University of Sousse, Tunisia

University of Mons, Belgium

Full Python Interface Control: Auto Generation And Adaptation of Deep Neural Networks For Edge Computing and IoT Applications FPGA-Based Acceleration

Presented by

INISTA

August 2021

Co-authors

Alexandre Quenon

Vitor Ramos Gomes da Silva

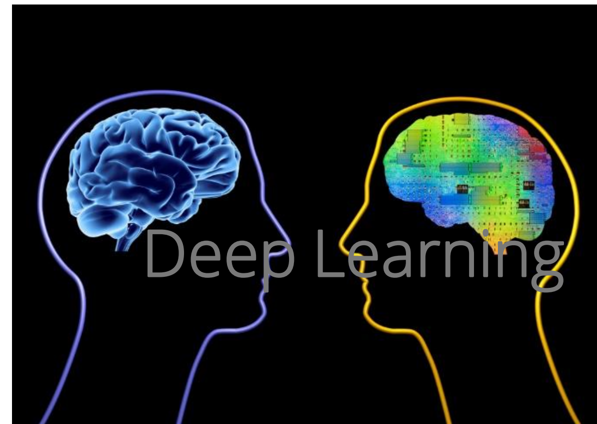
Prof Carlos VALDERRAMA

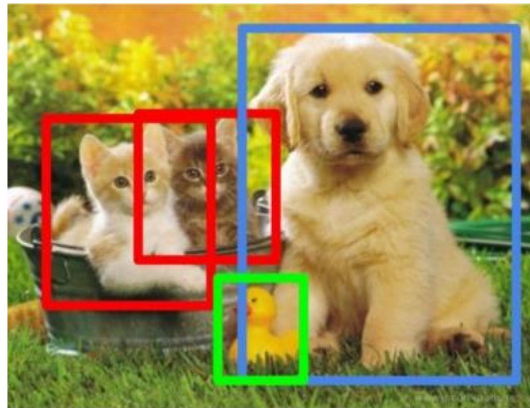
Prof Chokri Souani

1. Introduction, motivations and objectives
2. DNN’s hardware implementation
3. Automated Environement
4. Results
5. Conclusion
6. Perspectives

1. Introduction, motivations and objectives
2. DNN’s hardware implementation
3. Automated Environement
4. Results
5. Conclusion
6. Perspectives

Introduction, motivations and objectives





CAT, DOG, DUCK

Input
Chest X-Ray Image

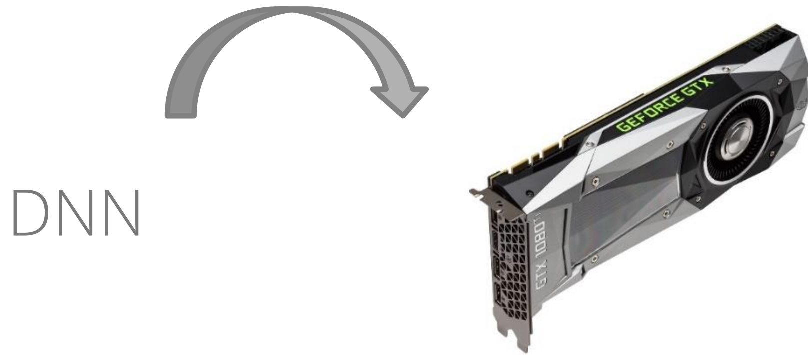
CheXNet
121-layer CNN

Output
Pneumonia Positive (85%)

Introduction, motivations and objectives

Motivations

DNN



Problems at the levels of:

X - Energy Dissipation: HPC, Embedded Systems

X – Miniaturization: embedded systems

Intensive processing on computing units, specifically GPUs:

- The size of the networks
- The volume of data involved
- Time required for learning

The use of recent DNNs becomes complicated in embedded systems and even in high-performance computing (HPC).

Introduction, motivations and objectives

Motivations

Problems at the levels of:

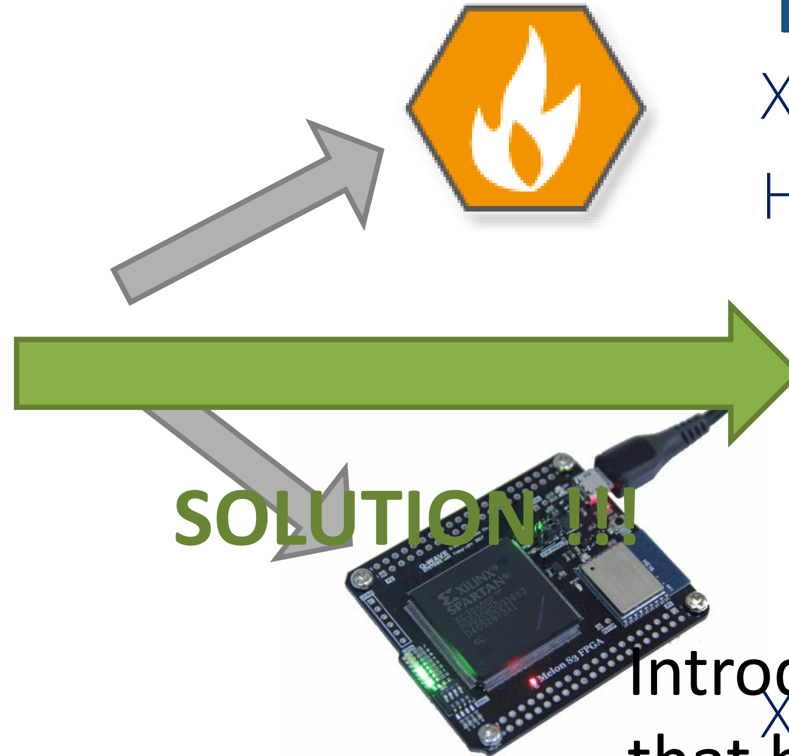
Hardware acceleration:

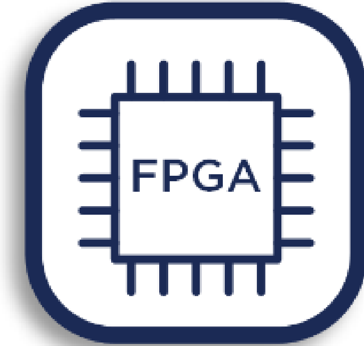
X - Energy Dissipation:

HPC, Embedded Systems

X – Miniaturization:

embedded systems

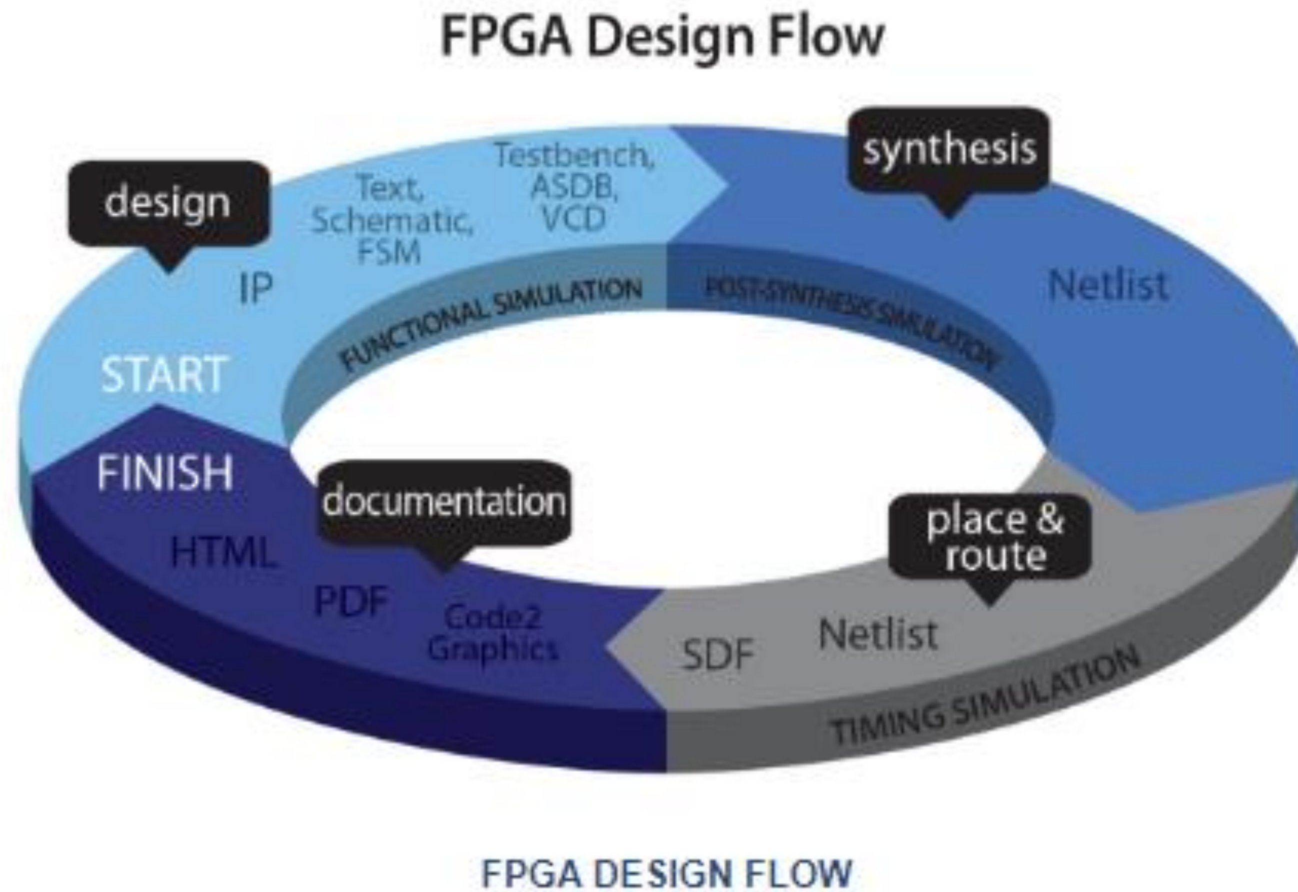




Introduce hardware accelerators that have both a high computing performance and consume less energy

1- Introduction, motivations and objectives

Motivations



1- Introduction, motivations and objectives

Objectives:

1 Propose hardware library (i.e, IPs) for artificial neural network architecture to be used in deep learning applications.

