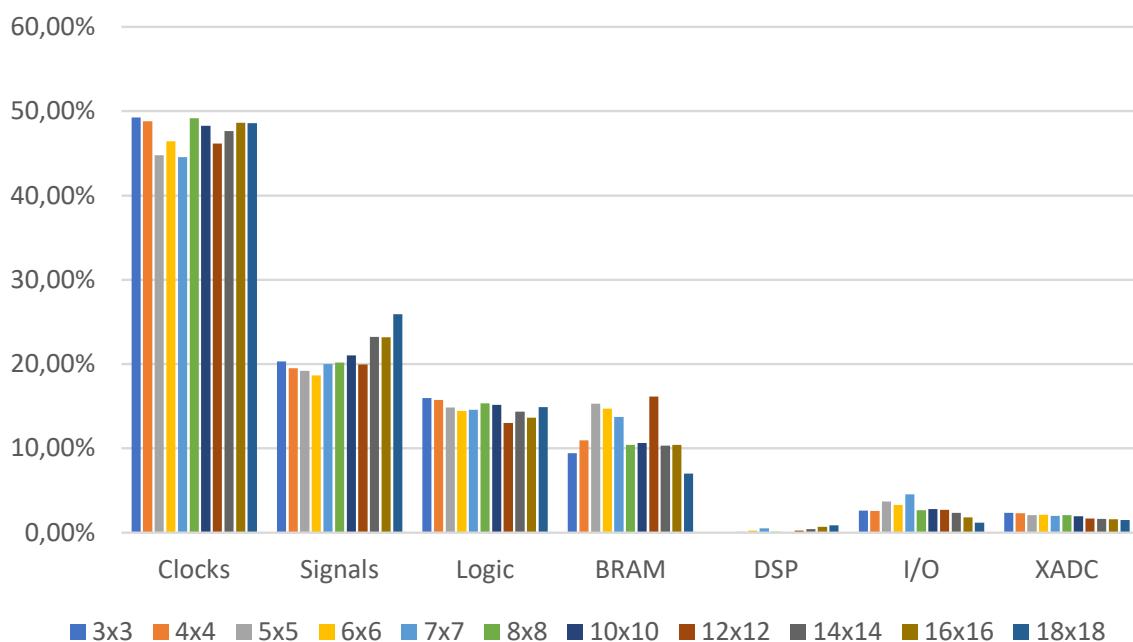
**Figure 5.16:** Post Implementation Static Power Consumption Programmable logic for integer 16 PEs



**Figure 5.17:** Post Implementation Dynamic Power Consumption per Programmable logic with integer 16 PEs



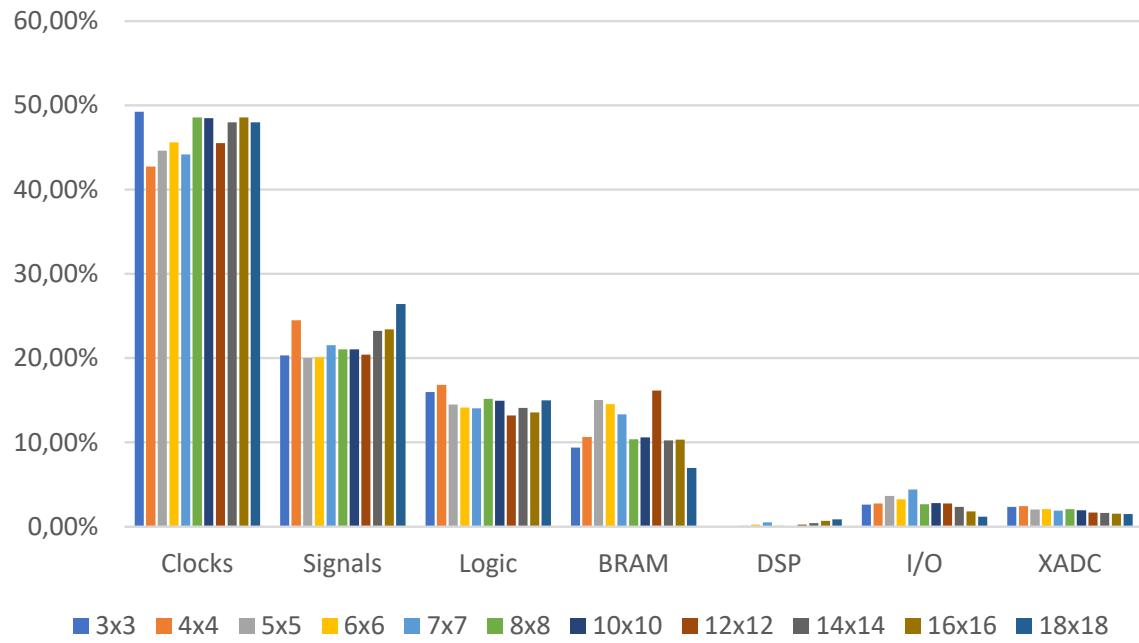
**Figure 5.18:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 16 PEs



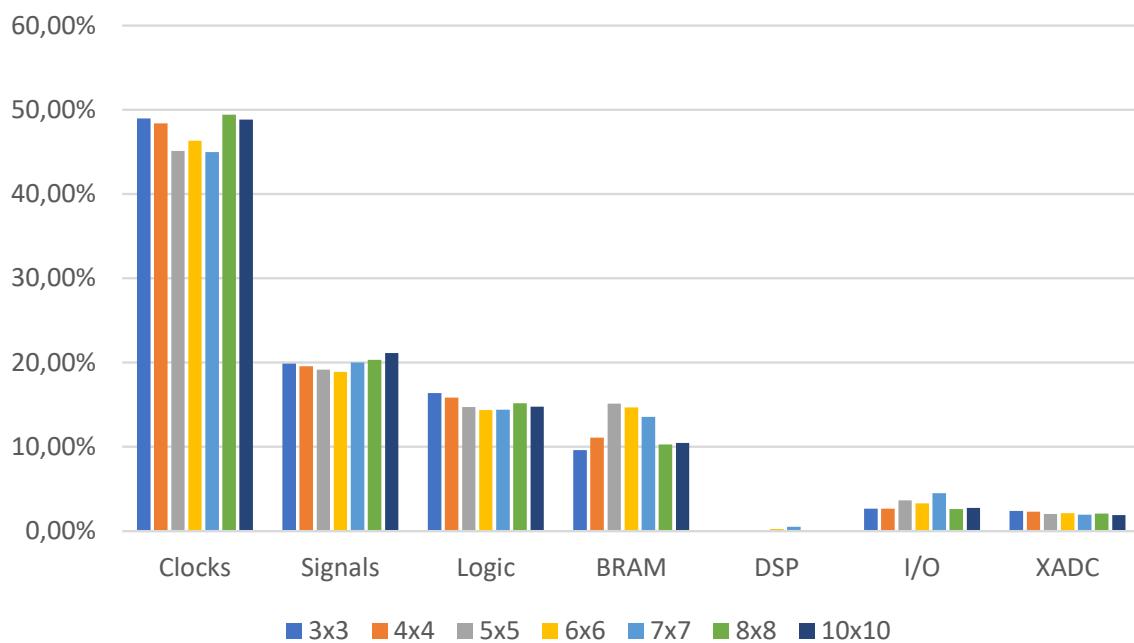
**Figure 5.19:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 16 PEs

## 5. Results

---



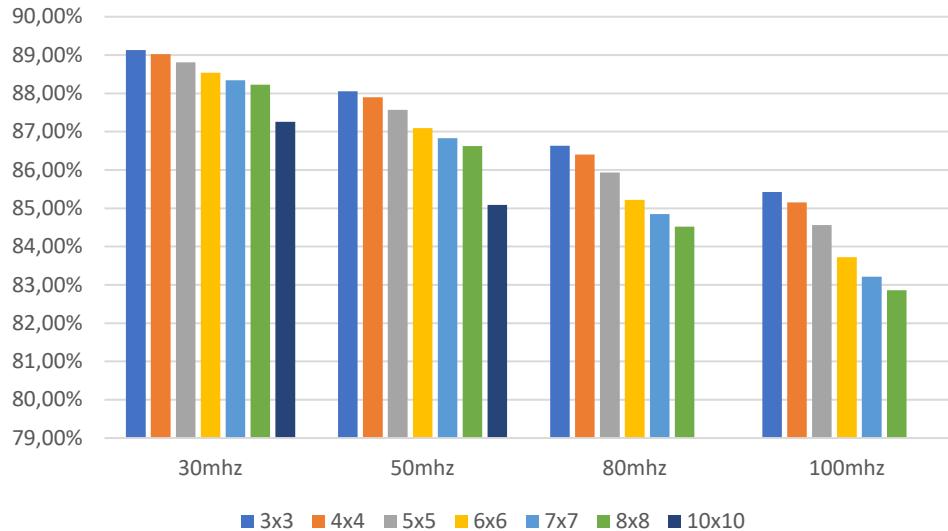
**Figure 5.20:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 80 MHz and integer 16 PEs



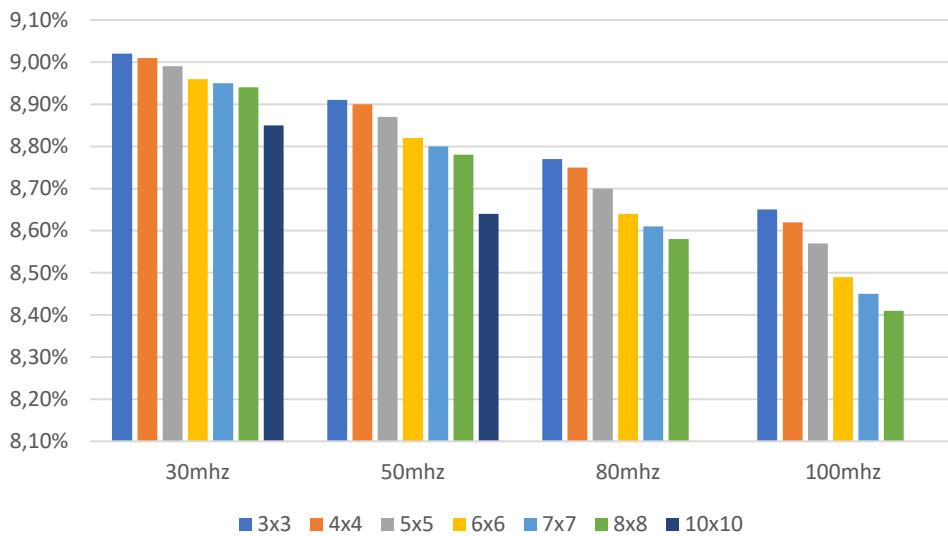
**Figure 5.21:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 100 MHz and integer 16 PEs

- Integer 32:

From now on, the MXU sizes and the frequencies will show a reduction in their values, this is mainly because big designs (with almost full FPGA's utilization) are not able to meet anymore the timing requirements.



**Figure 5.22:** Post Implementation Power Consumption of Processing System for integer 32 PEs



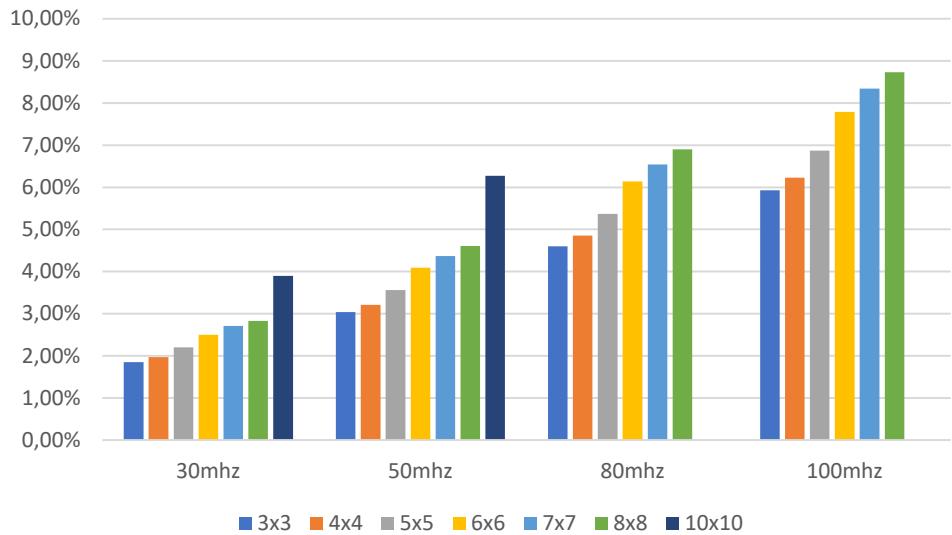
**Figure 5.23:** Post Implementation Static Power Consumption Programmable logic for integer 32 PEs

It is worth to mention the power consumed by the DSP entities, comparing to integer 8 and 16 PEs, is bigger. The main reason is that there is no more one to one mapping between PEs and DSP entities.

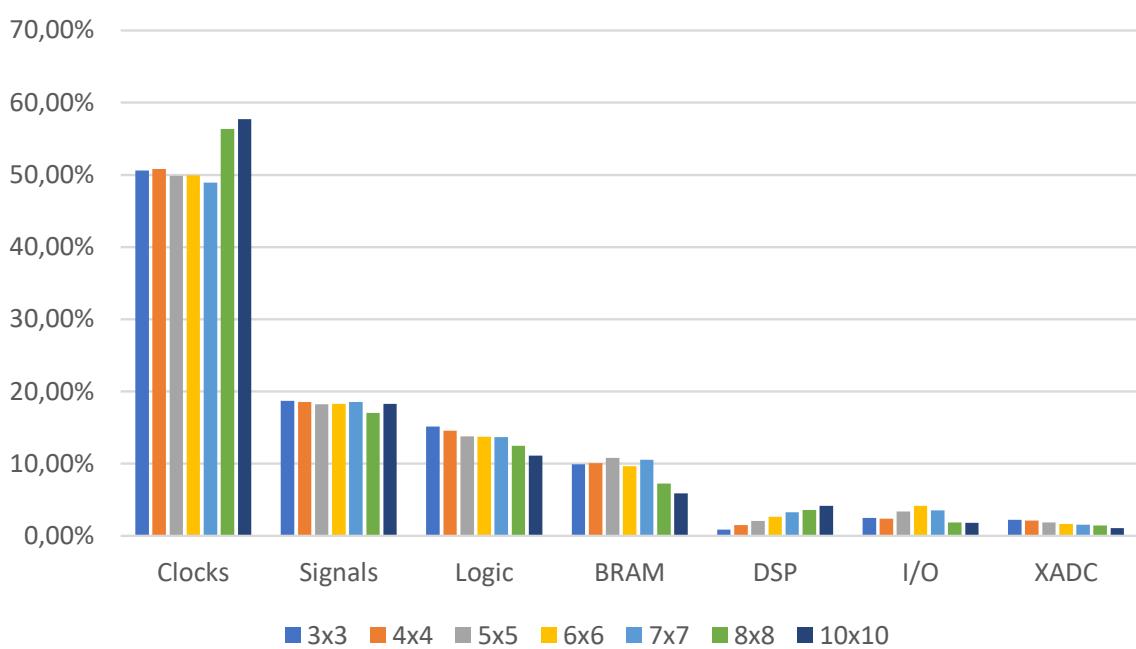
- Integer 64:

## 5. Results

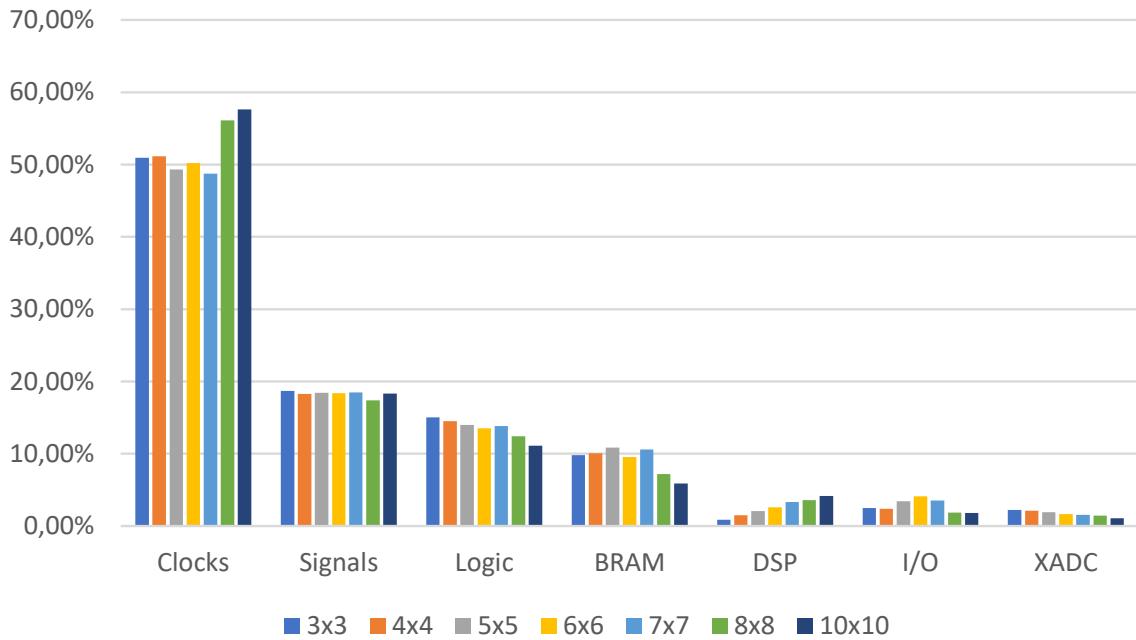
---



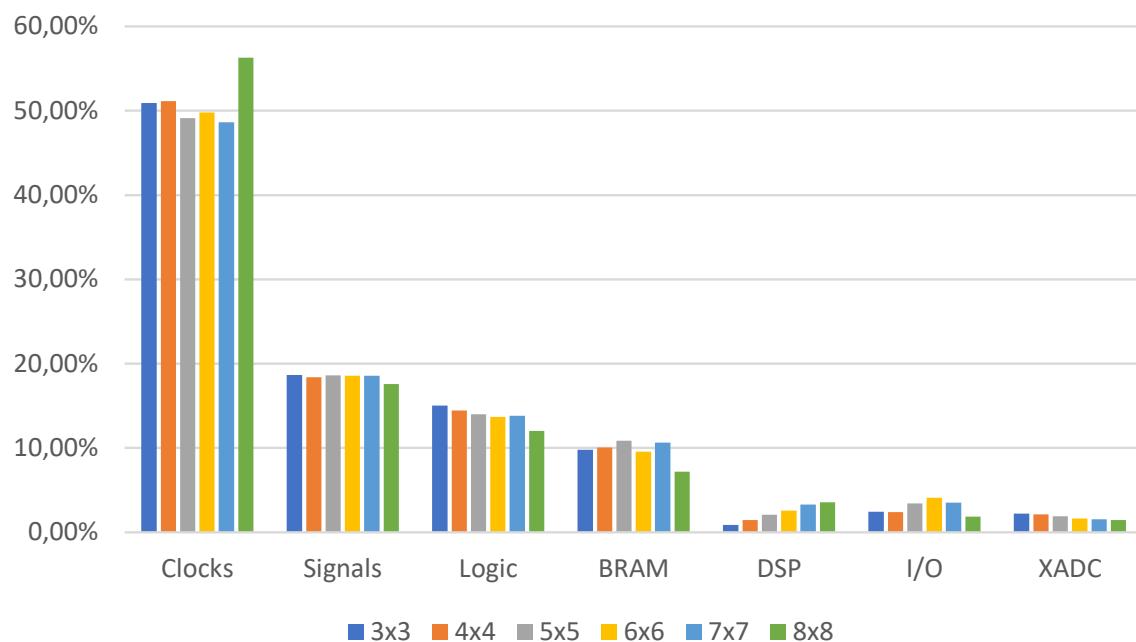
**Figure 5.24:** Post Implementation Dynamic Power Consumption per Programmable logic with integer 32 PEs



**Figure 5.25:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 32 PEs



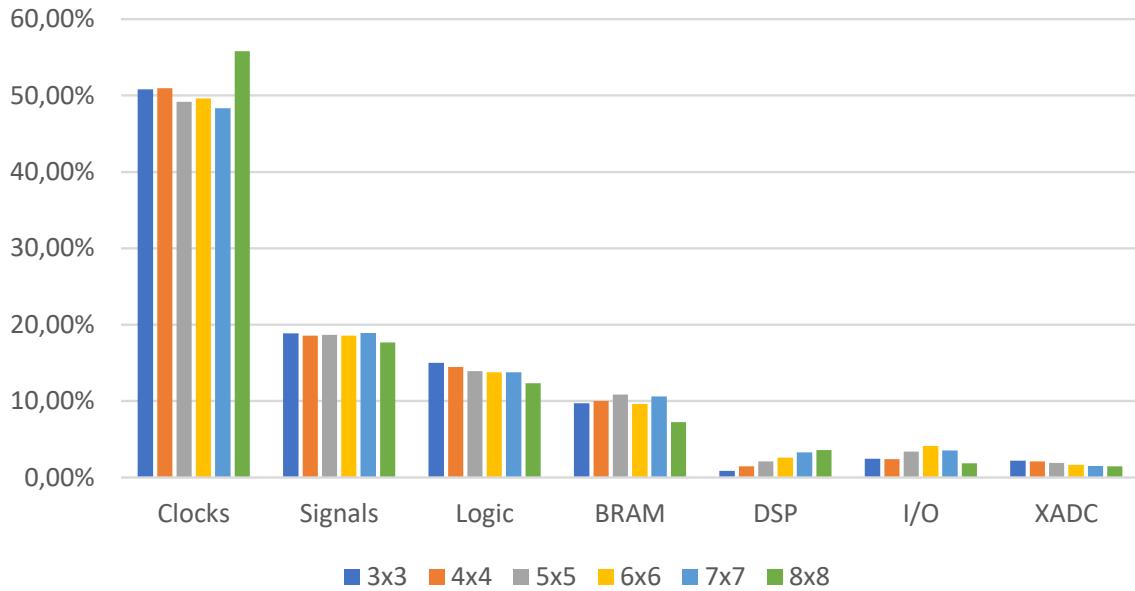
**Figure 5.26:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 32 PEs



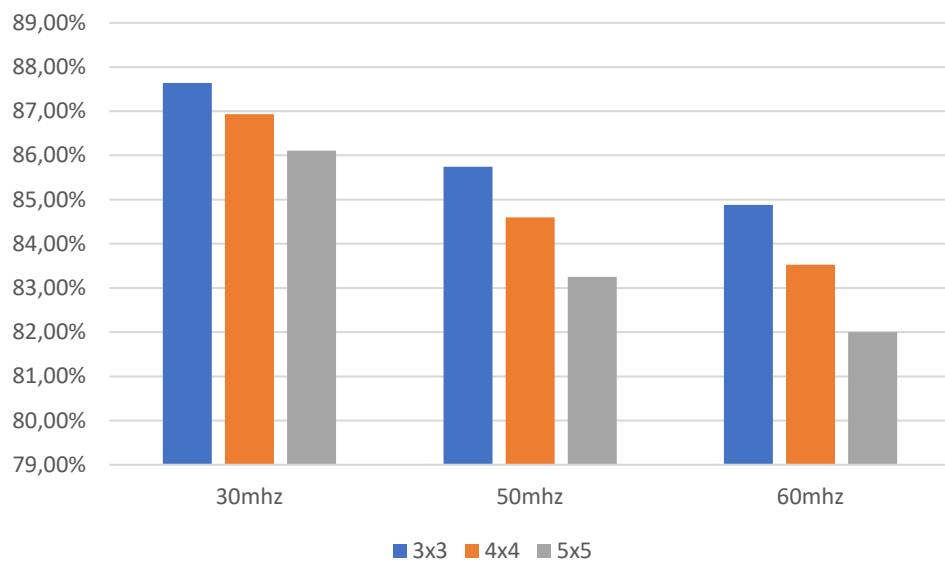
**Figure 5.27:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 80 MHz and integer 32 PEs

## 5. Results

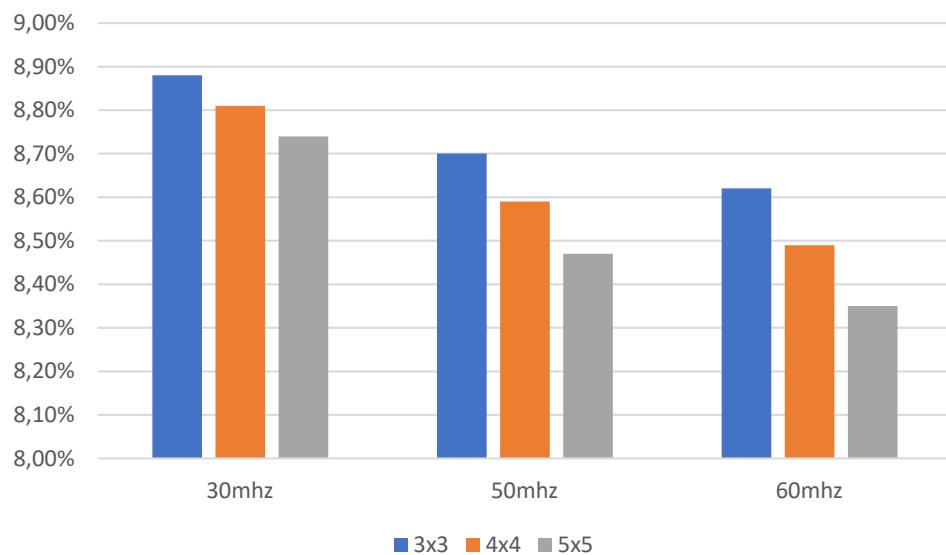
---



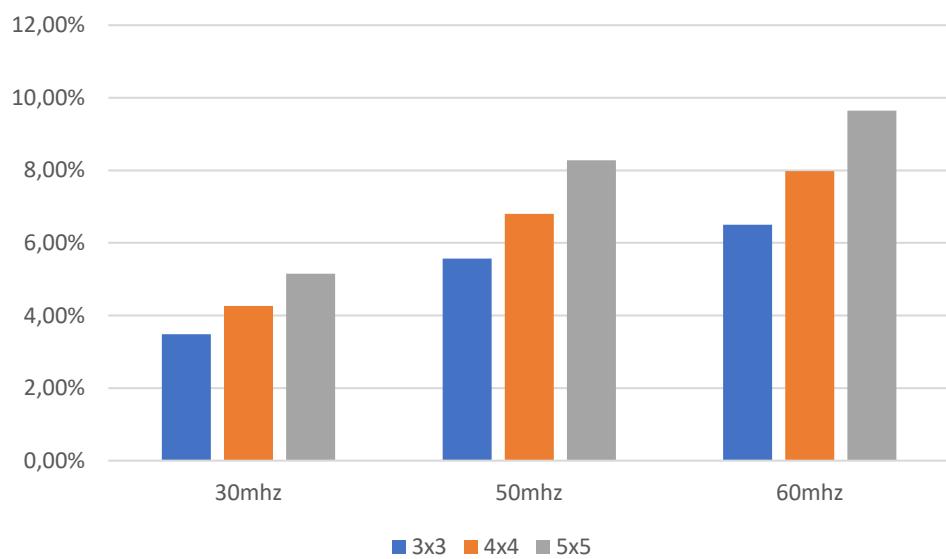
**Figure 5.28:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 100 MHz and integer 32 PEs



**Figure 5.29:** Post Implementation Power Consumption of Processing System for integer 64 PEs



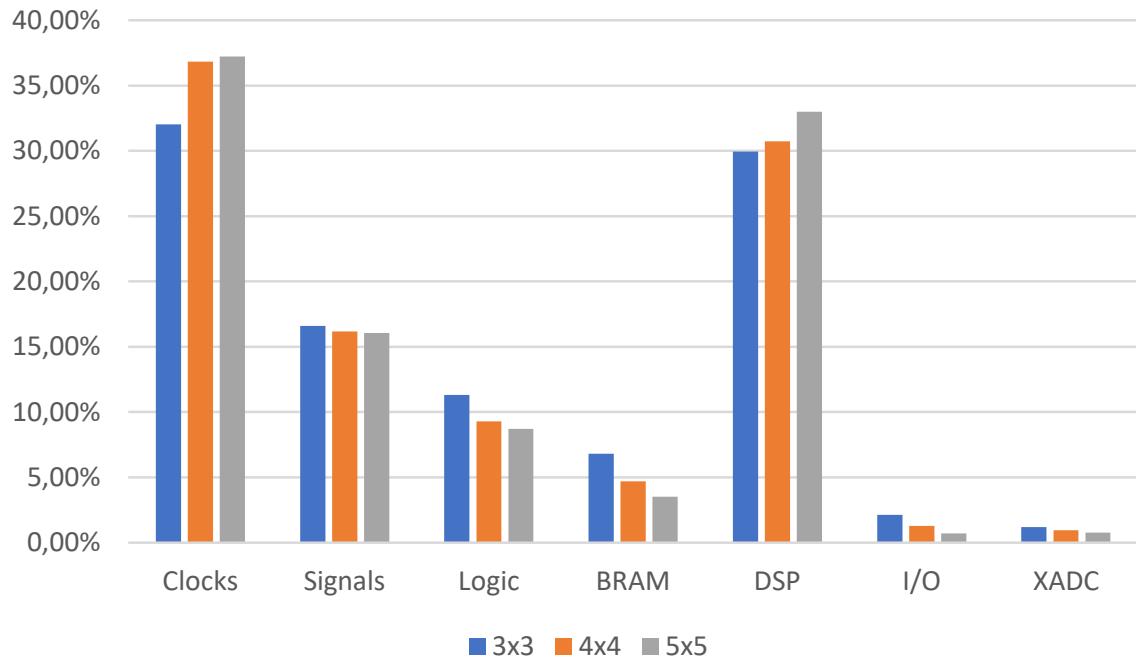
**Figure 5.30:** Post Implementation Static Power Consumption Programmable logic for integer 64 PEs



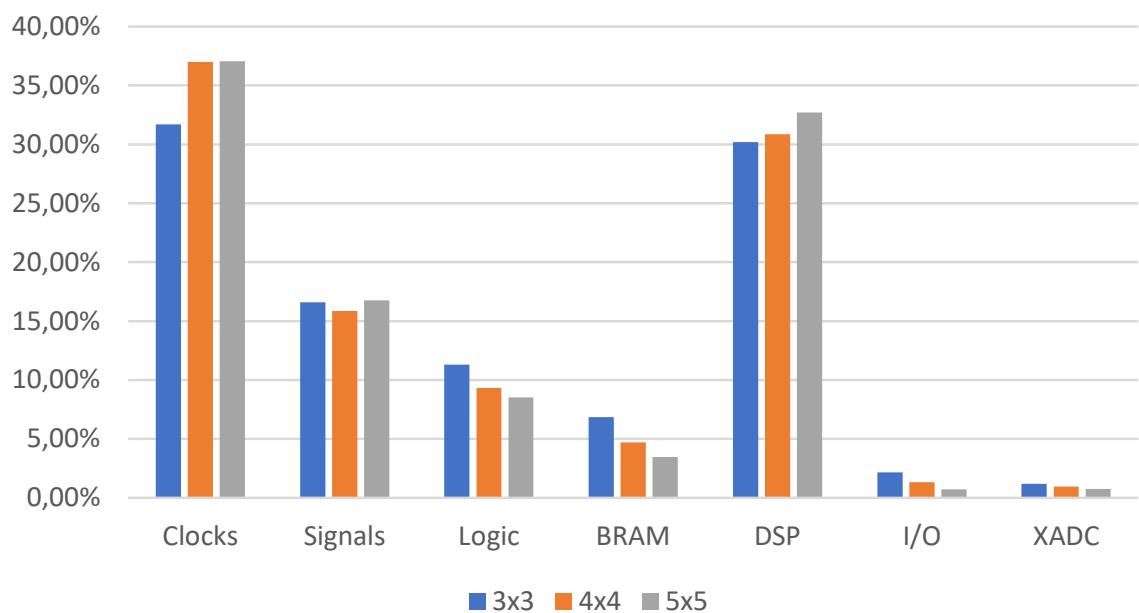
**Figure 5.31:** Post Implementation Dynamic Power Consumption per Programmable logic with integer 64 PEs

## 5. Results

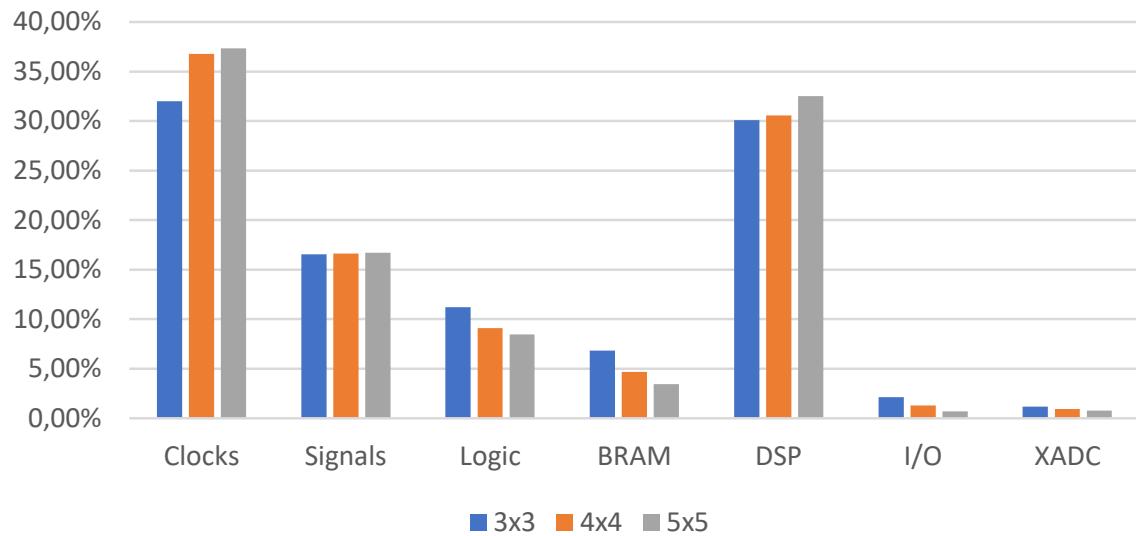
---



**Figure 5.32:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and integer 64 PEs



**Figure 5.33:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 50 MHz and integer 64 PEs



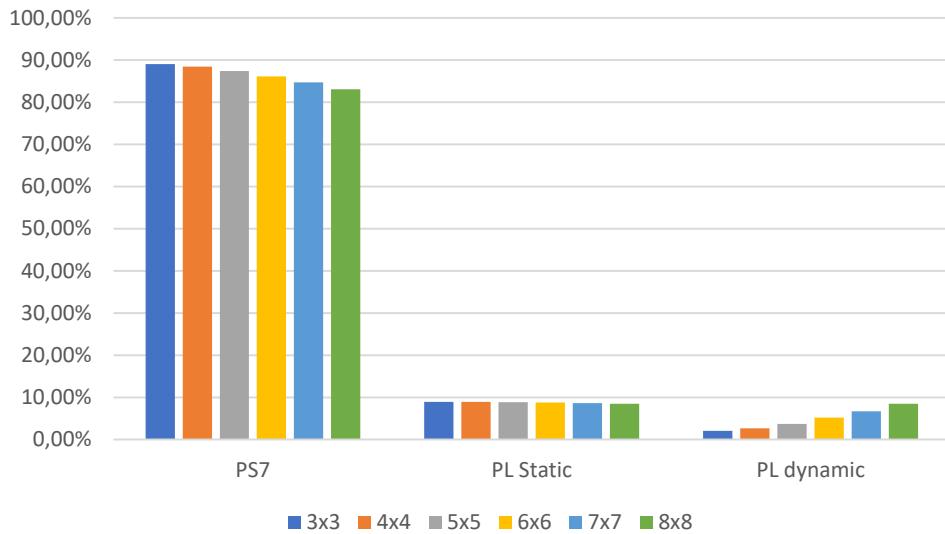
**Figure 5.34:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 60 MHz and integer 64 PEs

As mentioned in the Utilization chapter, the PEs on 64 bit integer are using 14 DSP entities (for having the possibility to compute vectorized operations on data). Therefore, this heavy utilization per PEs is impacting also the power consumed by the DSPs but as it can be seen in Figures fromi 5.32 to 5.34.