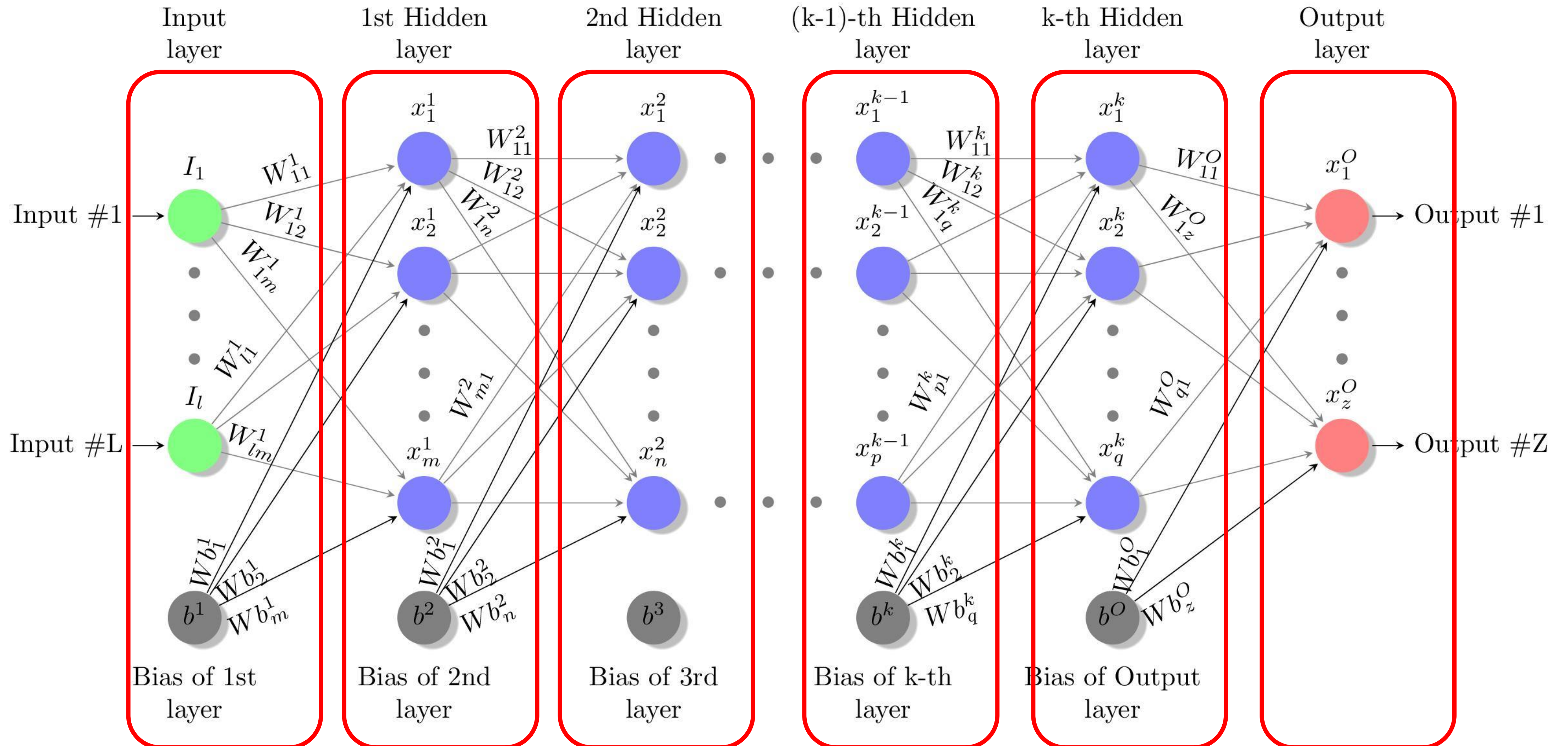
2 High level interface (i.e, Python) to customize the hardware architectures

3 Automate the process of DNN's customization, generation, and the implementation

1 Introduction, motivations and objectives
2 DNN’s hardware implementation
3 Automated Environement
4 Results
5 Conclusion
6 Perspectives

2- DNN's hardware implementation

Our framework is dedicated to building different DNN topologies, particularly supporting architectures with multiple stacked layers such as Stacked Autoencoders SAEs, Deep Belief Networks - DBNs, and Restricted



2- DNN's hardware implementation

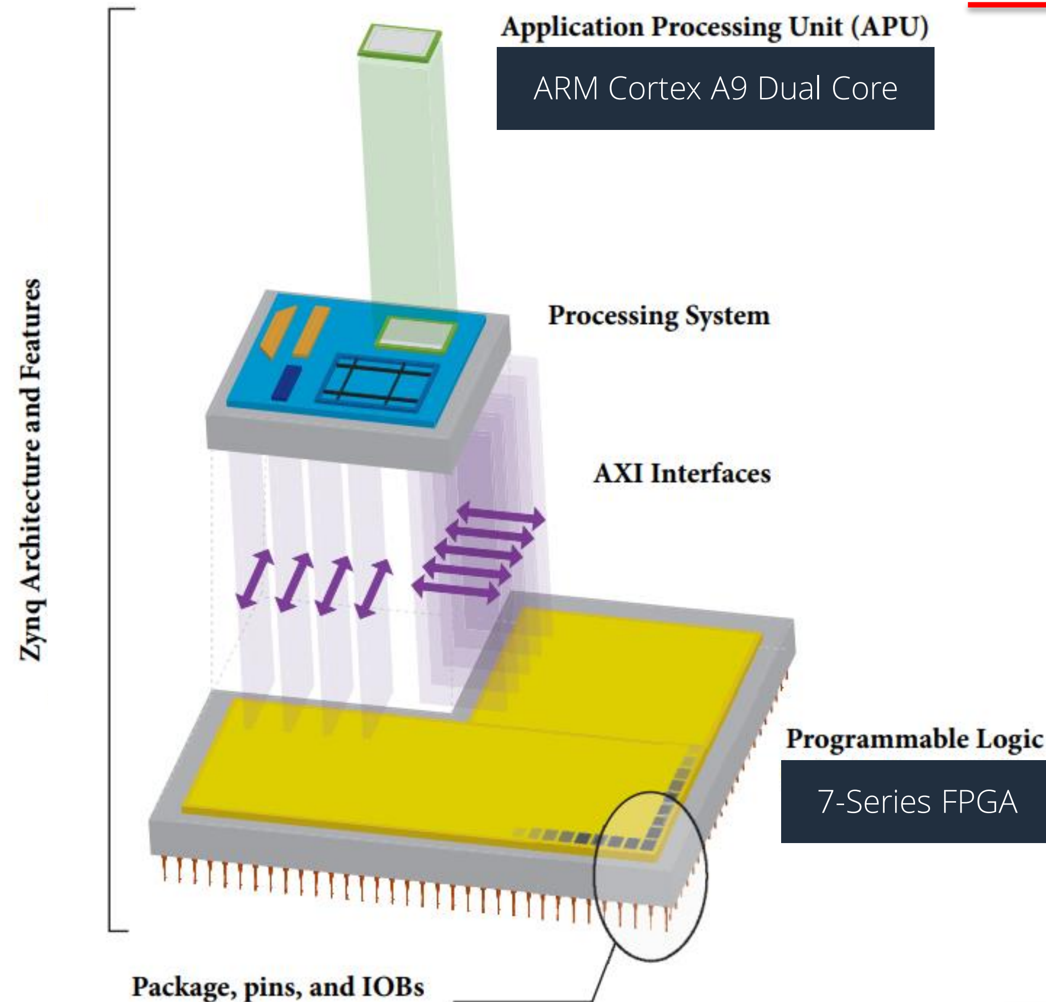
ZYNQ SoC

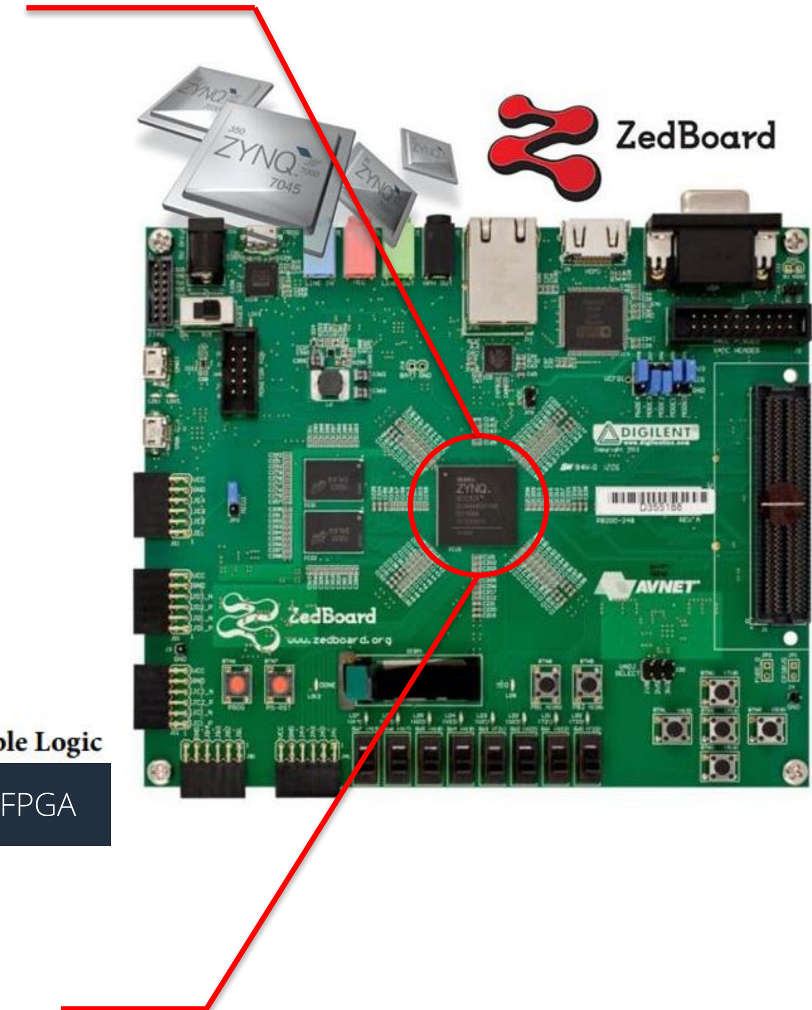
ARM Cortex-A9 Features:

- Dual Core
- CPU frequency: Up to 1 GHz
- 256 KB on-chip RAM
- Channel DMA Controller
- ARM AMBA AXI within PS and between PS and PL

7-Series FPGA Features:

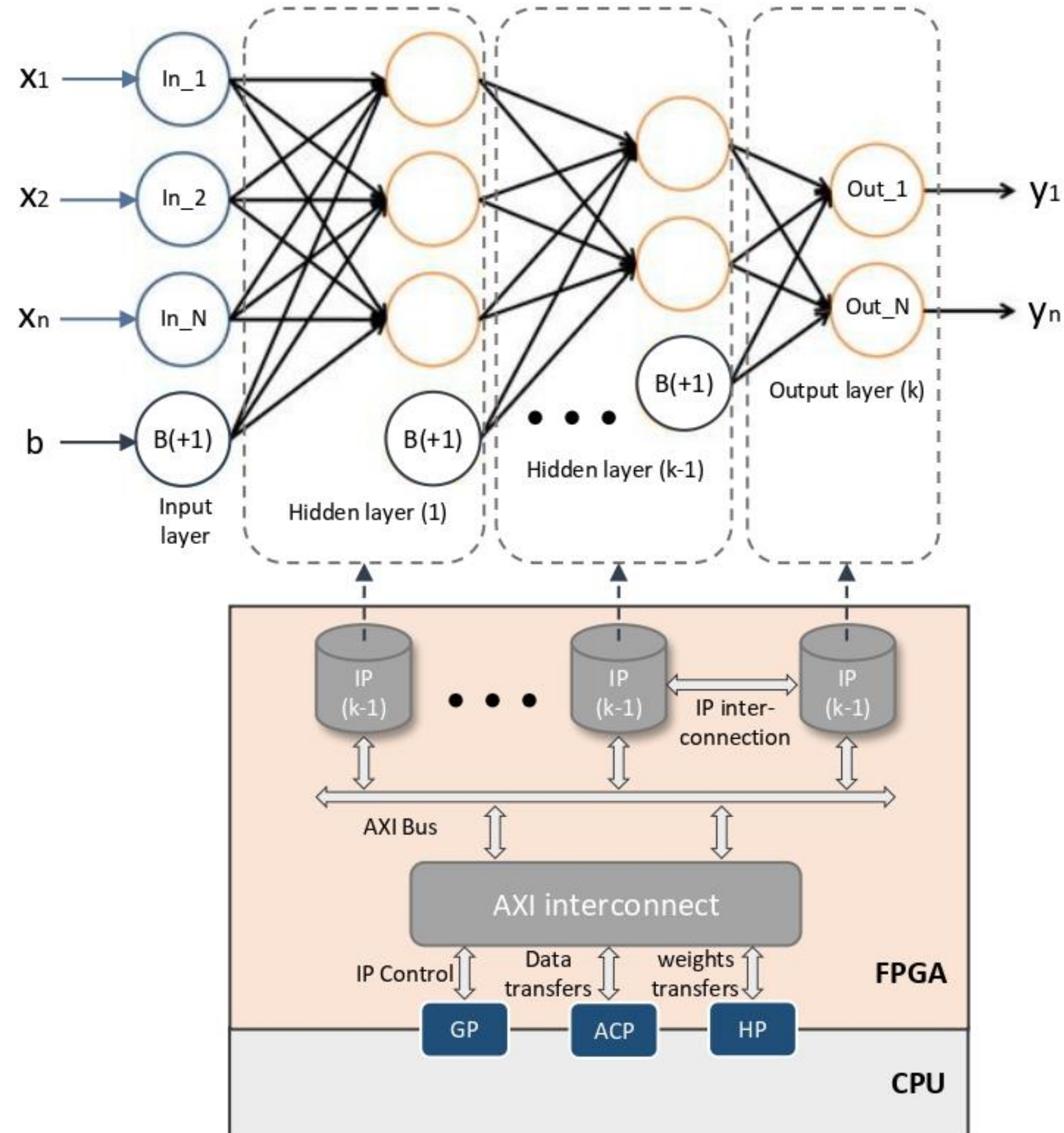
- 53k Look-Up Tables (LUTs)
- 85K Programmable Logic Cells
- 5 Mb Block RAM
- 220 DSP Slices