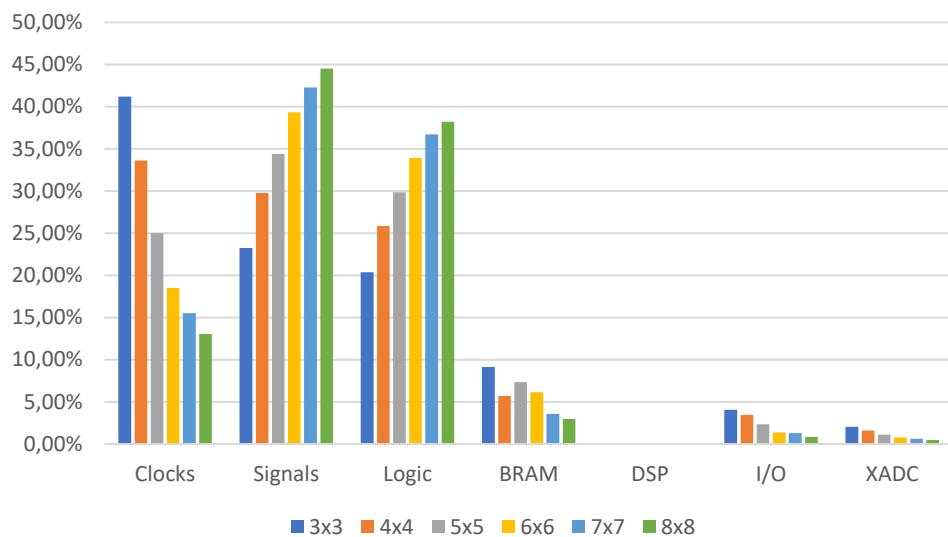
## 5. Results

---

- Brain floating point 16:

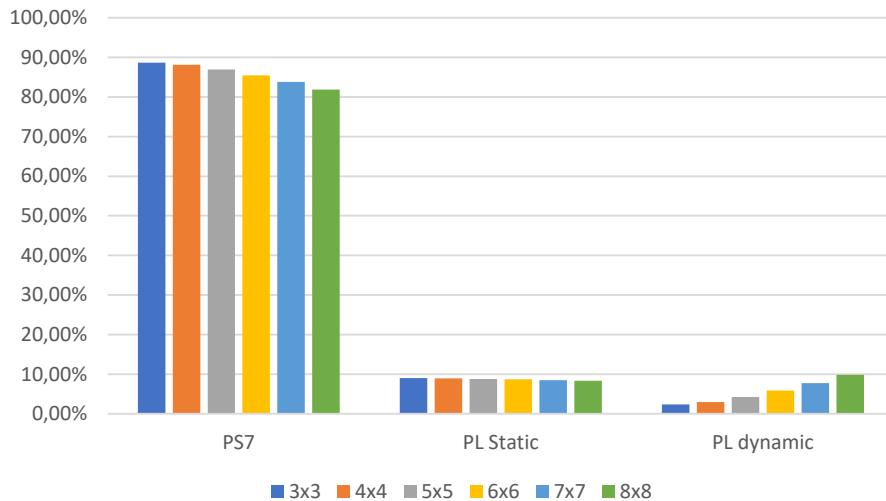


**Figure 5.35:** Post Implementation Power Consumption for bfp16 PEs

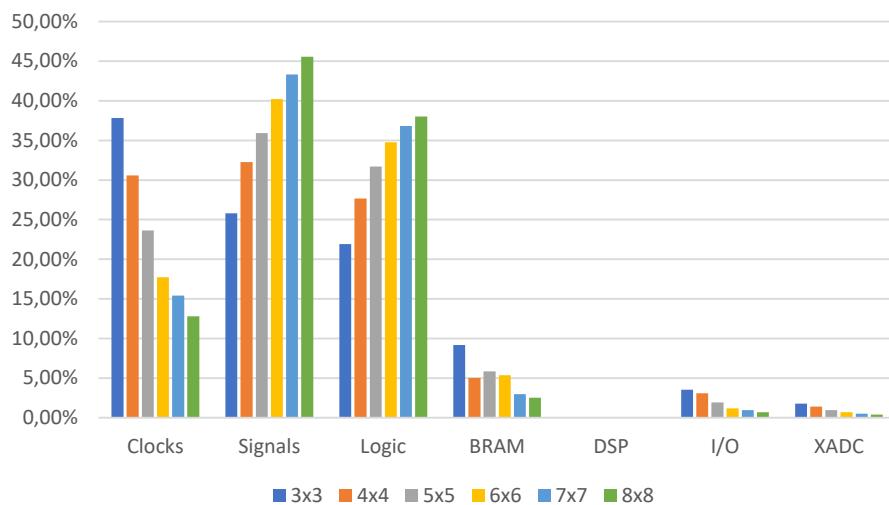


**Figure 5.36:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and bfp16 PEs

- Floating point 32:



**Figure 5.37:** Post Implementation Power Consumption for fp32 PEs



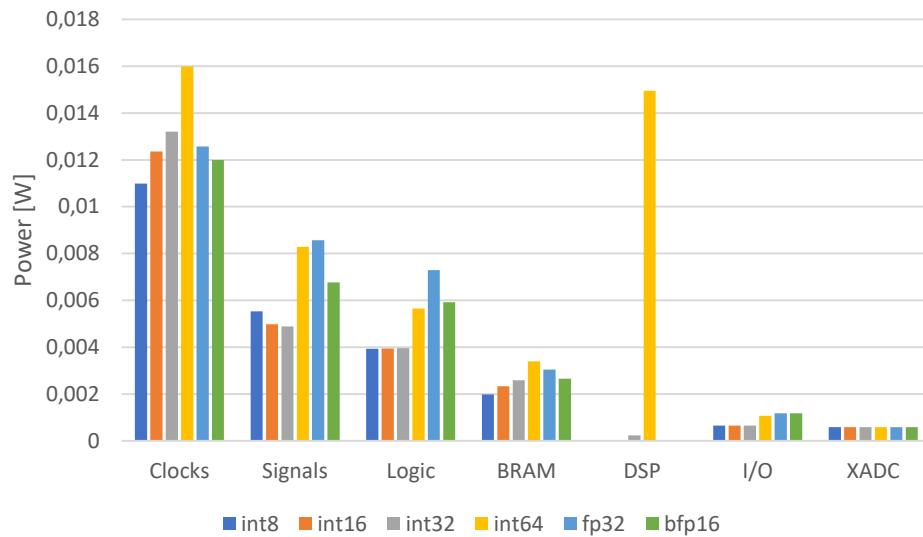
**Figure 5.38:** Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and fp32 PEs

For the bfp16 and fp32 it can be seen that the majority of the power is consumed by the interconnections and the logic. Mainly, because the PEs are implemented in logic.

## 5. Results

---

Until now, the focus has been on how much the single entities and the different type of power were impacting the total power consumption. It is also worth to compare the absolute values for different data precision, as in the Figure 5.39.



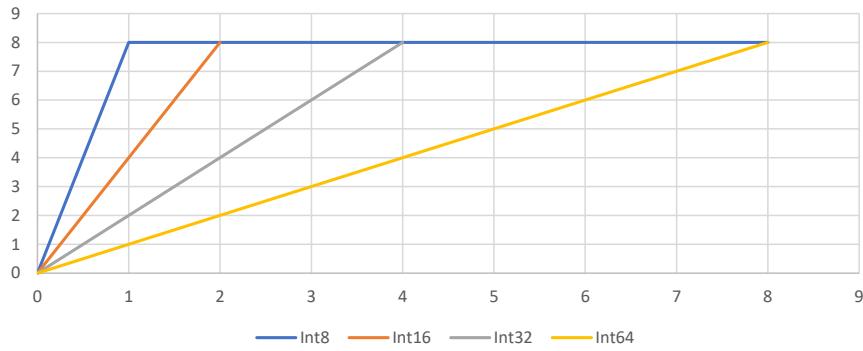
**Figure 5.39:** Comparison of Post Implementation Dynamic Power Consumption per entities in Programmable Logic with a clock frequency of 30 MHz and a MXU 3x3

As it is very well known from literature and it has also been evident from the other figures and observations, the power consumption per entities grows with the increase of the bitwidth (in the case of integer) and complexity (in the case of floating point). The power consumed by the DSPs (entities in which the integer PEs are implemented in) is negligible for the integer 8 and 16 while it starts to grow slowly using the integer 32 but it explodes with the 64 bit PEs. The high utilization of those PEs leads also to a huge impact in their power consumption.

should we add the runtime measurement?

## 5.4 Throughput

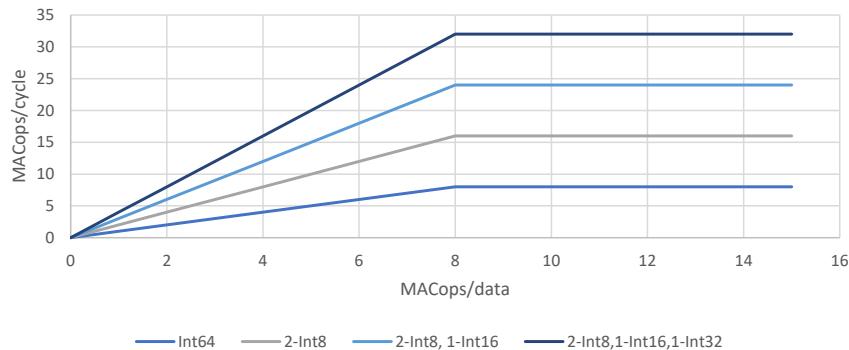
According to the definition, the Throughput is the amount of units of information a system can process in a given time. As said that, for the designed accelerator, it results to be equal to the number of rows into the Matrix Multiplication Unit. Normalizing this value with the clock frequency, it results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



**Figure 5.40:** Roofline model of the accelerator with a MXU size of 8x8

The theoretical throughput given by the roofline model (Figure ) and it is equal to the number of rows in the matrix multiplication unit. The assumption is that enough data are provided to the accelerator in order to have all the Processing Elements working with useful data, if the latter is not meet the throughput goes down. In Figure the different slopes for different data width are representing the different number of internal memory accessess in order to retrieve data for all the Processing Elements.

The throughput can be further increased in the 64-bitwidth configuration of the Processing Elements. As already mentioned, those 64-bit units are able to compute vectorized instructions and therefore increase the number of computation per cycle. However, this comes with the overhead of more memory accesses as it can be appreciated in the .



**Figure 5.41:** Roofline model of the accelerator with a MXU size of 8x8 and vectorized PEs

## 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 50 MHz).

Model	CPU (host) <sup>1</sup>	GPU(host) <sup>2</sup>	CPU(Pynq Z2 board) <sup>3</sup>	CPU(Pynq Z2 board) + accelerator
MNIST	0.3 ms	5.7 ms	2.9 ms	509 ms
Cifar 10	20 ms	22 ms	160 ms	13356 ms

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

Model	CPU (host) <sup>1</sup>	GPU(host) <sup>2</sup>	CPU(Pynq Z2 board) <sup>3</sup>	CPU(Pynq Z2 board) + accelerator
MNIST	0.3 ms	5.7 ms	2.9 ms	503 ms
Cifar 10	20 ms	22 ms	160 ms	13178 ms

**Table 5.2:** Execution Time for different platform and model, integer 16

Model	CPU (host) <sup>1</sup>	GPU(host) <sup>2</sup>	CPU(Pynq Z2 board) <sup>3</sup>	CPU(Pynq Z2 board) + accelerator
MNIST	0.3 ms	5.7 ms	2.9 ms	496.9 ms
Cifar 10	20 ms	22 ms	160 ms	13218 ms

**Table 5.3:** Execution Time for different platform and model, integer 32

Looking at the previous tables, the latency for different data precision it is not changing. This is due to the hardware structure, the Matrix Multiplication Unit is build in such a way that the latency between the one operation and the next one is always of 3 clock cycles (for integer operations).

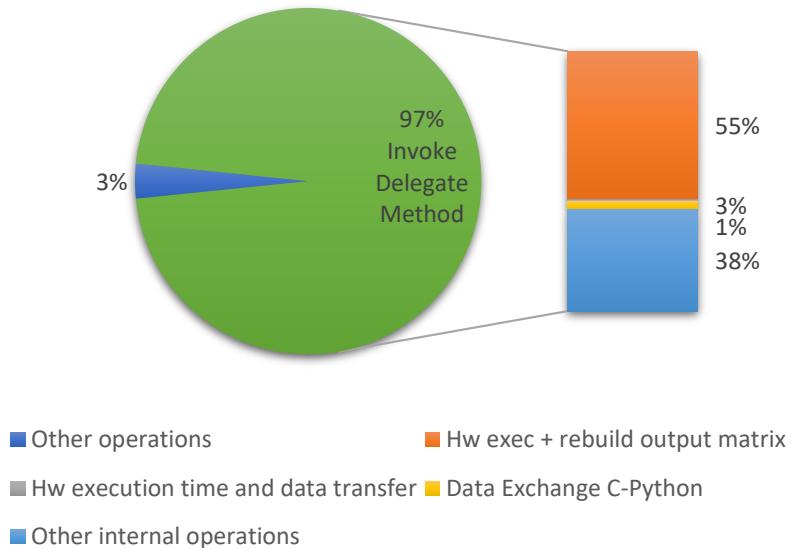
<sup>1</sup>Intel i7-6700HQ, 2.60 Ghz

<sup>2</sup>NVIDIA, GeForce GTX960M, 1.176 Ghz

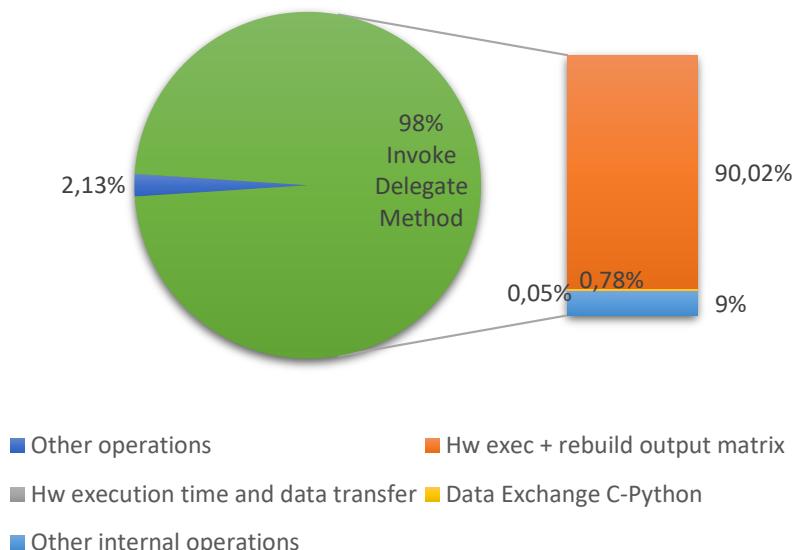
<sup>3</sup>Arm dual-core Cortex-A9, 650 MHz

It is worth to analyze and reason about the increase in the latency in the configuration with the accelerator, since one of the main goal was to reduce the latency time.

Focusing on the following Figure:



**Figure 5.42:** Total Execution time of Invoke method (left) in the configuration with accelerator and MNIST model



**Figure 5.43:** Total Execution time of Invoke method (left) in the configuration with accelerator and Cifar10 model

As it has been mentioned before, the most compute intensive part is always the Convolution operations. Introducing the hardware accelerator and its library comes with several overheads as it can be seen in Figures 5.42 and 5.43:

- Data exchange between C and python: the accelerator library has been developed in Python code with the C interface to Tensorflow Lite. This means that every matrices (input, output and weight) is copied to the python sublayer for further processing. Migrating all the accelerator library into C code will remove this overhead.
- Hardware execution time and rebuilding of output matrix: After every execution of the computation by the accelerator it is parsing the accelerator's output and rebuilding the output matrix accordingly to the current execution indexes. It can be removed preprocessing the model before the deployment, transforming the matrices in a suitable format for the accelerator.
- Hardware execution time and data transfer to the accelerator: This is the actual execution time of the hardware and the data transfer from/to the accelerator. It is also bounded by the fixed internal memory access. It can be reduced by increasing the frequency of Programmable Logic.
- Other internal operations: it includes the time for reshaping the input matrix in a format suitable for the accelerator and the save back from python to C of the output matrix. It can be removed preprocessing the model before the deployment, transforming the matrices in a suitable format for the accelerator. Moreover, the migration towards a complete C implementation is going to remove the overhead due to the saving back of the output matrix.

Taking into account all the previous details and suggestions, the latency of the model can be pushed down to the latency in the solo-CPU execution with the benefits of less power consumption and CPU overload.

## 5.6 Accuracy

The accuracy of inference process in Machine Learning model is how much the prediction is close to the actual value. For example, using the MNIST model, how much is accurate the prediction of a number giving the number as input to the Neural Network.

In the following case, the accuracy for different data width and model will be presented with reference to the actual value, in this case the inference without the hardware accelerator. Moreover, the data provided to the accelerator are bounded by the filter size of the weight, which is always fixed to 3x3 for the used models. Therefore, the MXU for the following comparison has been the standard one, the 8x8.

Model	$\leq \pm 5\%$	$\pm 5\% \div 25\%$	$\pm 25\% \div 50\%$	$\pm 50\% \div 75\%$	$\geq \pm 75\%$
MNIST			x		
Cifar 10					x

**Table 5.4:** Accuracy Output<sup>1</sup> with Convolution on integer 8

Model	$\leq \pm 5\%$	$\pm 5\% \div 25\%$	$\pm 25\% \div 50\%$	$\pm 50\% \div 75\%$	$\geq \pm 75\%$
MNIST				x	
Cifar 10				x	

**Table 5.5:** Accuracy Output<sup>1</sup> with Convolution on integer 16

Model	$\leq \pm 5\%$	$\pm 5\% \div 25\%$	$\pm 25\% \div 50\%$	$\pm 50\% \div 75\%$	$\geq \pm 75\%$
MNIST				x	
Cifar 10				x	

**Table 5.6:** Accuracy Output<sup>1</sup> with Convolution on integer 32

It comes suddenly evident that the output prediction with the accelerator integrated varies of a huge amount from the expected one. One of the reason of the wrong prediction may reside in the input data feed to the model, they are totally random. Being feed with random, and probably unreasonable, data the prediction accuracy has been degraded. Another improvement from the hardware point of view, which could improve the output accuracy, should be to accumulate on the different bitwidth precision, for example compute multiplication on 8 bits integer and accumulate values on 16 bit integer. Moreover, as in every software product, bugs have not been detected but this does not means that the written software is bug-free.

<sup>1</sup>The accuracy is measured percentage(wrt the reference accuracy) of the difference between the reference accuracy (the model's output on the CPU only execution) and the output accuracy of the CPU+ hardware accelerator of the main prediction, the higher one.

## 5. Results

---

# 6

## Conclusion

### 6.1 Discussion

A big portion of inference process for Neural Networks involves massive multiply and add computation, basic operation of tensor convolutions, and across several execution data, especially weight tensors, are reused. As consequence, for speeding-up and reduce the power consumption (especially in mobile devices) of ML models an hardware accelerator has been developed. 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.

The approach of the work has been a hardware/software codesign in order to accommodate the high compute intensive request of Machine Learning, the tensor convolution. Therefore, the hardware core for tensor convolution has been developed from scratch, while the common components, such as memories and bus interface, have been chosen from the avialable ones in the tools. Moving one step at the time above in the abstraction level, the accelerator library has been developed and deployed. In order to accomplish in a fixed time, the core of the library has been developed in Python, which has been interfaced with a C-code template provided from the developers of thee ML-framework used. This has lead to an hybrid library which encapsulates a frozen Python code layer, called from the C-code, the latter is only in charge of retrieving the data and passing them to the Python layer. Again one step above in the abstraction, the ML-framework level is reached. In this level, the most popular ML-framework, TensorFlow, has been choosen. It also offers the possibility of delegate part of the excution graph to coprocessor or GPUs. Moreover, Tensorflow pretrained models have been quantized for different bitwidth and data precision.

It is possible to build a custom hardware accelerator for a specific ML operation and then integrate it into a framework without changing the model nor the framework. The bottom up approach and the delegate class available in Tensorflow has allowed to fully tailor a new class of hardware accelerators which can accomodate different needs (i.e. depending on which part of the model has to be accelerater). As it has been organized, changing the core software in the Python code and the core in the hardware, it can be also used for addressing different models operations.

## 6.2 Future Works

For every human artifacts, there is always work to do. In addition, for Computer Engineering artifacts there is also an importart step which is the software (and in this case also of the hardware) optimization. In particular:

- Software Optimization and migration to a full C code implementation for further reducing the latency.
- A deep software/hardware testing for finding additional bugs.
- Power estimation using the simulation's switching activity in order to obtain a very precise and reliable power consumption.
- Comparison of model execution on different state-of-the-art platforms.

Following the previous reccomandation, the work may arrive to a competitive level such as the one of the GPUs or other hardware platforms.

# 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, NVidia, <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 convolutional networks for rapid-deployment”, **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, Unprecedented acceleration at every scale”, **2020**.
- [22] Nvidia, NVDLA Hardware Architectural Specification, <http://nvdla.org/hw/v1/hwarch.html>.
- [23] Arm, “AMBA® AXI™ and ACE™ Protocol Specification”, **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: extreme 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**.
- [39] Xilinx, “Vivado Design Suite UserGuide: Power Analysis and Optimization”, **2020**.

# A

## Accelerator library

Script for creating library:

```
1 import cffi
2 import sys
3 sys.path.append('/usr/local/lib')
4
5 #
# ##### The Frankenstein , a mix of C and Python
# #####
6 ##### create .so library from PYNQ python code for DTPU accelerator
# #####
7 ##### on board compiling , it requires
# #####
8 ##### to have tensorflow/tensorflow/lite in /usr/include/pythonX.X
# #####
9 ##### from r2.1 branch
# #####
10 #
# #####
11 #
# #####
12 ffibuilder = cffi.FFI()
13
14 ffibuilder.cdef("""
15 extern "Python" {
16     bool destroy_p(void);
17     bool CopyFromBufferHandle_p(void);
18     bool CopyToBufferHandle_p(void);
19     void FreeBufferHandle_p(void);
20     bool SelectDataTypeComputation_p(int);
21     bool Init_p(int, int, int);
22     bool Prepare_p(int);
23     bool Invoke_p(bool, int);
24     void load_overlay(void);
25     bool ResetHardware_p(void);
26     void push_weight_to_heap(void *, int *, int);
27     void push_input_tensor_to_heap(void *, int *, int);
28     void push_output_tensor_to_heap(void *, int *, int);
29     bool print_power_consumption_p(void);
30     bool start_power_consumption(void);
31     void activate_time_probe_p(bool);
32     bool print_python_time_probes(void);
33 };
34     void * tflite_plugin_create_delegate();
```

## A. Accelerator library

---

```
35 void tflite_plugin_destroy_delegate(void * , void * );
36 bool SelectDataTypeComputation(int);
37 bool print_power_consumption();
38 bool measure_power_consumption();
39 bool print_execution_stats();
40 bool activate_time_probe(bool ); """
41
42
43 cpp_file=open("./DTPU_delegate.cpp", "r")
44 ffibuilder.set_source("dtpu_lib", cpp_file.read(), source_extension=".cpp",
45 extra_compile_args=[ '-Wno-unused-result', '-Wsign-compare', '-DNDEBUG',
46 '-g', '-fwrapv', '-O2', '-Wall', '-Wstrict-prototypes',
47 '-g', '-fdebug-prefix-map=/build/python3.5.2=.', '-specs=/usr/share/
48 dpkg/no-pie-compile.specs', '-fstack-protector-strong',
49 '-Wformat', '-Werror=format-security', '-I/usr/local/include', '-L/usr/
50 local/lib'],
51 extra_link_args=[ '-Wl,-Bsymbolic-functions', '-specs=/usr/share/dpkg/
52 no-pie-link.specs',
53 '-Wl,-z,relro', '-specs=/usr/share/dpkg/no-pie-compile.specs', '-D_FORTIFY_SOURCE=2', '-fPIC'],
54 libraries=['pthread', 'expat', 'z', 'dl', 'util', 'm', 'tensorflow'])
55 #if you want to simply access a global variable you just use its name.
56 # However to change its value you need to use the global keyword.
57 python_file=open("./DTPU_delegate.py", "r")
58 ffibuilder.embedding_init_code(python_file.read())
59
60 ffibuilder.compile(target="DTPU_delegate.*", verbose=True)
61
62 cpp_file.close()
63 python_file.close()
```

C++ code of the library:

```

1 // release dependent libraries tensorflow r2.1
2 #include <tensorflow/lite/c/builtin_op_data.h>
3 #include <tensorflow/lite/c/c_api_internal.h>
4 #include <tensorflow/lite/builtin_ops.h>
5 #include <tensorflow/lite/context_util.h>
6 #include <tensorflow/c/c_api.h>
7 #include <vector>
8 #include <time.h>
9 #define DEBUG 1
10
11 static bool destroy_p(void);
12 static bool CopyFromBufferHandle_p(void);
13 static bool CopyToBufferHandle_p(void);
14 static void FreeBufferHandle_p(void);
15 static bool SelectDataTypeComputation_p(int);
16 static bool Init_p(int, int, int);
17 static bool Prepare_p(int);
18 static bool Invoke_p(bool, int);
19 static void load_overlay(void);
20 static bool ResetHardware_p(void);
21 static void push_weight_to_heap(void *, int *, int);
22 static void push_input_tensor_to_heap(void *, int *, int);
23 static void push_output_tensor_to_heap(void *, int *, int);
24 static bool print_power_consumption_p(void);
25 static bool start_power_consumption(void);
26 static void activate_time_probe_p(bool);
27 static bool print_python_time_probes(void);
28
29
30 /*
31 possible operations
32 KTfLiteBuiltinAdd = 0,
33 KTfLiteBuiltinConcatenation = 2,
34 KTfLiteBuiltinConv2d = 3,
35 KTfLiteBuiltinDepthwiseConv2d = 4,
36 KTfLiteBuiltinDepthToSpace = 5,
37 KTfLiteBuiltinFullyConnected = 9,
38 KTfLiteBuiltinMul = 18,
39 KTfLiteBuiltinSub = 41,
40 KTfLiteBuiltinDelegate = 51,
41 KTfLiteBuiltinAddN = 106, struct timespec ts_start, ts_end;
42 */
43
44
45 int bit_width_computation;
46 int NO_FP=-1;
47 bool signed_computation=false;
48 bool only_con2d=false;
```

## A. Accelerator library

---

```
49 // time probes
50 bool time_probe=false;
51 int n_execution=0;
52 double avg_time_delegate;
53 double avg_time_data_exchange;
54
55
56
57 using namespace tflite;
58
59 #ifdef __cplusplus
60 extern "C" {
61 #endif
62 // This is where the execution of the operations or whole graph
63 // happens.
64 // The class below has an empty implementation just as a
65 // guideline
66 // on the structure.
67 class DTPU_delegate {
68 public:
69     // Returns true if my delegate can handle this type of op.
70     static bool SupportedOp(const TfLiteRegistration* registration
71         ) {
72         // from builtin_ops.h
73         #ifdef DEBUG
74             printf("[DEBUG - C]--- Supported Operation of DTPU delegate
75                 class --- \n");
76         #endif
77             switch (registration->builtin_code) {
78                 /*case kTfLiteBuiltinConv2d:
79                     only_conv2d=true;
80                     #ifdef DEBUG
81                         printf("[DEBUG-C]--- Supported operations only 2d
82                             convolution----\n");
83                     #endif
84                     */
85                 case kTfLiteBuiltinDepthwiseConv2d:
86                     #ifdef DEBUG
87                         printf("[DEBUG - C]--Hello world! I can make 2D
88                             convolution and depth wise 2D convolution---\n");
89                     #endif
90                     return true;
91                 default:
92                     return false;
93             }
94     }
95
96     // Any initialization code needed
97     bool Init(TfLiteContext* context, const TfLiteDelegateParams *
```

```

        delegate_params) {
92 #ifdef DEBUG
93     printf("[DEBUG - C]--- Init of DTPU delegate class --- \n");
94 #endif
95
96 #ifdef DEBUG
97     printf("[DEBUG - C]--- Init of DTPU delegate class check
98         if tensors indexes are equal to the ones in the Invoke
99             --- \n");
100    for (int input_index: TfLiteIntArrayView(delegate_params->
101        input_tensors)){
102
103        printf("[DEBUG - C]--- Init of DTPU delegate class getting
104            tensors %d --- \n",input_index);
105    }
106 #endif
107
108 if (time_probe){
109     avg_time_delegate=0.00;
110     avg_time_data_exchange=0.00;
111     n_execution=0;
112 }
113
114 // instantiate buffers and soft reset of accelerator
115 return Init_p(context->tensors_size ,delegate_params->
116     input_tensors->size ,delegate_params->output_tensors->
117     size );
118
119 // Any preparation work needed (e.g. allocate buffers)
120 TfLiteStatus Prepare(TfLiteContext* context , TfLiteNode* node)
121 {
122 #ifdef DEBUG
123     printf("[DEBUG - C]--- Prepare of DTPU delegate class --- \n")
124     ;
125 #endif
126     // initialize , link the buffers accordint to the size of
127     // node data
128     // kTfLiteMmapRo aka weights
129     int num_weight_tensor=0;
130     // set precison check
131     if (NO_FP== -1){
132         printf("ERROR! Need to execute
133             SelectDataTypeComputation function before calling
134             the Tensorflow Interpreter\n");
135         return kTfLiteError ;
136     }
137
138

```



```

170     case 64:
171         #ifdef DEBUG
172             printf("[DEBUG-C]---- kTfLiteInt64 -----\\n");
173         #endif
174             push_weight_to_heap( in_t.data.i64 , in_t.dims->
175                 data , in_t.dims->size);
176             break;
177     }
178     else { // use fp units
179         switch (bit_width_computation){
180             case 16:
181                 if(context->allow_fp32_relax_to_fp16 && NO_FP==3 )
182                     { // NO_FP==3 -> fp active and bfp active
183                     #ifdef DEBUG
184                         printf("[DEBUG-C]---- kTfLitefloat32 relaxed aka
185                             bfp16 -----\\n");
186                     #endif
187                     // remembedr f16 is TfLiteFloat16*
188
189                     /*typedef struct {
190                         uint16_t data;
191                     } TfLiteFloat16;
192                     */
193                     push_weight_to_heap( in_t.data.f16 , in_t.dims->data ,
194                         in_t.dims->size);
195                     }
196                     break;
197             case 32:
198                 #ifdef DEBUG
199                     printf("[DEBUG-C]---- kTfLitefloat32 -----\\n");
200                 #endif
201                     push_weight_to_heap( in_t.data.f , in_t.dims->data ,
202                         in_t.dims->size);
203                     break;
204             default:
205                 printf("[DEBUG-C]---- ERROR! no fp precision defined
206                         -----\\n");
207             break;
208         }
209     }
210     #ifdef DEBUG
211         printf("[DEBUG-C]--- number of weights found= %d \\n",
212             num_weight_tensor);
213     #endif
214     if(Prepare_p(num_weight_tensor)){

```

```
212     return kTfLiteOk ;
213 }
214 return kTfLiteError ;
215 }
216 // Actual running of the delegate subgraph.
217 TfLiteStatus Invoke(TfLiteContext* context , TfLiteNode* node)
218 {
219     struct timespec ts_start ,ts_end ;
220     int curr_input=0;
221 #ifdef DEBUG
222     printf("[DEBUG - C]--- Invoke of DTPU delegate class --- \n"
223         );
224     printf("[DEBUG - C]--- Invoke of DTPU delegate class getting
225         tensors --- \n");
226 #endif
227
228 if (time_probe){
229     if (!timespec_get(&ts_start ,TIME_UTC)){
230         fprintf(stderr , "error during the acquisition of start
231             time !\n");
232         exit(-1);
233     }
234
235 // run inference on the delegate and data transfer to/from
236 // memory/accelerator
237 for (int input_index : TfLiteIntArrayView(node->inputs)){
238     // one of this should be the weight tensor
239 #ifdef DEBUG
240     printf("[DEBUG - C]--- Invoke of DTPU delegate class
241         getting tensors %d--- \n",input_index);
242 #endif
243     TfLiteTensor in_t= context->tensors[input_index];
244     if (!(in_t.allocation_type==kTfLiteMmapRo)){ //cause the
245         weights have been transferred into the Prepare method
246         if (curr_input!=0){
247             curr_input=input_index;
248         }
249         // get dimesion of tensors
250         // push to python sublayer
251         if (!NO_FP){
252             switch(bit_width_computation){
253             default:
254             case 8:
255                 #ifdef DEBUG
256                     if (signed_computation){
257                         printf("[DEBUG-C]--- kTfLiteInt8 ----- \n
```

```

                " );
254         } else{
255             printf("[DEBUG-C]---- kTfLiteUInt8 -----\
256                         n");
257         }
258         #endif
259         if(signed_computation){
260             push_input_tensor_to_heap(in_t.data.int8,in_t.dims->data
261                                     ,in_t.dims->size);
262         } else {
263             push_input_tensor_to_heap(in_t.data.uint8,in_t.dims->
264                                     data,in_t.dims->size);
265         }
266
267         break;
268     case 16:
269         #ifdef DEBUG
270             printf("[DEBUG-C]---- kTfLiteInt16 -----\
271                         n");
272         #endif
273         push_input_tensor_to_heap(in_t.data.i16,in_t.dims
274                                     ->data,in_t.dims->size);
275         break;
276     case 32:
277         #ifdef DEBUG
278             printf("[DEBUG-C]---- kTfLiteInt32 -----\
279                         n");
280         #endif
281         push_input_tensor_to_heap(in_t.data.i32,in_t.dims->
282                                     data,in_t.dims->size);
283         break;
284     case 64:
285         #ifdef DEBUG
286             printf("[DEBUG-C]---- kTfLiteInt64 -----\
287                         n");
288         #endif
289         push_input_tensor_to_heap(in_t.data.i64,in_t.dims->
290                                     data,in_t.dims->size);
291         break;
292     }
293 }
294
295 else { // use fp units
296     switch (bit_width_computation){
297         case 16:
298             if(context->allow_fp32_relax_to_fp16 && NO_FP==3 )
299             { // NO_FP==3 -> fp active and bfp active
300                 #ifdef DEBUG
301                 printf("[DEBUG-C]---- kTfLitefloat32 relaxed aka
302                         bfp16 -----\
303                         n");
304             #endif
305             push_input_tensor_to_heap(in_t.data.f16,in_t.dims

```

```

                ->data , in_t.dims->size ) ;
293         }
294     break;
295 case 32:
296     #ifdef DEBUG
297     printf( "[DEBUG-C]---- kTfLitefloat32 -----\\n" );
298     #endif
299     push_input_tensor_to_heap( in_t.data.f , in_t.dims->
300                               data , in_t.dims->size );
301     break;
302 default:
303     printf( "[DEBUG-C]---- ERROR! no fp precision defined
304             -----\\n" );
305     break;
306 }
307 }
308 }
309 }
310
311 for ( int output_index : TfLiteIntArrayView( node->outputs ) ){
312     auto& out_t= context->tensors[output_index];
313     // get dimesion of tensors
314     // push to python sublayer
315
316     #ifdef DEBUG
317     printf( "[DEBUG - C]--- Invoke of DTPU delegate class
318             getting output tensors %d--- \\n" ,output_index );
319     #endif
320
321     if (!NO_FP){
322         switch( bit_width_computation ){
323             default:
324             case 8:
325                 #ifdef DEBUG
326                     if (signed_computation){
327                         printf( "[DEBUG-C]---- kTfLiteInt8 -----\\n
328                                 " );
329                     } else{
330                         printf( "[DEBUG-C]---- kTfLiteUInt8 -----\\
331                                 n" );
332                     }
333                 #endif
334                 if (signed_computation){
335                     push_output_tensor_to_heap( out_t.data.int8 ,out_t.dims
336                                     ->data ,out_t.dims->size );
337                 } else {

```

```

335         push_output_tensor_to_heap(out_t.data.uint8,out_t.dims
336             ->data,out_t.dims->size);
337     }
338
339     break;
340 case 16:
341     #ifdef DEBUG
342         printf("[DEBUG-C]---- kTfLiteInt16 -----\\n"
343                 );
344     #endif
345     push_output_tensor_to_heap(out_t.data.i16,out_t.
346         dims->data,out_t.dims->size);
347     break;
348 case 32:
349     #ifdef DEBUG
350         printf("[DEBUG-C]---- kTfLiteInt32 -----\\n");
351     #endif
352     push_output_tensor_to_heap(out_t.data.i32,out_t.dims
353         ->data,out_t.dims->size);
354     break;
355 case 64:
356     #ifdef DEBUG
357         printf("[DEBUG-C]---- kTfLiteInt64 -----\\n");
358     #endif
359     push_output_tensor_to_heap(out_t.data.i64,out_t.dims
360         ->data,out_t.dims->size);
361     break;
362 }
363 }
364 else { // use fp units
365     switch (bit_width_computation){
366 case 16:
367     if(context->allow_fp32_relax_to_fp16 && NO_FP==3 )
368         { // NO_FP==3 -> fp active and bfp active
369         #ifdef DEBUG
370         printf("[DEBUG-C]---- kTfLitefloat32 relaxed aka
371                 bfp16 -----\\n");
372         #endif
373         push_output_tensor_to_heap(out_t.data.f16,out_t.
374             dims->data,out_t.dims->size); // a uint16
375         pointer
376         }
377         break;
378 case 32:
379     #ifdef DEBUG
380         printf("[DEBUG-C]---- kTfLitefloat32 -----\\n");
381     #endif
382         push_output_tensor_to_heap(out_t.data.f,out_t.dims
383             ->data,out_t.dims->size);

```

```

374         break;
375     default:
376         printf("[DEBUG-C]---- ERROR! no fp precision defined
377             -----\\n");
378         break;
379     }
380 }
381 }
382 if(time_probe){
383 if(!timespec_get(&ts_end, TIME_UTC)){
384     fprintf(stderr,"erorr during the acquisition of end time
385             !\\n");
386     exit(-1);
387 }
388 // update average and execution time
389 avg_time_data_exchange+=ts_end.tv_sec*1000 + ((double)
390     ts_end.tv_nsec)/1000000 - ts_start.tv_sec*1000 - ((
391     double)ts_start.tv_nsec)/1000000;
392
393 n_execution++;
394 }
395 if(time_probe){
396 if(!timespec_get(&ts_start, TIME_UTC)){
397     fprintf(stderr,"erorr during the acquisition of end time
398             !\\n");
399     exit(-1);
400 }
401 if(Invoke_p(only_con2d, curr_input)){
402     if(time_probe){
403         if(!timespec_get(&ts_end, TIME_UTC)){
404             fprintf(stderr,"erorr during the acquisition of end time
405                 !\\n");
406             exit(-1);
407         }
408         avg_time_delegate+=ts_end.tv_sec*1000 + ((double)ts_end.
409             tv_nsec)/1000000 - ts_start.tv_sec*1000 - ((double)
410             ts_start.tv_nsec)/1000000;
411     }
412     return kTfLiteOk;
413 }
414     return kTfLiteError ;
415 }
416 }
417 };
418

```

```

415 TfLiteStatus SelectDataTypeComputation( int data_type ){
416 #ifdef DEBUG
417 printf("[DEBUG - C]--- SelectDataTypeComputation of DTPU
418     delegate class --- \n");
419 #endif
420 int precision= data_type & 0x000f;
421 signed_computation= ((data_type & 0x00100)>>8)==1 ? true :
422     false;
423
424 NO_FP= (data_type & 0x060)>>5;
425 switch(precision){
426     default:
427         case 1: //INT8
428             bit_width_computation=8;
429             break;
430         case 3: //INT16
431             bit_width_computation=16;
432             break;
433         case 7: //INT32
434             bit_width_computation=32;
435             break;
436         case 15: //INT64
437             bit_width_computation=64;
438             break;
439 }
440 // check compatibility of signed and unsigned
441 if(signed_computation && bit_width_computation!=8){
442     printf("ERROR-> signed/unsigned distinction is only
443         compatible with 8 bit computation");
444     return kTfLiteError;
445 }
446 if(SelectDataTypeComputation_p(data_type) ){
447     return kTfLiteOk;
448 }
449 return kTfLiteError ;
450
451 TfLiteStatus ResetHardware( ){
452 #ifdef DEBUG
453 printf("[DEBUG - C]--- Reset underlaying hardware --- \n");
454 #endif
455 if(ResetHardware_p()){
456     return kTfLiteOk;
457 }
458 return kTfLiteError ;
459 }
460