Source: The Zynq Book

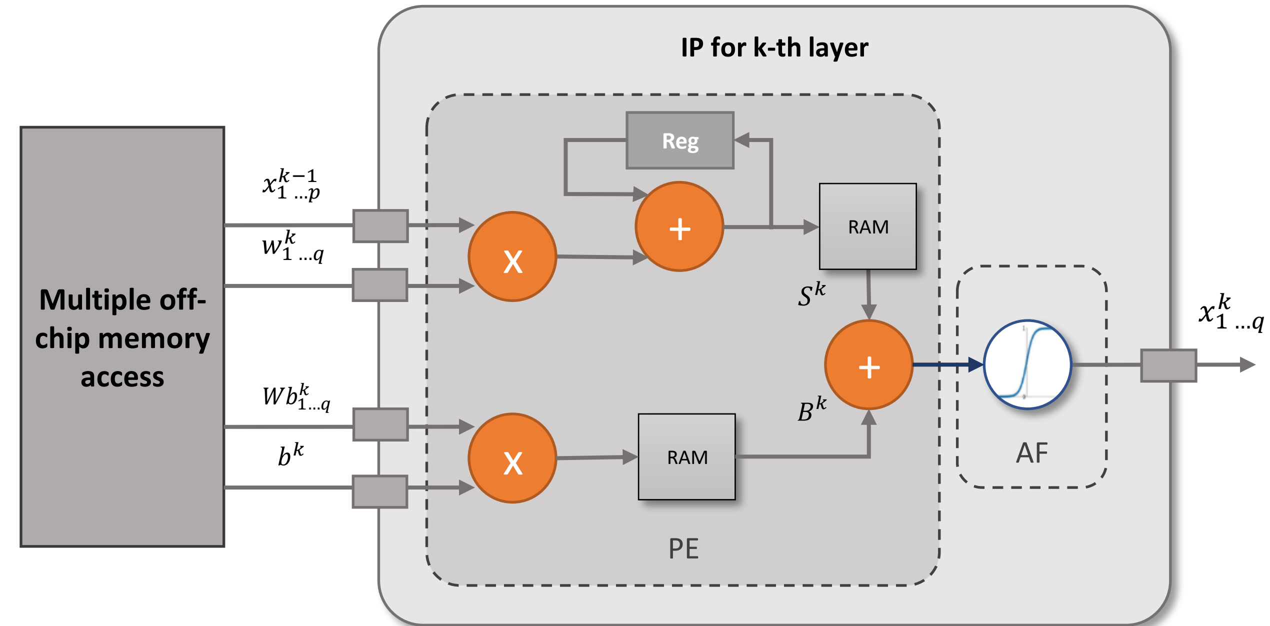
2- DNN's hardware implementation



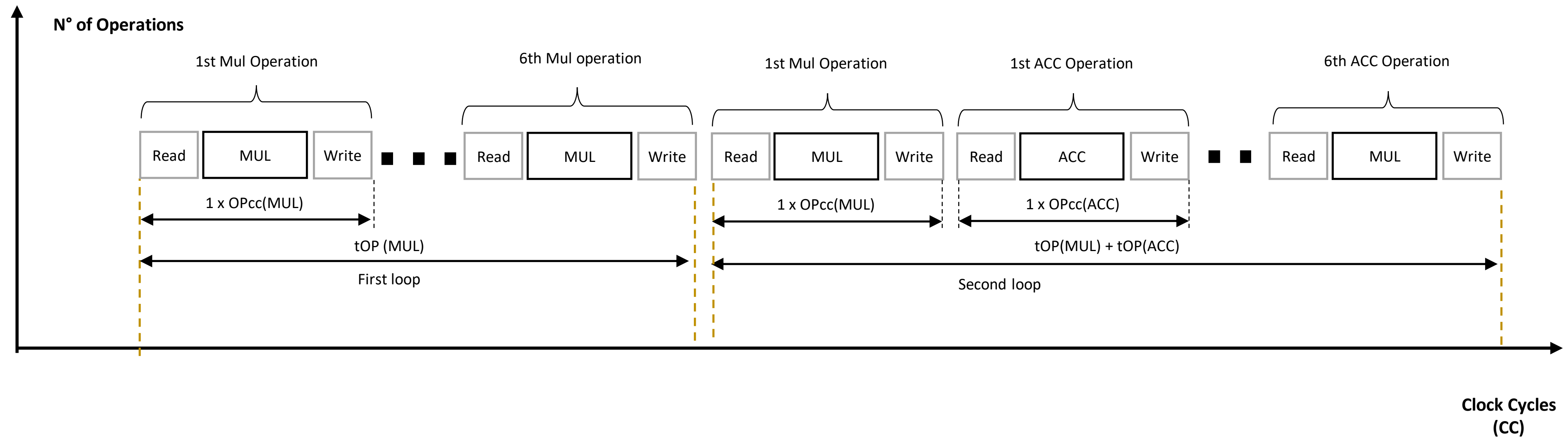
2- DNN's hardware implementation

Basic IP-layer structure. The IP includes a Processing Element PE to compute S^k and B^k values and the Activation Function AF operator. The PE is considered as a sequential series of accumulated multiplications of x^{k-1} by its associated weight W^k plus the multiplication of b^k by its associated weight Wb^k.

$$x_j^{Ln} = f(\sum_{i=1}^{U_{Ln-1}} (x_i^{Ln-1} * W_{ij}^{Ln}) + Wb_j^{ln} * b^{Ln}) = f(S_j^{Ln} + B_j^{Ln})$$