```

```
461 // Create the TfLiteRegistration for the Kernel node which will
462 // replace
463 TfLiteRegistration GetMyDelegateNodeRegistration() {
464     // This is the registration for the Delegate Node that gets
465     // added to
466     // the TFLite graph instead of the subGraph it replaces.
467     // It is treated as a an OP node. But in our case
468     // Init will initialize the delegate
469     // Invoke will run the delegate graph.
470     // Prepare for preparing the delegate.
471     // Free for any cleaning needed by the delegate.
472 #ifdef DEBUG
473     printf("[DEBUG - C] --- get delegate node registration
474         function ---\n");
475 #endif
476     TfLiteRegistration kernel_registration;
477     kernel_registration.builtin_code = kTfLiteBuiltinDelegate;
478     kernel_registration.custom_name = "DTPU_delegate";
479     kernel_registration.free = [](TfLiteContext* context, void*
480         buffer) -> void {
481         delete reinterpret_cast<DTPU_delegate*>(buffer);
482     };
483     kernel_registration.init = [](TfLiteContext* context, const
484         char* buffer,
485                     size_t) -> void* {
486         // In the node init phase, initialize MyDelegate instance
487         const TfLiteDelegateParams* delegate_params =
488             reinterpret_cast<const TfLiteDelegateParams*>(buffer);
489         DTPU_delegate* my_delegate = new DTPU_delegate;
490         if (!my_delegate->Init(context, delegate_params)) {
491             return nullptr;
492         }
493         return my_delegate;
494     };
495     kernel_registration.invoke = [](TfLiteContext* context,
496                                     TfLiteNode* node) ->
497                                     TfLiteStatus {
498         DTPU_delegate* kernel = reinterpret_cast<DTPU_delegate*>(
499             node->user_data);
500         return kernel->Invoke(context, node);
501     };
502     kernel_registration.prepare = [](TfLiteContext* context,
503                                     TfLiteNode* node) ->
504                                     TfLiteStatus {
505         DTPU_delegate* kernel = reinterpret_cast<DTPU_delegate*>(
506             node->user_data);
507         return kernel->Prepare(context, node);
508     };
509 }
```

```

501     return kernel_registration;
502 }
504
505 // TfLiteDelegate methods
506 // interface to tensorflow runtime
507 TfLiteStatus DelegatePrepare(TfLiteContext* context,
508                             TfLiteDelegate* delegate) {
509     // Claim all nodes that can be evaluated by the delegate and
510     // ask the
511     // framework to update the graph with delegate kernel instead.
512     // Reserve 1 element, since we need first element to be size.
513 #ifdef DEBUG
514     printf("[DEBUG - C] ---- preparing the delegate ----\n");
515 #endif
516     std::vector<int> supported_nodes(1);
517     TfLiteIntArray* plan;
518     TF_LITE_ENSURE_STATUS(context->GetExecutionPlan(context, &plan));
519     TfLiteNode* node;
520     TfLiteRegistration* registration;
521     for (int node_index : tflite::TfLiteIntArrayView(plan)) {
522         TF_LITE_ENSURE_STATUS(context->GetNodeAndRegistration(
523             context, node_index, &node, &registration));
524         if (DTPU_delegate::SupportedOp(registration)) {
525             supported_nodes.push_back(node_index);
526         }
527     }
528     // Set first element to the number of nodes to replace.
529     supported_nodes[0] = supported_nodes.size() - 1;
530     TfLiteRegistration my_delegate_kernel_registration =
531         GetMyDelegateNodeRegistration();
532
533     // This call split the graphs into subgraphs, for subgraphs
534     // that can be
535     // handled by the delegate, it will replace it with a
536     // 'my_delegate_kernel_registration'
537     return context->ReplaceNodeSubsetsWithDelegateKernels(
538         context, my_delegate_kernel_registration,
539         reinterpret_cast<TfLiteIntArray*>(supported_nodes.data()),
540         delegate);
541 }
542
543 void FreeBufferHandle(TfLiteContext* context, TfLiteDelegate*
544                         delegate,
545                         TfLiteBufferHandle* handle) {
546 #ifdef DEBUG
547     printf("[DEBUG - C]--- Do any cleanups---\n");
548 #endif

```

## A. Accelerator library

---

```
544     FreeBufferHandle_p() ;
545 }
546
547
548 TfLiteStatus CopyToBufferHandle(TfLiteContext* context ,
549                                 TfLiteDelegate* delegate ,
550                                 TfLiteBufferHandle buffer_handle
551                                 ,
552                                 TfLiteTensor* tensor) {
553 #ifdef DEBUG
554     printf("[DEBUG - C]--- Copies data from tensor to delegate
555             buffer if needed.----\n");
556 #endif
557     if(CopyToBufferHandle_p()){
558         return kTfLiteOk;
559     }
560     return kTfLiteError;
561 }
562
563 TfLiteStatus CopyFromBufferHandle(TfLiteContext* context ,
564                                   TfLiteDelegate* delegate ,
565                                   TfLiteBufferHandle
566                                   buffer_handle ,
567                                   TfLiteTensor* tensor) {
568 #ifdef DEBUG
569     printf("[DEBUG - C]---Copies the data from delegate buffer
570             into the tensor raw memory----\n");
571 #endif
572     if(CopyFromBufferHandle_p()){
573         return kTfLiteOk;
574     }
575     return kTfLiteError;
576 }
577
578 TfLiteStatus activate_time_probe(bool activate){
579 #ifdef DEBUG
580     printf("[DEBUG-C]---- activating time probes ----\n");
581 #endif
582     if (!time_probe && activate){
583         time_probe=true;
584         #ifdef DEBUG
585             printf("[DEBUG-C]---- activated time probes ----\n");
586         #endif
587         activate_time_probe_p(activate);
588     } else{
589         printf("ATTENTION! Time probes are not active\n");
590     }
591     return kTfLiteOk;
592 }
```

```
589 }
590 }
591
592
593 TfLiteStatus print_execution_stats(){
594     #ifdef DEBUG
595         printf("[DEBUG - C]---- printing time probes of the library
596             ----\n");
597     #endif
598     printf("If you are seeing too many zeros you probably did
599             not set the time probes variable to true!\n");
600
601     // print c time probes
602     printf("Overall time of delegate invoke: %3f [ms]\n",
603             avg_time_delegate/n_execution);
604     printf("Data exchange between interfaces (C->Python->C) : %3f
605             [ms]\n",avg_time_data_exchange/n_execution);
606
607     // print python time probes
608     if(print_python_time_probes()) {
609         return kTfLiteOk;
610     }
611     return kTfLiteError;
612 }
613
614 TfLiteStatus measure_power_consumption() {
615     #ifdef DEBUG
616         printf("[DEBUG - C]---Measuring power consumption of the
617             accelerator during invoke ----\n");
618     #endif
619     if(start_power_consumption()) {
620         return kTfLiteOk;
621     }
622     return kTfLiteError;
623 }
624
625 TfLiteStatus print_power_consumption(){
626     #ifdef DEBUG
627         printf("[DEBUG - C]---Printing power consumption of the
628             accelerator during invoke ----\n");
629     #endif
630     if(print_power_consumption_p()) {
631         return kTfLiteOk;
632     }
633     return kTfLiteError;
634 }
```

```
632 // instantiate the delegate , it returns null if there is an
633 // error
634 TfLiteDelegate * tflite_plugin_create_delegate()
635 //char** argv , char** argv2 , size_t argc , void (*report_error)(
636 //    const char * ) )
637 {
638     TfLiteDelegate* delegate = new TfLiteDelegate;
639
640     delegate->data_ = nullptr;
641     delegate->flags = kTfLiteDelegateFlagsNone;
642     delegate->Prepare = &DelegatePrepare;
643     // This cannot be null.
644     delegate->CopyFromBufferHandle = &CopyFromBufferHandle;
645     // This can be null.
646     delegate->CopyToBufferHandle = &CopyToBufferHandle;
647     // This can be null.
648     delegate->FreeBufferHandle = &FreeBufferHandle;
649     // load overlay
650     load_overlay();
651 #ifdef DEBUG
652     printf("[DEBUG - C] ---the delegate method of DTPU is born for
653         TensorFlow %s---\n",TF_Version());
654 #endif
655     return delegate;
656 }
657
658
659 void tflite_plugin_destroy_delegate(void * delegate_op , void *
660 argtypes) {
661 // destroy the delegate
662 TfLiteDelegate * delegate= (TfLiteDelegate *) delegate_op;
663 #ifdef DEBUG
664     printf("[DEBUG - C]-----cleaning memory --> callback of python
665         function---\n");
666 #endif
667     if (!destroy_p()) {
668         printf("ERROR IN FREEING BUFFERS!");
669     }
670     // free(argtypes);
671     free(delegate);
672 }
673 #ifdef __cplusplus
674 } // extern "C"
675 #endif
```

Frozen python code in the accelerator library:

```
1 from dtpu_lib import ffi
2 from pynq import Overlay
3 from pynq import allocate
4 from pynq import MMIO
5 from pynq import Xlnk
6 from pynq.lib import dma
7 import numpy as np
8 import math
9 import _thread
10 import sys
11 import time
12 import struct # see https://docs.python.org/3/library/struct.html#struct-examples
13 _DEBUG_PRINT=True
14 _TIME_PROBES=False
15 #####
16 ##### memory map of xadc #####
17 #####
18 C_BASEADDRESS=0x43C10000 #
19 SRR= 0x0 # w software reset register
20 SR= 0x04 # r status register
21 AOSR= 0x08 # r allarm output status register
22 CONVSTR= 0x0C # w Bit[0] = ADC convert start register (3) Bit[1]
    = Enable temperature update logic Bit[17:2] = Wait cycle for
    temperature update
23 SYSMONRR=0x10 # w xadc hard macro reset register
24 GIER=0x5C # rw global interrupt enable register
25 IPISR=0x60 # r toggle on write ip interrupt status register
26 IPIER=0x68 # rw ip interrupt enable register
27 TEMPERATURE=0x200 # The 12-bit Most Significant Bit (MSB)
    justified result of on-device temperature measurement is
    stored in this register
28 VCC_INT=0x204 # The 12-bit MSB justified result of on-device V
    CCINT supply monitor measurement is stored in this register.
29 VCC_AUX=0x208 # The 12-bit MSB justified result of on-device V
    CCAUX Data supply monitor measurement is stored in this
    register
30 VP_VN=0x20C # rw When read: The 12-bit MSB justified result of A
    /D conversion on the dedicated analog input channel (Vp/Vn)
    is stored in this register. When written: Write to this
    register resets the XADC hard macro
31 VREF_P=0x210 # r The 12-bit MSB justified result of A/D
    conversion on the reference input V REF_P is stored in this
    register.
32 VREF_N= 0x214 #r The 12-bit MSB justified result of A/D
    conversion on the reference input V REF_N is stored in this
    register.
33 VCC_BRAM= 0x218 # r The 12-bit MSB justified result of A/D
```

```
conversion on the reference input V BRAM is stored in this
register
34 SUPPLY_A_OFFSET=0x220 # r The calibration coefficient for the
    supply sensor offset of ADC A is stored in this register
35 ADC_A_OFFSET= 0x224 # r The calibration coefficient for the ADC
    A offset calibration is stored in this register.
36 ADC_A_GAIN_ERR=0x228 # r The calibration coefficient for the
    gain error of ADC A is stored in this register.
37 DEV_CORE_SUPPLY=0x234 #r The VCCINT of PSS core supply.
    Present only on Zynq-7000 devices.
38 DEV_AUX_CORE_SUPPLY=0x238 # r The VCCAUX of PSS core supply.
    Present only on Zynq-7000 devices.
39 DEV_CORE_MEM_SUPPLY=0x23C # r The VCCMEM of PSS core supply.
    Present only on Zynq-7000 devices
40 # v axux p/n
41 V_AUX_0= 0x240 #r The 12-bit MSB justified result of A/D
    conversion on the auxiliary analog input 0 is stored in this
    register.
42 V_AUX_1= 0x244 #r
43 V_AUX_2= 0x248 #r
44 V_AUX_3= 0x24C #r
45 V_AUX_4= 0x250 #r
46 V_AUX_5= 0x254 #r
47 V_AUX_6= 0x258 #r
48 V_AUX_7= 0x25C #r
49 V_AUX_8= 0x260 #r
50 V_AUX_9= 0x264 #r
51 V_AUX_10= 0x268 #r
52 V_AUX_11= 0x26C #r
53 V_AUX_12= 0x270 #r
54 V_AUX_13= 0x274 #r
55 V_AUX_14= 0x278 #r
56 V_AUX_15= 0x27C #r
57 ## value of 12 bit msb
58 MAX_TMP= 0x280 # r
59 MAX_VCC_INT= 0x284 # r
60 MAX_VCC_AUX= 0x288 # r
61 MAV_V_BRAM= 0x28C # r
62 MIN_TMP= 0x290 # r
63 MIN_VCC_INT= 0x294 # r
64 MIN_VCC_AUX= 0x298 # r
65 MIN_V_BRAM=0x29C # r
66 MAX_VCC_PINT= 0x2A0 # r
67 MAX_VCC_PAUX= 0x2A4 # r
68 MAX_VCC_DDRO= 0x2A8 # r
69 MIN_VCC_PINT= 0x2b0 # r
70 MIN_VCC_PAUX= 0x2b4 # r
71 MIN_VCC_DDRO= 0x2b8 # r
72 SUPPLY_B_OFFSET= 0x2C0 # r The calibration coefficient for the
```

```

    supply sensor offset of ADC A is stored in this register
73 ADC_B_OFFSET= 0x2C4 # r The calibration coefficient for the ADC
    A offset calibration is stored in this register.
74 ADC_B_GAIN_ERR= 0x2C8 # r The calibration coefficient for the
    gain error of ADC A is stored in this register.
75 FLAGS=0x2FC # The 16-bit register gives general status
    information of ALARM, Over Temperature (OT), Disable XADC
    information. Whether the XADC is using the internal reference
    voltage or external reference voltage is also p
76 CONF_REG_0=0x300 # rw
77 CONF_REG_1=0x304 # rw
78 CONF_REG_2=0x308 # rw
79 SEQ_REG_0= 0x320 # r/w adc channel selection
80 SEQ_REG_1= 0x324 # r/w adc channel selection
81 SEQ_REG_2= 0x328 # r/w adc channel average enable
82 SEQ_REG_3= 0x32C # r/w adc channel average enable
83 SEQ_REG_4= 0x330 # r/w adc channel analog input mode
84 SEQ_REG_5= 0x334 # r/w adc channel analog input mode
85 SEQ_REG_6= 0x338 # r/w adc channel acquistion
86 SEQ_REG_7= 0x33C # r/w adc channel acquistion
87 ALLARM_THRESHOLD_0= 0x340 #rw The 12bit MSB justified alarm
    threshold register 0 Temperature Upper
88 ALLARM_THRESHOLD_1= 0x344 #rw he 12bit MSB justified alarm
    threshold register 1 V CCINT Upper
89 ALLARM_THRESHOLD_2= 0x348 #rw The 12bit MSB justified alarm
    threshold register 2 V CCAUX Upper
90 ALLARM_THRESHOLD_3= 0x34C #rw the 12bit MSB justified alarm
    threshold register 3 T Upper
91 ALLARM_THRESHOLD_4= 0x350 #rw the 12bit MSB justified alarm
    threshold register 4 Temperature Lower
92 ALLARM_THRESHOLD_5= 0x354 #rw the 12bit MSB justified alarm
    threshold register 5 V CCINT Lower
93 ALLARM_THRESHOLD_6= 0x358 #rw The 12bit MSB justified alarm
    threshold register 6 V CCAUX Lower
94 ALLARM_THRESHOLD_7= 0x35C # rw The 12bit MSB justified alarm
    threshold register 7 OT Lower
95 ALLARM_THRESHOLD_8= 0x360 # rw The 12bit MSB justified alarm
    threshold register 8 VBRAM Upper
96 ALLARM_THRESHOLD_9= 0x364 # rw The 12bit MSB justified alarm
    threshold register 9 V CCPint Upper This register is only on
    Zynq-7000 devices.
97 ALLARM_THRESHOLD_10= 0x368 # rw The 12bit MSB justified alarm
    threshold register 10 V CCPaux Upper This register is only
    on Zynq-7000 devices
98 ALLARM_THRESHOLD_11= 0x36C # rw The 12bit MSB justified alarm
    threshold register 11 CCDDRO Upper This register is only on
    Zynq-7000 devic
99 ALLARM_THRESHOLD_12= 0x370 # rw he 12bit MSB justified alarm
    threshold register 12 VBRAM Lower

```

```
100 ALLARM_THRESHOLD_13= 0x374 # rw The 12Bit MSB justified alarm
    threshold register 13 V CCPint Lower This register is only on
    Zynq-7000 devices
101 ALLARM_THRESHOLD_14= 0x378 # rw The 12bit MSB justified alarm
    threshold register 14 V CCPaux Lower This register is only on
    Zynq-7000 devices
102 ALLARM_THRESHOLD_15= 0x37C # rw he 12bit MSB justified alarm
    threshold register 15 v CCDDRO Lower This register is only on
    Zynq-7000 devices
103 ##### MEMORY MAP of acceleratro #####
104 ##### MEMORY MAP of acceleratro #####
105 ##### MEMORY MAP of acceleratro #####
106 BASE_ADDRESS_ACCELERATOR=0x43C00000
107 ADDRESS_RANGE_ACCELERATOR=0x10000
108 # address reg offset
109 CTRL =0x0000
110 STATUS =0x0004
111 IARG_RQT_EN =0x0010
112 OARG_RQT_EN =0x0014
113 CMD =0x0028
114 OARG_LENGTH_MODE =0x003C
115 ISCALAR_FIFO_RST =0x0040
116 OSCALAR_FIFO_RST =0x0044
117 ISCALAR_RQT_EN =0x0048
118 OSCALAR_RQT_EN =0x004C
119 ISCALAR0_DATA =0x0080
120 ISCALAR1_DATA =0x0084
121 ISCALAR2_DATA =0x0088
122 ISCALAR3_DATA =0x008C
123 ISCALAR4_DATA =0x0090
124 ISCALAR5_DATA =0x0094
125 ISCALAR6_DATA =0x0098
126 ISCALAR7_DATA =0x009C
127 ISCALAR8_DATA =0x00A0
128 ISCALAR9_DATA =0x00A4
129 ISCALAR10_DATA=0x00A8
130 ISCALAR11_DATA =0x00AC
131 ISCALAR12_DATA =0x00B0
132 ISCALAR13_DATA =0x00B4
133 ISCALAR14_DATA =0x00B8
134 ISCALAR15_DATA =0x00BC
135 OSCALAR0_DATA =0x00C0
136 OSCALAR1_DATA =0x00C4
137 OSCALAR2_DATA =0x00C8
138 OSCALAR3_DATA =0x00CC
139 OSCALAR4_DATA =0x00D0
140 OSCALAR5_DATA =0x00D4
141 OSCALAR6_DATA =0x00D8
142 OSCALAR7_DATA =0x00DC
```

```
143 IARG0_STATUS =0x0100
144 IARG1_STATUS =0x0104
145 IARG2_STATUS =0x0108
146 IARG3_STATUS =0x010C
147 IARG4_STATUS =0x0110
148 IARG5_STATUS =0x0114
149 IARG6_STATUS =0x0118
150 IARG7_STATUS =0x011C
151 OARG0_STATUS =0x0140
152 OARG1_STATUS =0x0144
153 OARG2_STATUS =0x0148
154 OARG3_STATUS =0x014C
155 OARG4_STATUS =0x0150
156 OARG5_STATUS =0x0154
157 OARG6_STATUS =0x0158
158 OARG7_STATUS =0x015C
159 ISCALAR0_STATUS =0x0180
160 ISCALAR1_STATUS =0x0184
161 ISCALAR2_STATUS =0x0188
162 ISCALAR3_STATUS =0x018C
163 ISCALAR4_STATUS =0x0190
164 ISCALAR5_STATUS =0x0194
165 ISCALAR6_STATUS =0x0198
166 ISCALAR7_STATUS =0x019C
167 ISCALAR8_STATUS =0x01A0
168 ISCALAR9_STATUS =0x01A4
169 ISCALAR10_STATUS =0x01A8
170 ISCALAR11_STATUS =0x01AC
171 ISCALAR12_STATUS =0x01B0
172 ISCALAR13_STATUS =0x01B4
173 ISCALAR14_STATUS =0x01B8
174 ISCALAR15_STATUS =0x01BC
175 OSCALAR0_STATUS =0x01C0
176 OSCALAR1_STATUS =0x01C4
177 OSCALAR2_STATUS =0x01C8
178 OSCALAR3_STATUS =0x01CC
179 OSCALAR4_STATUS =0x01D0
180 OSCALAR5_STATUS =0x01D4
181 OSCALAR6_STATUS =0x01D8
182 OSCALAR7_STATUS =0x01DC
183 OSCALAR8_STATUS =0x01E0
184 OSCALAR9_STATUS =0x01E4
185 OSCALAR10_STATUS =0x01E8
186 OSCALAR11_STATUS =0x01EC
187 OSCALAR12_STATUS =0x01F0
188 OSCALAR13_STATUS =0x01F4
189 OSCALAR14_STATUS =0x01F8
190 OSCALAR15_STATUS =0x01FC
191 OARG0_LENGTH =0x0200
```

## A. Accelerator library

---

```
192 OARG1_LENGTH =0x0204
193 OARG2_LENGTH =0x0208
194 OARG3_LENGTH =0x020C
195 OARG4_LENGTH =0x0210
196 OARG5_LENGTH =0x0214
197 OARG6_LENGTH =0x0218
198 OARG7_LENGTH =0x021C
199 OARG0_TDEST =0x0240
200 OARG1_TDEST =0x0244
201 OARG2_TDEST =0x0248
202 OARG3_TDEST =0x024C
203 OARG4_TDEST =0x0250
204 OARG5_TDEST =0x0254
205 OARG6_TDEST =0x0258
206 OARG7_TDEST =0x025C
207 ##### CSR DEFINITIONS #####
208 ##### MEMORY MAP #####
209 ##### bitwidth 8 #####
210 ##### see csr_definition.vh #####
211 ##### ARITHMETIC_PRECISION=0 #####
212 ##### FP_MODE=1 #####
213 BATCH_SIZE=2 # aka active rows
214 TRANSPARENT_DELAY_REGISTER=3
215 DEBUG=4
216 TEST_OPTIONS=5
217 ACTIVATE_CHAIN=0x1
218 INT8=0x1
219 INT16=0X3
220 INT32=0x7
221 INT64=0xF
222 # precision of fp computation is tuned using the
223 # integer precision
224 ACTIVE_FP=1
225 ACTIVE_BFP=0x03
226 ROUNDING=0x00
227 NO_FP=0x00
228 SIGNED=0x1
229 NO_SIGNED=0x0
230 WMEM_STARTING_ADDRESS=0 #32 MSB
231 #####
232 #### accelerator adapter command #####
233 #####
234 CMD_UPDATE_IN_ARG=0x0
235 CMD_UPDATE_OUT_ARG=0x1
236 CMD_EXECUTE_STEP=0x2
237 CMD_EXECUTE_CONTINOUS=0x4
238 CMD_STOP_EXECUTE_CONTINOUS=0x5
```

```

241 #####
242 BASE_ADDRESS_INTC=0x40800000
243 ADDRESS_RANGE_INTC=0x10000
244 BASE_ADDRESS_DMA_INFIFO=0x40400000
245 ADDRESS_RANGE_DMA_INFIFO=0x10000
246 BASE_ADDRESS_DMA_WM=0x40410000
247 ADDRESS_RANGE_DMA_WM=0x10000
248 accelerator=None
249 infifo_buffer_transfer=None
250 output_fifo_buffer=None
251 weight_buffer=None
252 csr_buffer=None
253 overlay=None
254 driver_csr=None
255 driver_wm=None
256 driver_fifo_in=None
257 driver_fifo_out=None
258 #####
259 ### DESIGN DEPENDENT DEFINITION #####
260 #####
261 WMEM_SIZE=16384 # 1Mbytes
262 CSRMEM_SIZE=1024
263 INFIFO_SIZE=2048 #1Kbytes
264 OUTFIFO_SIZE=2048 #1Kbytes
265 ROWS=0
266 COLUMNS=0
267 DATAWIDTH=64
268 BUFFER_DEPTH=2
269 output_size=0
270 input_size=0
271 tot_size_weight=0
272 tot_size_input=0
273 tot_size_output=0
274 curr_data_precision=INT8
275 curr_bitwidth_data_computation=8
276 PACK_TYPE="b" # default is 1 byte signed for integer lower case
    -> signed upper case-> unsigned
277 DTYPE_NP=np.uint8
278 FP=False
279 BFP=False
280 size_tot=0
281 num_weight=0
282 global_iteration=1 ## at least one execution of the tensor
    accelerator
283 global_iteration_shift_wm=[]
284 weight_tensors=[]
285 input_tensors=[]
286 output_tensors=[]
287 output_tensors_p=[]

```

```
288 weight_buffer_multiple=[]
289 index_wm=0
290 class Tensor:
291     def __init__(self,data,tot_dim,size_l):
292         self.tot_dim=tot_dim
293         self.data=data
294         self.size_l=size_l
295     filter_height=0
296     filter_width=0
297 ##### time probes #####
298 ##### time probes #####
299 ##### time probes #####
300 avg_hw_execution=0.0
301 n_execution=0
302 avg_hw_execution_internal=0.00
303 n_execution_internal=0
304 #####
305 ##### XADC #####
306 #####
307 xadc_mon=None
308 #####
309 ##### Retrieve and display power consumption #####
310 ##### Supply sensor: Vccint,Vccaux,Vccbram #####
311 ##### Vccpint, Vccpaux,Vcc0ddr #####
312 ##### Nominal values of resistances and Vcc #####
313 #####
314 # from vivado report power
315 # [V]
316 vcc_pl_int_nom=1.00
317 vcc_pl_aux_nom=1.80
318 vcc_pl_bram_nom= 1.00
319 vcc_ps_int_nom=1.80
320 vcc_ps_aux_nom=1.80
321 vcc_ddr_nom=1.50
322 # equivalent series resistances of capacitor -> worst case
323 # [omh]
324 r_pl_int=225
325 r_pl_aux=300
326 r_pl_bram=225
327 r_ps_int=225
328 r_ps_aux=400
329 r_ddr= 0.005
330 n_sample=1
331 ps_power=0
332 pl_power=0
333 mem_power=0
334 ps_power_max=sys.float_info.min
335 pl_power_max=sys.float_info.min
336 mem_power_max=sys.float_info.min
```

```

337 ps_power_min=sys.float_info.max
338 pl_power_min=sys.float_info.max
339 mem_power_min=sys.float_info.max
340 tmp_max=sys.float_info.min
341 tmp_min=sys.float_info.max
342 tmp_avg=0.00
343
344 def sample_power( threadName , delay ):
345     global ps_power
346     global pl_power
347     global mem_power
348     global n_sample
349     global xadc_mon
350     global vcc_ps_aux_nom
351     global ps_power_max
352     global pl_power_max
353     global mem_power_max
354     global ps_power_min
355     global pl_power_min
356     global mem_power_min
357     global tmp_max
358     global tmp_min
359     global tmp_avg
360     while True:
361         time.sleep(0.8/1000)
362         vcc_pl_int=( xadc_mon.read(VCC_INT) & 0x0000FFF0) >> 4
363         vcc_pl_int= (vcc_pl_int* vcc_ps_aux_nom) / 4096
364         vcc_pl_aux=( xadc_mon.read(VCC_AUX) & 0x0000FFF0) >> 4
365         vcc_pl_aux= (vcc_pl_aux* vcc_ps_aux_nom) / 4096
366         vcc_pl_bram= ( xadc_mon.read(VCC_BRAM) & 0x0000FFF0) >> 4
367         vcc_pl_bram= (vcc_pl_bram* vcc_ps_aux_nom) / 4096
368         vcc_ps_int= ( xadc_mon.read(DEV_CORE_SUPPLY) & 0x0000FFF0)
369             >> 4
370         vcc_ps_int= (vcc_ps_int* vcc_ps_aux_nom) / 4096
371         vcc_ps_aux=( xadc_mon.read(DEV_AUX_CORE_SUPPLY) & 0x0000FFF0)
372             ) >> 4
373         vcc_ps_aux= (vcc_ps_aux* vcc_ps_aux_nom) / 4096
374         vcc_ddr= ( xadc_mon.read(DEV_CORE_MEM_SUPPLY) & 0x0000FFF0)
375             >> 4
376         vcc_ddr= (vcc_ddr* 3) / 4096
377         n_sample+=1
378         ps_power_i=((vcc_ps_int_nom-vcc_ps_int)/r_ps_int)*
379             vcc_ps_int_nom + ((vcc_ps_aux_nom-vcc_ps_aux)/r_ps_aux)*
380                 vcc_ps_aux_nom
381         pl_power_i= ((vcc_pl_int_nom-vcc_pl_int)/r_pl_int)*
382             vcc_pl_int_nom + ((vcc_pl_aux_nom-vcc_pl_aux)/r_pl_aux)*
383                 vcc_pl_aux_nom + ((vcc_pl_bram_nom-vcc_pl_bram)/r_pl_bram)
384                     )*vcc_pl_bram_nom
385         mem_power_i=((vcc_ddr-vcc_ddr_nom)/r_ddr)*vcc_ddr

```

```
378     ## update max
379     if pl_power_i > pl_power_max:
380         pl_power_max=pl_power_i
381     if ps_power_i > ps_power_max:
382         ps_power_max=ps_power_i
383     if mem_power_i > mem_power_max:
384         mem_power_max=mem_power_i
385     #update min
386     if pl_power_i < pl_power_min:
387         pl_power_min=pl_power_i
388     if ps_power_i < ps_power_min:
389         ps_power_min=ps_power_i
390     if mem_power_i < mem_power_min:
391         mem_power_min=mem_power_i
392     ## update values for the averages
393     ps_power+=ps_power_i
394     pl_power+=pl_power_i
395     mem_power+=mem_power_i
396     # temperature
397     tmp=( xadc_mon.read(TEMPERATURE) & 0x0000FFF0) >> 4
398     tmp=(tmp* 503.975)/4096 - 273.15
399     ## update max
400     if tmp > tmp_max:
401         tmp_max=tmp
402     ## update min
403     if tmp < tmp_min:
404         tmp_min=tmp
405     tmp_avg+=tmp
406
407 ##### LOAD DESIGN #####
408 ##### LOAD DESIGN #####
409 ##### LOAD DESIGN #####
410 @ffi.def_extern()
411 def load_overlay():
412     global accelerator
413     global overlay
414     global xadc_mon
415     global ROWS
416     global COLUMNS
417     global ps_power
418     global pl_power
419     global mem_power
420     global ps_power_max
421     global pl_power_max
422     global mem_power_max
423     global ps_power_min
424     global pl_power_min
425     global mem_power_min
426     global tmp_max
```

```

427 global tmp_min
428 global tmp_avg
429 ## modify this part for choosing a different overlay and
430     recompile the library
431 f_clk="30mhz"
432 datawidth="only_integer8"
433 mxu_size="mxu_8x8"
434 ROWS=8
435 COLUMNS=8
436 print("Hardware design space points",f_clk," ", " ", mxu_size,
437       " ", datawidth)
438 overlay = Overlay("/home/xilinx/dtpu_configurations/" +
439                     datawidth+"/"+f_clk+"/"+ mxu_size+"/pynqz2.bit") # tcl is
440                     also parsed
441 overlay.download() # Explicitly download bitstream to PL
442 if overlay.is_loaded():
443     # Checks if a bitstream is loaded
444     if _DEBUG_PRINT: print("[DEBUG-PYTHON] -----overlay is loaded
445                         -----")
446 else :
447     if _DEBUG_PRINT: print("[DEBUG-PYTHON] ----- overlay is not
448                         loaded-----")
449     exit(-1)
450 if overlay.monitor is not None:
451     xadc_mon=overlay.monitor.xadc_wiz_0_0
452     xadc_mon.write(SRR,0x0000000A) # reset
453 else:
454     print("ERROR NO XADC")
455 if overlay.dtpu is not None:
456     accelerator=overlay.dtpu.axis_accelerator_ada
457 else:
458     print("ERROR NO ACCELERATOR")
459     exit(-1)
460 overlay.reset()
461 # clean power variable
462 n_sample=1
463 ps_power=0
464 pl_power=0
465 mem_power=0
466 ps_power_max=sys.float_info.min
467 pl_power_max=sys.float_info.min
468 mem_power_max=sys.float_info.min
469 ps_power_min=sys.float_info.max
470 pl_power_min=sys.float_info.max
471 mem_power_min=sys.float_info.max
472 tmp_max=sys.float_info.min
473 tmp_min=sys.float_info.max
474 tmp_avg=0.00
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512

```

## A. Accelerator library

---

```
470
471 @ffi.def_extern()
472 def Init_p(tot_tensors, input_tens_size, output_tens_size):
473     global accelerator
474     global overlay
475     global size_tot
476     global input_size
477     global output_size
478     global avg_hw_execution
479     global n_execution
480     global avg_hw_execution_internal
481     global n_execution_internal
482     global tmp_avg
483     if _DEBUG_PRINT: print("[DEBUG - PYTHON] --- Init p function")
484     ## soft reset and accelerator configuration
485     accelerator.write(CTRL,0x00000001)
486     accelerator.write(CTRL,0x00000000)
487     accelerator.write(IARG_RQT_EN,0x00000007) ## all data
488     # avialable csr, weights and data
489     accelerator.write(OARG_LENGTH_MODE,0x00000001) # software mode
490     accelerator.write(OARG0_LENGTH,OUTFIFO_SIZE) # size outfifo
491     accelerator.write(ISCALAR_RQT_EN,0) # NO input SCALAR
492     accelerator.write(OSCALAR_RQT_EN,0) # no output scalar
493     accelerator.write(OARG0_TDEST,0) # only one output
494     size_tot=tot_tensors
495     if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- total tensors",
496                         size_tot,"---")
497     input_size=input_tens_size
498     if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- int tensors",
499                     input_size,"---")
500     output_size=output_tens_size
501     if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- out tensors",
502                         output_tens_size,"---")
503     n_execution=0
504     avg_hw_execution=0.00
505     avg_hw_execution_internal=0.00
506     n_execution_internal=0
507     tmp_avg=0.00
508     return True
509
510
511 @ffi.def_extern()
512 def SelectDataTypeComputation_p(data_type):
513     global curr_data_precision
514     global curr_bitwidth_data_computation
515     global PACK_TYPE
516     global FP
517     global BFP
```

```

514 global DTTYPE_NP
515 if _DEBUG_PRINT: print("[DEBUG - PYTHON] ---")
516     SelectDataTypeComputation DTPU class ---")
517 if data_type !=0:
518     #case switch
519     if ((data_type)&0x00000f)==INT8:
520         curr_data_precision=INT8
521         curr_bitwidth_data_computation=8
522         if (data_type&0x00100)==SIGNED:
523             PACK_TYPE="b"
524             DTTYPE_NP=np.int8
525         else:
526             PACK_TYPE="B"
527             DTTYPE_NP=np.uint8
528         elif ((data_type)&0x00000f)==INT16:
529             curr_data_precision=INT16
530             curr_bitwidth_data_computation=16
531             if (data_type&0x00100)==SIGNED:
532                 PACK_TYPE="h"
533                 DTTYPE_NP=np.int16
534             else:
535                 PACK_TYPE="H"
536                 DTTYPE_NP=np.uint16
537             elif ((data_type)&0x00000f)==INT32:
538                 curr_data_precision=INT32
539                 curr_bitwidth_data_computation=32
540                 if (data_type&0x00100)==SIGNED:
541                     PACK_TYPE="i"
542                     DTTYPE_NP=np.int32
543                 else:
544                     PACK_TYPE="I"
545                     DTTYPE_NP=np.uint32
546                 elif ((data_type)&0x00000f)==INT64:
547                     curr_data_precision=INT64
548                     curr_bitwidth_data_computation=64
549                     if (data_type&0x00100)==SIGNED:
550                         PACK_TYPE="q"
551                         DTTYPE_NP=np.int64
552                     else:
553                         PACK_TYPE="Q"
554                         DTTYPE_NP=np.uint64
555                 else:
556                     print("ERROR PYTHON! Setting the Data type of computation"
557                         )
558 # floating point check
559 if ((data_type & 0x000060)>> 5)== ACTIVE_FP:
560     FP=True
561     BFP=False
562     PACK_TYPE="f"

```

```

561     DTTYPE_NP=np.float32
562     elif ((data_type & 0x000060)>> 5)== ACTIVE_BFP:
563         FP=True
564         BFP=True
565         PACK_TYPE="e"
566         DTTYPE_NP=np.uint16 ## accroding to tensorflow bfp16
567             representation
568     else:
569         FP=False
570         BFP=False
571     else:
572         curr_data_precision=INT8
573         curr_bitwidth_data_computation=8
574         FP=False
575         BFP=False
576     if _DEBUG_PRINT:
577         print("[DEBUG-PYTHON]----precision default 8 bit signed----"
578             )
579         print("[DEBUG-PYTHON]---- Signed : ",PACK_TYPE.islower() , "
580             type: ",curr_data_precision , " ->",
581             curr_bitwidth_data_computation,"-----")
582     return True
583
584 @ffi.def_extern()
585 def push_input_tensor_to_heap( tensor ,size ,dim_size):
586     global input_tensors
587     global tot_size_input
588     #push the tensor to the heap for handling their transfeifr in
589     #the Prepare_p
590     tot_size=1
591     if not(FP) or not(BPF):
592         if PACK_TYPE.islower(): # signed
593             if curr_data_precision==INT8:
594                 tensor_i=ffi.cast("int8_t *",tensor)
595             elif curr_data_precision==INT16:
596                 tensor_i=ffi.cast("int16_t *",tensor)
597             elif curr_data_precision==INT32:
598                 tensor_i=ffi.cast("int32_t *",tensor)
599             else: # int64
600                 tensor_i=ffi.cast("int64_t *",tensor)
601         else: #unsigned
602             if curr_data_precision==INT8:
603                 tensor_i=ffi.cast("uint8_t *",tensor)
604             elif curr_data_precision==INT16:
605                 tensor_i=ffi.cast("uint16_t *",tensor)
606             elif curr_data_precision==INT32:
607                 tensor_i=ffi.cast("uint32_t *",tensor)
608             else: # int64
609                 tensor_i=ffi.cast("uint64_t *",tensor)

```

```

605     else:
606         if BFP:
607             tensor_i=ffi.cast("uint16_t *",tensor)
608         else:
609             tensor_i=ffi.cast("float *",tensor)
610         size_i=ffi.cast("int *",size)
611         tot_size=1
612         size_l=4*[1]
613         data_p=[]
614         for i in range(dim_size):
615             size_l[i]=size[i]
616             tot_size*=size[i]
617         tot_size_input+=tot_size
618         if _DEBUG_PRINT: print("[DEBUG-PYTHON]----- size of tensor
619             input ",tot_size_input,"-----")
620         for i in range(tot_size):
621             data_p.append(tensor_i[i])
622         input_tensors.append(Tensor(data_p,tot_size,size_l))
623
624     @ffi.def_extern()
625     def push_output_tensor_to_heap(tensor, size, dim_size):
626         global output_tensors
627         global tot_size_output
628         global output_tensors_p
629         #push the tensor to the heap for handling their transfeqr in
630         #the Prepare_p
631         tot_size=1
632         output_tensors_p.append(tensor)
633         if not(FP) or not(BPF):
634             if PACK_TYPE.islower(): # signed
635                 if curr_data_precision==INT8:
636                     tensor_i=ffi.cast("int8_t *",tensor)
637                 elif curr_data_precision==INT16:
638                     tensor_i=ffi.cast("int16_t *",tensor)
639                 elif curr_data_precision==INT32:
640                     tensor_i=ffi.cast("int32_t *",tensor)
641                 else: # int64
642                     tensor_i=ffi.cast("int64_t *",tensor)
643             else: #unsigned
644                 if curr_data_precision==INT8:
645                     tensor_i=ffi.cast("uint8_t *",tensor)
646                 elif curr_data_precision==INT16:
647                     tensor_i=ffi.cast("uint16_t *",tensor)
648                 elif curr_data_precision==INT32:
649                     tensor_i=ffi.cast("uint32_t *",tensor)
650                 else: # int64
651                     tensor_i=ffi.cast("uint64_t *",tensor)
652         else:
653             if BFP:

```

```

652     tensor_i=ffi.cast("uint16_t *",tensor)
653 else:
654     tensor_i=ffi.cast("float *",tensor)
655 size_i=ffi.cast("int *",size)
656 tot_size=1
657 size_l=4*[1]
658 data_p=[]
659 for i in range(dim_size):
660     size_l[i]=size[i]
661     tot_size*=size[i]
662 tot_size_output+=tot_size
663 if _DEBUG_PRINT: print("[DEBUG-PYTHON]----- size of tensor
664     output ",tot_size,"-----")
665 for i in range(tot_size):
666     data_p.append(0)
667 output_tensors.append(Tensor(data_p,tot_size,size_l))
668
669 @ffi.def_extern()
670 def push_weight_to_heap(tensor,size,dim_size):
671     global weight_tensors
672     global tot_size_weight
673     #push the tensor to the heap for handling their transfevr in
674     #the Prepare_p
675     tot_size=1
676     if not(FP) or not(BPF):
677         if PACK_TYPE.islower(): # signed
678             if curr_data_precision==INT8:
679                 tensor_i=ffi.cast("int8_t *",tensor)
680             elif curr_data_precision==INT16:
681                 tensor_i=ffi.cast("int16_t *",tensor)
682             elif curr_data_precision==INT32:
683                 tensor_i=ffi.cast("int32_t *",tensor)
684             else: # int64
685                 tensor_i=ffi.cast("int64_t *",tensor)
686             else: #unsigned
687                 if curr_data_precision==INT8:
688                     tensor_i=ffi.cast("uint8_t *",tensor)
689                 elif curr_data_precision==INT16:
690                     tensor_i=ffi.cast("uint16_t *",tensor)
691                 elif curr_data_precision==INT32:
692                     tensor_i=ffi.cast("uint32_t *",tensor)
693                 else: # int64
694                     tensor_i=ffi.cast("uint64_t *",tensor)
695             else:
696                 if BFP:
697                     tensor_i=ffi.cast("uint16_t *",tensor)
698                 else:
699                     tensor_i=ffi.cast("float *",tensor)
700 size_i=ffi.cast("int *",size)

```

```

699     tot_size=1
700     size_l=4*[1]
701     data_p=[]
702     for i in range(dim_size):
703         size_l[i]=size[i]
704         tot_size*=size[i]
705     tot_size_weight+=tot_size
706     if _DEBUG_PRINT: print("[DEBUG-PYTHON]----- size of tensor"
707                         " weight ",tot_size_weight,"-----")
708     for i in range(tot_size):
709         data_p.append(tensor_i[i])
710     weight_tensors.append(Tensor(data_p,tot_size,size_l))
711
712     @ffi.def_extern()
713     def Prepare_p(weight_num):
714         global output_fifo_buffer
715         global infifo_buffer_transfer
716         global weight_buffer
717         global csr_buffer
718         global overlay
719         global driver_wm
720         global driver_csr
721         global driver_fifo_in
722         global driver_fifo_out
723         global num_weight
724         global global_iteration
725         global global_iteration_shift_wm
726         global curr_data_precision
727         global weight_tensors
728         global filter_height
729         global filter_width
730         global weight_buffer_multiple
731         global index_wm
732         if _DEBUG_PRINT: print("[DEBUG - PYTHON ] --- Prepare p of"
733                         " DTPU class ---")
734         if _DEBUG_PRINT: print("[DEBUG - PYTHON ] --- in size",
735                         " input_size ,output size",output_size," ---")
736         if _DEBUG_PRINT: print("[DEBUG - PYTHON ] --- weight size",
737                         " weight_num , " ---")
738         #allocate buffers for data transfer
739         num_weight=weight_num
740         filter_height=num_weight*[0]
741         filter_width=num_weight*[0]
742         ## symmetric input/output fifo
743         output_fifo_buffer=allocate(shape=(INFIFO_SIZE,),dtype='u8')
744         weight_buffer=allocate(shape=(WMEM_SIZE,),dtype='u8')
745         csr_buffer=allocate(shape=(CSRMEM_SIZE,),dtype='u8')
746         infifo_buffer_transfer=allocate(shape=(INFIFO_SIZE,),dtype='u8'
747             ')

```

```

743     driver_wm=overlay.axi_dma_weight_mem
744     driver_csr=overlay.axi_dma_csr_mem
745     driver_fifo_in=overlay.axi_dma_infifo
746     driver_fifo_out=overlay.axi_dma_outfifo
747     #
748     ##### populate buffers pack depending on the precision
749     #
750     if _DEBUG_PRINT:
751         print("[DEBUG - PYTHON] --- Prepare p of DTPU class " ,
752               num_weight,"weight to transfer ---")
753         for i in range(num_weight):
754             tmp=weight_tensors[i]
755             print("[DEBUG-PYTHON] --- weight ",i,"-----")
756             print("[DEBUG-PYTHON] --- size ",*tmp.size_l,"-----")
757             for j in range(tmp.tot_dim):
758                 print(tmp.data[j],end=" ")
759             print("",end="\n")
760     index_wm=0# it eats the first data ?
761     shift=int(64/curr_bitwidth_data_computation)
762     iter=int(tot_size_weight/(WMEM_SIZE*(64/
763                               curr_bitwidth_data_computation))) # if it fits in th
764     eaccelerator memory
765     # always 4D tensors
766     # assumptio is that the filter sizes always fit the
767     # accelerator
768     if False:
769         weight_buffer_multiple=1
770         for w_ind in range(1): # pack only the weight for deep wise
771             convolution
772             tmp=np.array(weight_tensors[w_ind].data , dtype=DTYPE_NP)
773             tmp=tmp.reshape(*weight_tensors[w_ind].size_l)
774             filter_height[w_ind],filter_width[w_ind]=tmp.shape[1:3]
775             for i in range(len(tmp)):
776                 for l in range(weight_tensors[w_ind].size_l[3]):
777                     global_iteration_shift_wm.append(index_wm)
778                     for j in range(len(tmp[i])):
779                         # boundary check
780                         shift=int(64/curr_bitwidth_data_computation)
781                         if shift > len(tmp[i]):
782                             shift=len(tmp[i])
783                         weight_buffer[index_wm]=np.uint64(int.from_bytes(
784                             tmp[i,j,0:shift,l],byteorder="little",signed=
785                             False))
786                         index_wm+=1

```

```

780         for j in range(ROWS-len(tmp[i])):
781             weight_buffer[index_wm]=0
782             index_wm+=1 # padding with zeros
783     else:
784         #print("it requires multiple iterations for the weight
785         # matrix") # multiple iteration on total weight 1MB should
786         # be enou-gh
787     weight_buffer_multiple=[]*np.uint64(0)
788     for w_ind in range(1): # pack only the weight for deep wise
789         convolution
790         tmp=np.array(weight_tensors[w_ind].data, dtype=DTYPE_NP)
791         tmp=tmp.reshape(*weight_tensors[w_ind].size_l)
792         filter_height[w_ind],filter_width[w_ind]=tmp.shape[1:3]
793         for i in range(len(tmp)):
794             for l in range(weight_tensors[w_ind].size_l[3]):
795                 global_iteration_shift_wm.append(index_wm)
796                 for j in range(len(tmp[i])):
797                     # boundary check
798                     shift=int(64/curr_bitwidth_data_computation)
799                     if shift > len(tmp[i]):
800                         shift=len(tmp[i])
801                         weight_buffer_multiple.append(np.uint64(int.
802                             from_bytes( tmp[i,j,0:shift ,l],byteorder="
803                             little ",signed=False)))
804                     index_wm+=1
805                     for j in range(ROWS-len(tmp[i])):
806                         weight_buffer[index_wm]=0
807                         index_wm+=1 # padding with zeros
808     if _DEBUG_PRINT:
809         for i in range(10):
810             print(hex(weight_buffer[i]))
811 #####
812 #####
813 #####
814 #####
815 #####
816 #####
817 #####
818 #####
819 #####
820 #####
821 #####
822 #####
823 #####

```

## A. Accelerator library

---

```
824     global global_iteration_shift_wm
825     global curr_data_precision
826     global input_tensors
827     global output_tensors
828     global filter_width
829     global filter_height
830     global tot_size_output
831     global tot_size_input
832     global output_tensors_p
833     global avg_hw_execution
834     global n_execution
835     global avg_hw_execution_internal
836     global n_execution_internal
837     global weight_buffer_multiple
838 #
# ##### populate buffers pack depending on the precision
#####
839 #####
840 #
#####
841 tmp=[]
842 if _DEBUG_PRINT:
843     print("[DEBUG - PYTHON] --- Invoke p of DTPU class",
844           input_size_num_weight,"input tensors to transfer ---")
845     for i in range(input_size_num_weight):
846         tmp=input_tensors[i]
847         print("[DEBUG-PYTHON] --- input tensor ",i,"---")
848         print("[DEBUG-PYTHON] --- size ",*tmp.size_l,"---")
849         for j in range(tmp.tot_dim):
850             print(tmp.data[j],end=" ")
851 index=0
852 shift=int(64/curr_bitwidth_data_computation)
853 # check if it fits the inputs
854 # always 4D tensors
855 # assumption is that the filter sizes always fit the
856 # accelerator
857 #then compact
858 ## split the input shape into submatrices equal to filter
859 # sizes
860 applied_weight=0
861 #over allocate input_fifo_buffer
862 input_fifo_buffer = []*np.uint64(0)
863 for w_ind in range(len(input_tensors)):
864     tmp=np.array(input_tensors[w_ind].data, dtype=DTYPE_NP)
865     tmp=tmp.reshape(*input_tensors[w_ind].size_l)
866     for batch in range(len(tmp)):
867         for channel in range(tmp.shape[-1]):
```

```

865     tmp_s=tmp[batch ,:,:,channel]
866     #iteration for the whole matrix
867     for i in range(len(tmp_s)-filter_height[applyed_weight]):
868         :
869         for j in range(len(tmp_s[i])-filter_width[applyed_weight]):
870             tmp_ss=tmp_s[i:i+filter_height[applyed_weight],j:j+
871                         filter_width[applyed_weight]]
872             for row in range(len(tmp_ss)):
873                 shift=int(64/curr_bitwidth_data_computation)
874                 if shift> len(tmp_ss):
875                     shift=len(tmp_ss)
876                     input_fifo_buffer.append(np.uint64(int.from_bytes(
877                         tmp_ss[row,0:shift],byteorder="little ",signed=
878                         False)))
879                     index+=1
880             input_fifo_buffer=np.array(input_fifo_buffer,dtype='u8')
881             input_fifo_buffer=np.reshape(input_fifo_buffer,newshape=(index
882                                         ,))
883             if _DEBUG_PRINT:
884                 for i in range(10):
885                     print(hex(input_fifo_buffer[i]))
886             #iterate on the output matrix with also multiple weight
887             # iteration and inputs
888             ## assumption is that the output tensor is always one!
889             ## getting the output matrix structure
890             #accelerator.write(CMD, (0x00000000 |(CMD_EXECUTE_CONTINOUS
891                                         <<16)))
892             output_matrix=np.array(output_tensors[0].data, dtype=DTYPE_NP)
893             output_matrix=output_matrix.reshape(*output_tensors[0].size_1)
894             point_wise=np.array(weight_tensors[1].data, dtype=DTYPE_NP)
895             ##### deepwise convolution #####
896             if _DEBUG_PRINT:
897                 print("[DEBUG-PYTHON] ----- deepwise convolution
898                               -----")
899             if _TIME_PROBES:
900                 start_time=time.time()
901                 for shift_w in range(math.ceil(len(weight_buffer_multiple)/
902                                         WMEM_SIZE)):
903                     ##### program the dma for the weight #####
904                     if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- transferring weight
905                                         buffer -----")
906                     weight_buffer[0:len(weight_buffer_multiple[WMEM_SIZE*(
907                                         shift_w):WMEM_SIZE*(shift_w+1)])]=weight_buffer_multiple[
908                                         WMEM_SIZE*(shift_w):WMEM_SIZE*(shift_w+1)]

```

## A. Accelerator library

---

```
901     driver_wm.sendchannel.transfer(weight_buffer)
902     driver_wm.sendchannel.wait()
903     for batch_i in range(input_tensors[0].size_![0]):
904         for channel_i in range(input_tensors[0].size_![-1]):
905             ##### program the dma for the csr reg #####
906             ##### program the dma for the in/out fifos #####
907             if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- transferring
908                 csr buffer for weight----")
909             csr_buffer[ARITHMETIC_PRECISION]=(
910                 global_iteration_shift_wm[channel_i]<<32) | ((NO_FP
911                     <<8)) | (ACTIVATE_CHAIN<<4)| (curr_data_precision)
912             #csr_buffer.flush()
913             driver_csr.sendchannel.transfer(csr_buffer)
914             #driver_csr.sendchannel.wait()
915             for infifo_shift in range(math.ceil(input_fifo_buffer.
916                 size/INFIFO_SIZE)):
917                 ##### program the dma for the in/out fifos #####
918                 if _TIME_PROBES:
919                     start_time_i=time.time()
920                     if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- transferring
921                         input buffer",infifo_shift,"----")
922                     infifo_buffer_transfer[0:input_fifo_buffer[INFIFO_SIZE
923                         *(infifo_shift):INFIFO_SIZE*(infifo_shift+1)].size
924                         ]=input_fifo_buffer[INFIFO_SIZE*(infifo_shift):
925                             INFIFO_SIZE*(infifo_shift+1)]
926                     driver_fifo_in.sendchannel.transfer(
927                         infifo_buffer_transfer)
928                     #driver_fifo_in.sendchannel.wait()
929                     accelerator.write(OARG0_LENGTH,OUTFIFO_SIZE) # size
930                         outfifo
931                     accelerator.write(CMD, (0x00000000 |(CMD_EXECUTE_STEP
932                         <<16)))
933                     accelerator.write(CMD,((CMD_UPDATE_OUT_ARG<<16)|(1)))
934                     driver_fifo_out.recvchannel.transfer(
935                         output_fifo_buffer)
936                     if _DEBUG_PRINT: print("[DEBUG-PYTHON]----- getting
937                         output data ----")
938                     driver_fifo_out.recvchannel.wait()
939                     if _TIME_PROBES:
940                         end_time_i=time.time()
941                         avg_hw_execution_internal+=end_time_i-start_time_i
942                         n_execution_internal+=1
943                         if _DEBUG_PRINT: print(output_fifo_buffer)
944                         accelerator.write(CMD,((CMD_UPDATE_IN_ARG<<16)|(4))) #
945                             update input fifo
946                         #
```

```

#####
936     ##### unpack the output buffer depending on the
937         precision #####
938
939     ## get values from output fifo buffer and put them
940         into an array in order to sum all the data
941     for i in range(output_matrix.shape[1]-1):
942         for j in range(output_matrix.shape[2]-1):
943             tmp_sum=np.zeros(shape=(ROWS, int(64/
944                 curr_bitwidth_data_computation)), dtype=DTYPE_NP
945             )
946             tmp_data=output_fifo_buffer[channel_i*(ROWS*
947                 COLUMNS)+i*ROWS+j*COLUMNS:channel_i*(ROWS*
948                 COLUMNS)+(i+1)*ROWS+(j+1)*COLUMNS]
949             tmp_sum=np.frombuffer(tmp_data.tobytes(), dtype=
950                 DTYPE_NP)
951             #reshuffle and check if it is worth it
952             #if tmp_data.size >0:
953                 # for row in range(len(tmp_data)):
954                     # if row in tmp_data:
955                         #     tmp_sum[row]=np.frombuffer(tmp_data[row].
956                             tobytes(), dtype=DTYPE_NP)#convert(tmp_data[row]
957                         ])
958             output_matrix[batch_i,i,j,channel_i]=np.multiply(
959                 tmp_sum.sum(dtype=DTYPE_NP), point_wise[
960                     channel_i], dtype=DTYPE_NP)
961             accelerator.write(CMD,((CMD_UPDATE_IN_ARG<<16)|(1))) #
962                 update csr
963             accelerator.write(CMD,((CMD_UPDATE_OUT_ARG<<16)|(1)))
964             accelerator.write(CMD,((CMD_UPDATE_IN_ARG<<16)|(2))) #
965                 update w memory
966             #if _DEBUG_PRINT:
967                 # print("[DEBUG-PYTHON]----- point wise convolution
968                     -----")
969             if _TIME_PROBES:
970                 end_time=time.time()
971                 avg_hw_execution+=end_time-start_time
972                 n_execution+=1
973             accelerator.write STATUS,0x00000003##clear status
974             #accelerator.write(CMD,((CMD_UPDATE_IN_ARG<<16)|(1))) # update
975                 csr
976             #accelerator.write(CMD, (0x00000000 |(
977                 CMD_STOP_EXECUTE_CONTINOUS<<16))) # stop accelerator
978             #####
979             ##### point wise convolution ##### moved inside
980                 previous loop

```

## A. Accelerator library

---

```
964 #####  
965 #for batch_i in range(len(output_matrix)):  
966 #    for i in range(len(output_matrix[batch_i])):  
967 #        for j in range(len(output_matrix[batch_i,i])):  
968 #            for channel_i in range(len(output_matrix[batch_i,i,j])):  
969 #                :  
970 #                    output_matrix[batch_i,i,j,channel_i]=output_matrix[  
971 #                        batch_i,i,j,channel_i]*weight_tensors[1].data[channel_i]  
972 if _DEBUG_PRINT: print("[DEBUG -PYTHON] ----- accelerator done  
973 -----")  
974 if _DEBUG_PRINT:  
975     print("[DEBUG-PYTHON] ----- final output data to tensorflow  
976 -----")  
977     print(output_matrix)  
978 # copy the output matrix to tensorflow environment ffi.memmove  
979 # (dest,src,nbytets)  
980 ffi.memmove(ffi.buffer(output_tensors_p[0],output_matrix.  
981             nbytes),output_matrix,output_matrix.nbytes)  
982 # save the pointer to the output and then substitute the  
983 # values into the point wise convolution  
984 #clean up input/output  
985 input_tensors=[]  
986 output_tensors=[]  
987 tot_size_input=0  
988 tot_size_output=0  
989 del input_fifo_buffer  
990 return True  
991  
992 @ffi.def_extern()  
993 def ResetHardware_p():  
994     global accelerator  
995     global overlay  
996     if _DEBUG_PRINT: print("[DEBUG - PYTHON ] ----- Reset hardware p  
997         function -----")  
998     overlay.reset()  
999     accelerator.write(CTRL,0x00000001)  
1000    accelerator.write(CTRL,0x00000000)  
1001    return True  
1002  
1003 @ffi.def_extern()  
1004 def destroy_p():  
1005     global infifo_buffer_transfer  
1006     global output_fifo_buffer  
1007     global csr_buffer  
1008     global weight_buffer  
1009     global accelerator  
1010     global overlay  
1011     global global_iteration_shift_wm  
1012     global weight_tensors
```

```
1005     global input_tensors
1006     global output_tensors
1007     if _DEBUG_PRINT: print("[DEBUG - PYTHON] --- destroying the
1008         buffers ---")
1009     infifo_buffer_transfer.freebuffer()
1010     output_fifo_buffer.freebuffer()
1011     csr_buffer.freebuffer()
1012     weight_buffer.freebuffer()
1013     del accelerator
1014     del overlay
1015     del global_iteration_shift_wm
1016     del weight_tensors
1017     del input_tensors
1018     del output_tensors
1019     return True
1020
1021 @ffi.def_extern()
1022 def CopyFromBufferHandle_p():
1023     if _DEBUG_PRINT: print("[DEBUG - PYTHON] --- the from
1024         delegate and buffers ---")
1025     return True
1026
1027 @ffi.def_extern()
1028 def CopyToBufferHandle_p():
1029     if _DEBUG_PRINT: print("[DEBUG - PYTHON] --- copying to the
1030         delegate and buffers ---")
1031     return True
1032
1033 @ffi.def_extern()
1034 def FreeBufferHandle_p():
1035     global output_fifo_buffer
1036     global csr_buffer
1037     global weight_buffer
1038     global driver_csr
1039     global driver_wm
1040     global driver_fifo_in
1041     global driver_fifo_out
1042     global accelerator
1043     if _DEBUG_PRINT: print("[DEBUG - PYTHON] --- freeing buffers
1044         ---")
1045     output_fifo_buffer.freebuffer()
1046     csr_buffer.freebuffer()
1047     weight_buffer.freebuffer()
1048     del accelerator
1049     del driver_csr
1050     del driver_wm
1051     del driver_fifo_in
1052     del driver_fifo_out
1053
1054 @ffi.def_extern()
```

```

1050 def start_power_consumption():
1051     global xadc_mon
1052     if _DEBUG_PRINT: print("[DEBUG-PYTHON] —— start measurement
1053         of power consumption ——")
1054     if xadc_mon is not None:
1055         try:
1056             _thread.start_new_thread( sample_power, ("Sampling power",
1057                 0.5) ) # every 1ms
1058         except:
1059             print("Error: unable to start thread")
1060     return True
1061
1062 @ffi.def_extern()
1063 def print_power_consumption_p():
1064     global xadc_mon
1065     global ps_power
1066     global pl_power
1067     global mem_power
1068     if _DEBUG_PRINT: print("[DEBUG-PYTHON] —— printing power
1069         consumption from xadc readings ——")
1070     ##### Retrieve and display current temperature #####
1071     ##### Retrieve and display current temperature #####
1072     tmp=( xadc_mon.read(TEMPERATURE) & 0x0000FFF0) >> 4
1073     tmp=(tmp* 503.975)/4096 - 273.15
1074     print("Current temperature:", round(tmp,3), " C")
1075     print("Average execution temperature:", round(tmp_avg/n_sample
1076         ,3), " C")
1077     print("Max temperature:", round(tmp_max,3), " C")
1078     print("Min temperature:", round(tmp_min,3), " C")
1079     # printing power consumption
1080     tot_power=ps_power+pl_power+mem_power
1081     print("Average power consumption=", round(tot_power*1000/
1082         n_sample,5), " mWatt")
1083     print("—> Processing System:", round(ps_power*1000/n_sample
1084         ,5), " mWatt")
1085     print("—> Programmable Logic:", round(pl_power*1000/n_sample
1086         ,5), " mWatt")
1087     print("—> Memory:", round(mem_power*1000/n_sample,3), " mWatt"
1088         )
1089     print("Maximum power consumption")
1090     print("—> Processing System:", round(ps_power_max*1000,5), "
1091         mWatt")
1092     print("—> Programmable Logic:", round(pl_power_max*1000,5), "
1093         mWatt")
1094     print("—> Memory:", round(mem_power_max*1000,3), " mWatt")
1095     print("Minimum power consumption")
1096     print("—> Processing System:", round(ps_power_min*1000,5), "
1097         mWatt")

```

```
1088     print("----> Programmable Logic:",round(pl_power_min*1000,5),"  
1089         mWatt")  
1090     print("----> Memory:",round(mem_power_min*1000,5)," mWatt")  
1091     return True  
1092  
1093 @ffi.def_extern()  
1094 def activate_time_probe_p(activate):  
1095     global _TIME_PROBES  
1096     if _DEBUG_PRINT: print("[DEBUG-PYTHON]--- activating time  
1097         probe in python -----")  
1098     if not(_TIME_PROBES) and activate:  
1099         print("Time probes activated")  
1100         _TIME_PROBES=True  
1101  
1102 @ffi.def_extern()  
1103 def print_python_time_probes():  
1104     if _DEBUG_PRINT: print("[DEBUG-PYTHON]----- printing python  
1105         time probes -----")  
1106     print("Hardware execution time and rebuilding output matrix:",  
           avg_hw_execution/n_execution," [s]")  
1107     print("Hardware execution time:", avg_hw_execution_internal/  
           n_execution_internal," [s]")  
1108 #print("Hardware calls:",n_execution_internal)  
1109 return True
```



# B

## Top level entity of DTPU core

```
1 // ======  
2 // Filename : dtpu_core.v  
3 // Created On : 2020-04-22 17:05:56  
4 // Last Modified : 2020-05-20 15:03:03  
5 // Revision :  
6 // Author : Angione Francesco  
7 // Company : Chalmers University of Technology, Sweden  
- Politecnico di Torino, Italy  
8 // Email : francescoangione8@gmail.com  
9 //  
10 // Description : Cogitantium, the dumb tensor processor  
unit,top level entity of the accelerator  
11 //  
12 //  
13 // ======  
14  
15 'timescale 1ns / 1ps  
16 'include "precision_def.vh"  
17  
18 //`define DUMMY  
19  
20 module dtpu_core  
21 #(parameter DATA_WIDTH_MAC=64,  
22     ROWS=3 ,  
23     COLUMNS=3 ,  
24     SIZE_WMEMORY=8196 ,  
25     ADDRESS_SIZE_WMEMORY=32 ,  
26     ADDRESS_SIZE_CSR=32 ,  
27     SIZE_CSR=1024 ,  
28     DATA_WIDTH_CSR=8 ,  
29     DATA_WIDTH_WMEMORY=64 ,  
30     DATA_WIDTH_FIFO_IN=64 ,  
31     DATA_WIDTH_FIFO_OUT=64 ,  
32     MAX_BOARD_DSP=220  
33 )
```

## B. Top level entity of DTPU core

---

```
34  (
35      input wire clk,
36      (* X_INTERFACE_INFO = "xilinx.com:signal:reset:1.0
37          aresetn RST" *)
38      (* X_INTERFACE_PARAMETER = "POLARITY ACTIVE_LOW" *)
39      input wire aresetn,
40      input wire test_mode,
41      input wire enable,
42      /////////////////////////////////
43      ////////////// CSR INTERFACE ///////////
44      /////////////////////////////////
45      (* X_INTERFACE_PARAMETER = "MASTER_TYPE BRAM_CTRL, MEM_ECC
46          no, MEM_WIDTH 8, MEM_SIZE 1024" *)
47      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
48          csr_mem_interface EN" *)
49      output wire           csr_ce,
50      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
51          csr_mem_interface DOUT" *)
52      input wire [DATA_WIDTH_CSR-1:0]    csr_dout,
53      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
54          csr_mem_interface DIN" *)
55      output wire [DATA_WIDTH_CSR-1:0]    csr_din,
56      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
57          csr_mem_interface WE" *)
58      output wire           csr_we,
59      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
60          csr_mem_interface ADDR" *)
61      output wire [ADDRESS_SIZE_CSR-1:0]    csr_address,
62      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
63          csr_mem_interface CLK" *)
64      output wire           csr_clk,
65      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl:1.0
66          csr_mem_interface RST" *)
67      output wire           csr_reset,
68
69      /////////////////////////////////
70      ////////////// WEIGHT MEMORY ///////////
71      /////////////////////////////////
72      (* X_INTERFACE_PARAMETER = "MASTER_TYPE BRAM_CTRL,
73          MEM_ECC no, MEM_WIDTH 64, MEM_SIZE 8192" *)
74      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
75          :1.0 weight_mem_interface EN" *)
76      output wire   wm_ce,
77      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
78          :1.0 weight_mem_interface DOUT" *)
79      input wire [DATA_WIDTH_WMEMORY-1:0]      wm_dout,
80      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
81          :1.0 weight_mem_interface DIN" *)
```

```
70      output wire [DATA_WIDTH_WMEMORY-1:0]          wm_din,
71      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
72         :1.0 weight_mem_interface WE" *)
73      output wire                      wm_we,
74      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
75         :1.0 weight_mem_interface ADDR" *)
76      output wire [ADDRESS_SIZE_WMEMORY-1:0]  wm_address,
77      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
78         :1.0 weight_mem_interface CLK" *)
79      output wire        wm_clk,
80      (* X_INTERFACE_INFO = "xilinx.com:interface:bram_rtl
81         :1.0 weight_mem_interface RST" *)
82      output wire        wm_reset,
83
84      ///////////////////////////////
85      ////////////////// INPUT DATA FIFO /////////////////////
86      ///////////////////////////////
87      ////////////////// using stream axi
88      (* X_INTERFACE_INFO = "xilinx.com:interface:
89         acc_fifo_read:1.0 input_fifo RD_DATA" *)
90      input wire [DATA_WIDTH_FIFO_IN-1:0] infifo_dout,
91      (* X_INTERFACE_INFO = "xilinx.com:interface:
92         acc_fifo_read:1.0 input_fifo RD_EN" *)
93      output wire infifo_read,
94      (* X_INTERFACE_INFO = "xilinx.com:interface:
95         acc_fifo_read:1.0 input_fifo EMPTY_N" *)
96      input wire infifo_is_empty,
97
98      ///////////////////////////////
99      ////////////////// OUTPUT DATA FIFO ///////////////////
100      ///////////////////////////////
101      ////////////////// using stream axi
102      (* X_INTERFACE_INFO = "xilinx.com:interface:
103         acc_fifo_write:1.0 output_fifo WR_DATA" *)
104      output wire [DATA_WIDTH_FIFO_OUT-1:0] outfifo_din,
105      (* X_INTERFACE_INFO = "xilinx.com:interface:
106         acc_fifo_write:1.0 output_fifo WR_EN" *)
107      output wire outfifo_write,
108      (* X_INTERFACE_INFO = "xilinx.com:interface:
109         acc_fifo_write:1.0 output_fifo FULL_N" *)
110      input wire outfifo_is_full,
111
112      ///////////////////////////////
113      ////////////////// CONTROL FROM/TO PS ///////////////////
114      ///////////////////////////////
115      (* X_INTERFACE_INFO = "xilinx.com:interface:
116         acc_handshake_rtl:1.0 control_interface ap_start"
117         *)
```

## B. Top level entity of DTPU core

---

```
107      input wire cs_start ,
108      (* X_INTERFACE_INFO = "xilinx.com:interface:
109         acc_handshake_rtl:1.0 control_interface ap_ready"
110         *)
111      output wire cs_ready ,
112      (* X_INTERFACE_INFO = "xilinx.com:interface:
113         acc_handshake_rtl:1.0 control_interface ap_done" *)
114      output wire cs_done ,
115      (* X_INTERFACE_INFO = "xilinx.com:interface:
116         acc_handshake_rtl:1.0 control_interface ap_continue
117         " *)
118      input wire cs_continue ,
119      (* X_INTERFACE_INFO = "xilinx.com:interface:
120         acc_handshake_rtl:1.0 control_interface ap_idle" *)
121      output wire [3:0]state ,
122      output wire [3:0]d_out
123      );
124      /////////////////
125      //***** Cogitantium *****
126      //----- Cogitantium -----
127      // the dumb tensor processing unit
128      //***** ///////////////////////
129      wire [COLUMNS*ROWS*DATA_WIDTH_FIFO_OUT-1:0]
130          weight_to_mxu;
131      wire [COLUMNS*DATA_WIDTH_FIFO_IN-1:0] input_data_to_mxu
132          ;
133      wire [ROWS*DATA_WIDTH_FIFO_OUT-1:0]
134          output_data_from_mxu;
135      wire enable_deskew_ff_i,enable_enskew_ff_i;
136      wire ['LOG_ALLOWED_PRECISIONS-1:0] data_precision;
137      wire enable_i;
138      wire enable_load_array;
139      wire [ROWS*COLUMNS-1:0]read_weight_memory;
140      wire [COLUMNS:0]enable_load_activation_data;
141      wire [COLUMNS:0]enable_store_activation_data;
142      wire enable_cnt;
143      wire ld_max_cnt;
144      wire enable_cnt_weight;
145      wire ld_max_cnt_weight;
146      wire enable_chain;
147      wire ld_weight_page_cnt;
148      wire [1:0]enable_fp_unit;
```

```
147     wire [$clog2(COLUMNS):0] max_cnt_from_cu;
148     wire [$clog2(ROWS*COLUMNS):0] max_cnt_weight_from_cu;
149     wire reset_i;
150
151     assign d_out=data_precision;
152
153     assign reset_i=~aresetn;
154     /////////////////////////////////
155     // MATRIX MULTIPLICATION UNIT ///////////////////
156     /////////////////////////////////
157     mxu_wrapper
158     #(.M(ROWS), // matrix row -> weights
159       .K(COLUMNS), // matrix columns -> input data
160       .max_data_width(DATA_WIDTH_MAC), // it must be a
161         divisor of 64
162       .MAX_BOARD_DSP(MAX_BOARD_DSP)
163     ) engine (
164       .data_type(data_precision),
165       .reset(reset_i),
166       .clk(clk),
167       .enable(enable_i),
168       .enable_chain(enable_chain),
169       .enable_fp_unit(enable_fp_unit),
170       .enable_in_ff(enable_enskew_ff_i),
171       .enable_out_ff(enable_deskew_ff_i),
172       .test_mode(test_mode),
173       .input_data(input_data_to_mxu),
174       .weight(weight_to_mxu),
175       .y(output_data_from_mxu)
176   );
177
178   /////////////////////////////////
179   // CONTROL UNIT ///////////////////
180   /////////////////////////////////
181   control_unit #(
182     .DATA_WIDTH_FIFO_IN(DATA_WIDTH_FIFO_IN),
183     .DATA_WIDTH_FIFO_OUT(DATA_WIDTH_FIFO_OUT),
184     .DATA_WIDTH_WMEMORY(DATA_WIDTH_WMEMORY),
185     .DATA_WIDTH_CSR(DATA_WIDTH_CSR),
186     .ROWS(ROWS),
187     .COLUMNS(COLUMNS),
188     .ADDRESS_SIZE_CSR(ADDRESS_SIZE_CSR),
189     .ADDRESS_SIZE_WMEMORY(ADDRESS_SIZE_WMEMORY))
190   cu(
191     .clk(clk),
192     .reset(reset_i),
193     .test_mode(test_mode),
194     .glb_enable(enable),
195     .enable_mxu(enable_i),
196     .csr_address(csr_address),
```

```

195     .csr_dout(csr_dout),
196     .csr_ce(csr_ce),
197     .csr_reset(csr_reset),
198     .csr_we(csr_we),
199     .wm_ce(wm_ce),
200     .wm_reset(wm_reset),
201     .wm_we(wm_we),
202     .infifo_is_empty(infifo_is_empty),
203     .infifo_read(infifo_read),
204     .outfifo_is_full(outfifo_is_full),
205     .outfifo_write(outfifo_write),
206     .cs_continue(cs_continue),
207     .cs_done(cs_done),
208     .cs_idle(cs_idle),
209     .cs_ready(cs_ready),
210     .cs_start(cs_start),
211     .state_out(state),
212     .enable_deskew_ff(enable_deskew_ff_i),
213     .enable_enskew_ff(enable_enskew_ff_i),
214     .enable_fp_unit(enable_fp_unit),
215     .enable_chain(enable_chain),
216     .enable_load_array(enable_load_array),
217     .data_precision(data_precision),
218     .read_weight_memory(read_weight_memory),
219     .enable_load_activation_data(
220         enable_load_activation_data),
221     .enable_store_activation_data(
222         enable_store_activation_data),
223     .enable_cnt(enable_cnt),
224     .ld_max_cnt(ld_max_cnt),
225     .enable_cnt_weight(enable_cnt_weight),
226     .ld_max_cnt_weight(ld_max_cnt_weight),
227     .ld_weight_page_cnt(ld_weight_page_cnt),
228     .start_value_wm(start_value_wm),
229     .max_cnt_from_cu(max_cnt_from_cu), // it depends on
230         the current bitwidt [$clog2(COLUMNS):0]
231     .max_cnt_weight_from_cu(max_cnt_weight_from_cu) //[
232         $clog2(ROWS):0]
233
234     );
235
236
237
238     ls_array
239     #(.ROWS(ROWS),

```

```

240     .COLUMNS(COLUMNS),
241     .data_in_width(DATA_WIDTH_FIFO_IN),
242     .data_in_mem(DATA_WIDTH_WMEMORY),
243     .address_leng_wm(ADDRESS_SIZE_WMEMORY),
244     .size_wmemory(SIZE_WMEMORY)) ls_array_inst
245 (
246     .clk(clk),
247     .reset(reset_i),
248     .enable_load_array(enable_load_array),
249     .data_precision(data_precision),
250     .read_weight_memory(read_weight_memory),
251     .infifo_read(infifo_read),
252     .outfifo_write(outfifo_write),
253     .input_data_from_fifo(infifo_dout), // [data_in_width-1:0]
254     .data_to_fifo_out(outfifo_din), // [data_in_width-1:0]
255     .data_from_weight_memory(wm_dout), // [data_in_mem-1:0]
256     .data_from_mxu(output_data_from_mxu), // [data_in_width*ROWS
257         -1:0]
258     .data_to_mxu(input_data_to_mxu), // [data_in_width*COLUMNS
259         -1:0]
260     .weight_to_mxu(weight_to_mxu), // [data_in_width*ROWS-1:0]
261     .wm_address(wm_address), // [address_leng_wm-1:0]
262     .enable_load_activation_data(enable_load_activation_data),
263     .enable_store_activation_data(enable_store_activation_data)
264     ,
265     .enable_cnt(enable_cnt),
266     .ld_max_cnt(ld_max_cnt),
267     .enable_cnt_weight(enable_cnt_weight),
268     .ld_max_cnt_weight(ld_max_cnt_weight),
269     .ld_weight_page_cnt(ld_weight_page_cnt),
270     .start_value_wm(start_value_wm),
271     .max_cnt_from_cu(max_cnt_from_cu), // it depends on the
272         current bitwidt [$clog2(COLUMNS):0]
273     .max_cnt_weight_from_cu(max_cnt_weight_from_cu) //[$clog2(
274         ROWS):0]
275 );
276
277 `endif
278
279
280
281
282
283     ifdef DUMMY
284     always @(posedge clk) begin
285     if(reset_i) begin
286     input_data_from_fifo<=0;
287     weight_from_memory<=0;
288     end else begin
289             if (enable_load_array && fifo_read ) begin

```

## B. Top level entity of DTPU core

---

```
284         input_data_from_fifo <= infifo_dout;
285         weight_from_memory <= wm_dout;
286     end
287
288 end
289 end
290 // dummy assignment for 3 columns and rows
291 assign outfifo_din=( outfifo_write ? input_data_to_fifo:64'
292   b0);
293
294 `endif
295
296 // same clock for bram interface
297 assign csr_clk=clk;
298 assign wm_clk=clk;
299
300 endmodule
```