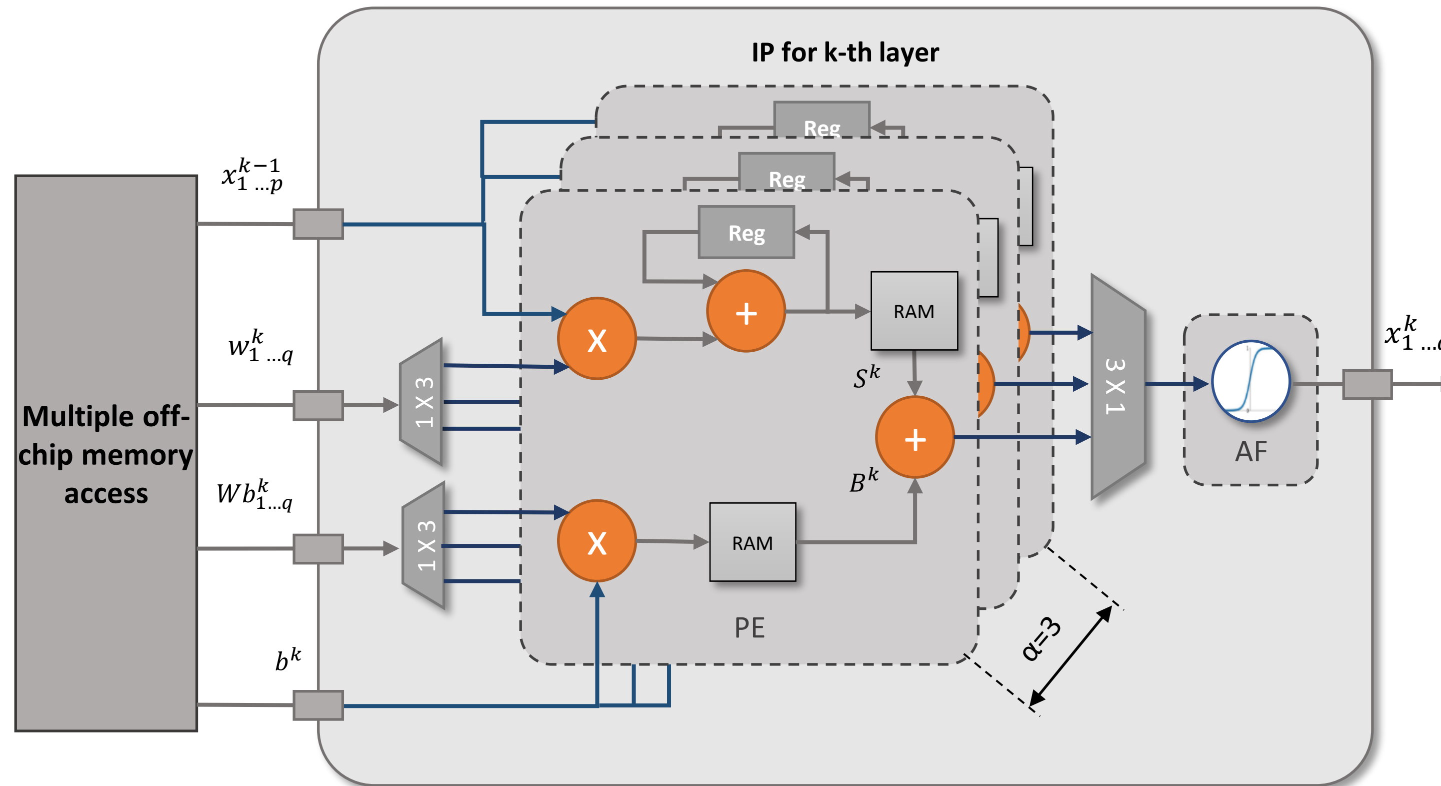


2- DNN’s hardware implementation



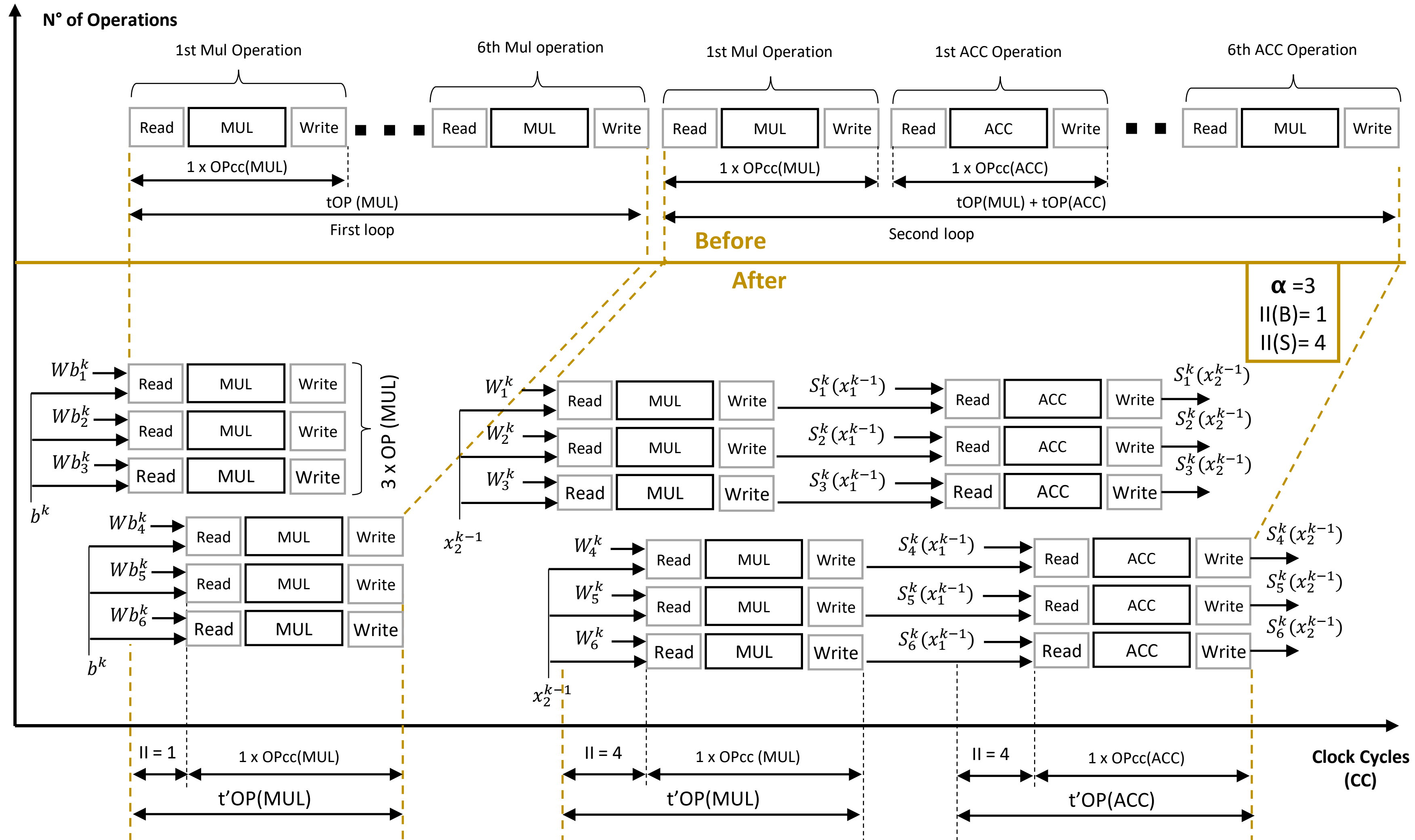
2- DNN's hardware implementation

The basic IP-layer after applying $\propto = 3$ factor



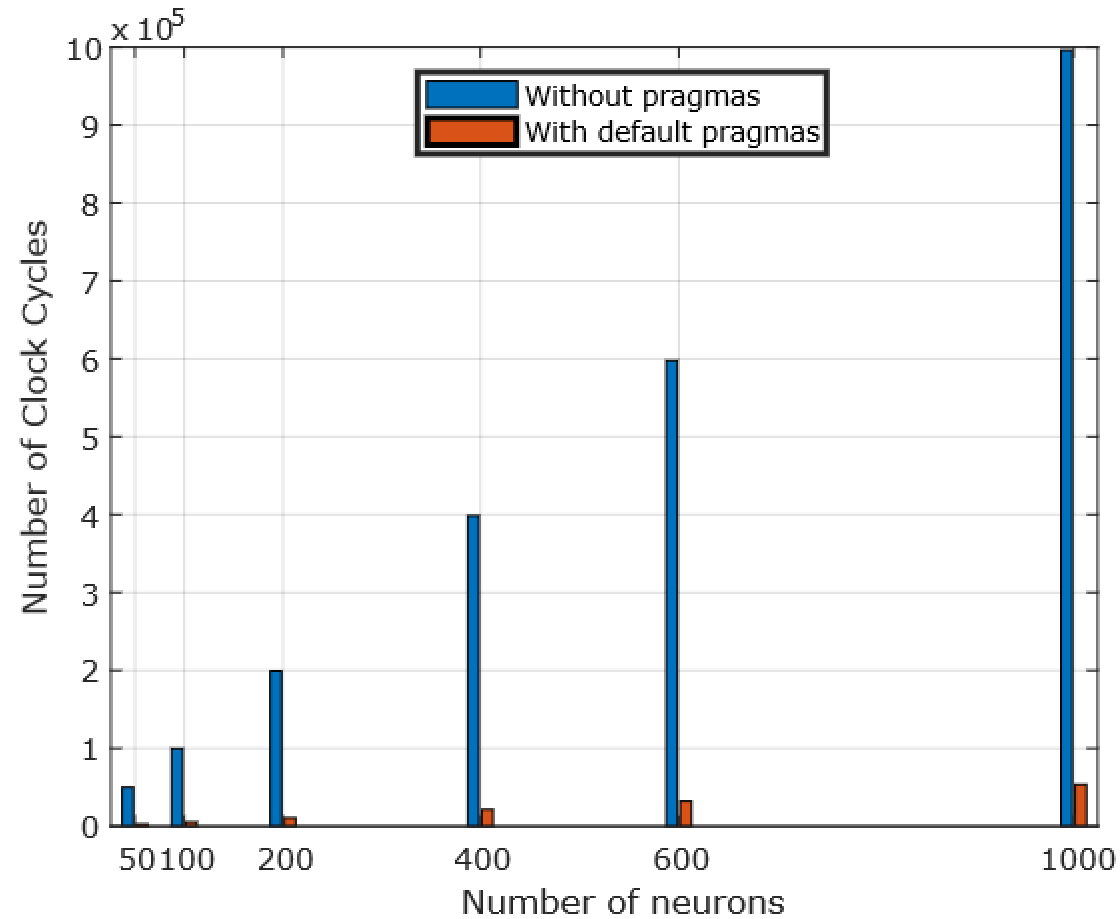
2- DNN's hardware implementation

Scheduler of processing strategy of first 6 neurons before and after applying the parallelization factor $\propto = 3$ and pipeline optimizations $II(B) = 1$ *and* $II(S) = 4$



2- DNN’s hardware implementation

Latency in clock cycles of the C++ template IP layer without (blue) and with (orange) the default pragmas generated by the proposed framework.



2- DNN's hardware implementation

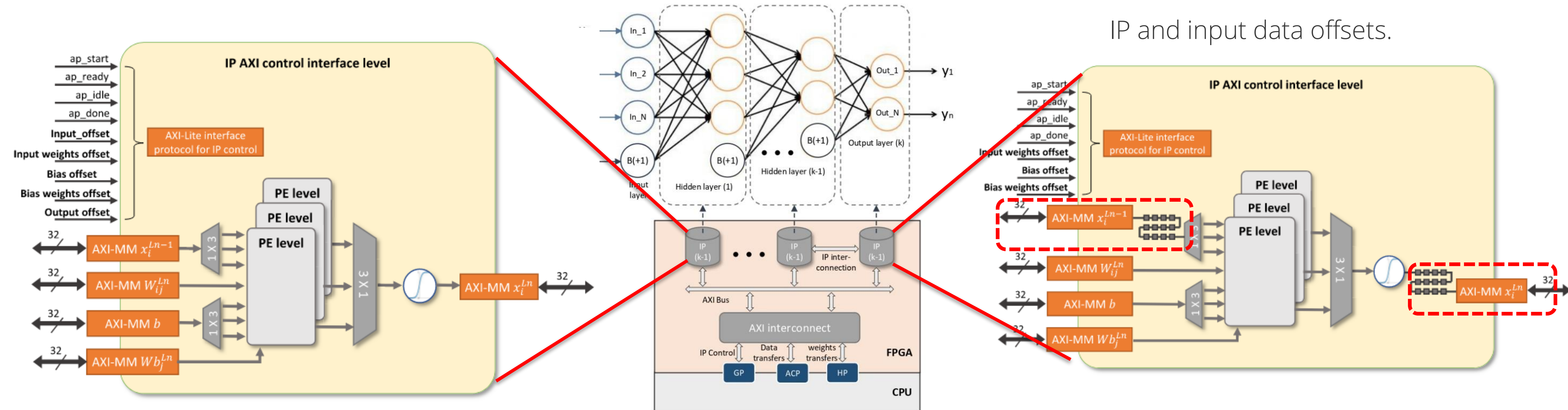
Full AXI4-MM interface: using AXI-MM slaves for inputs and

master for output. AXI-Lite controls the IP and input data offsets.

Mixed MM-Stream interface: using AXI-MM and AXI Stream

slaves for inputs and AXI-Stream master for outputs. AXI-Lite controls the

IP and input data offsets.



1. Introduction, motivations and objectives
2. DNN’s hardware implementation
3. Automated Environement
4. Results
5. Conclusion
6. Perspectives

3- Automated Environement

Pseudo-code 2: The Python API Functions to Build the DNN

Setup_Board(board_name,..)

outData layer1 = **First_hidden_layer** (inputData layer1, weight 1, bias 1, Nx)
outData layer2 = **Secon_hidden_layer**(outData layer1, weight 2, bias 2, Nx)
outData layer3 = **Output_layer**(outData layer2, weight 3 bias 3, Nx)

Configure layer(layer=1, α=2, data protocol="AXI-MM", weights protocol="AXI-MM")

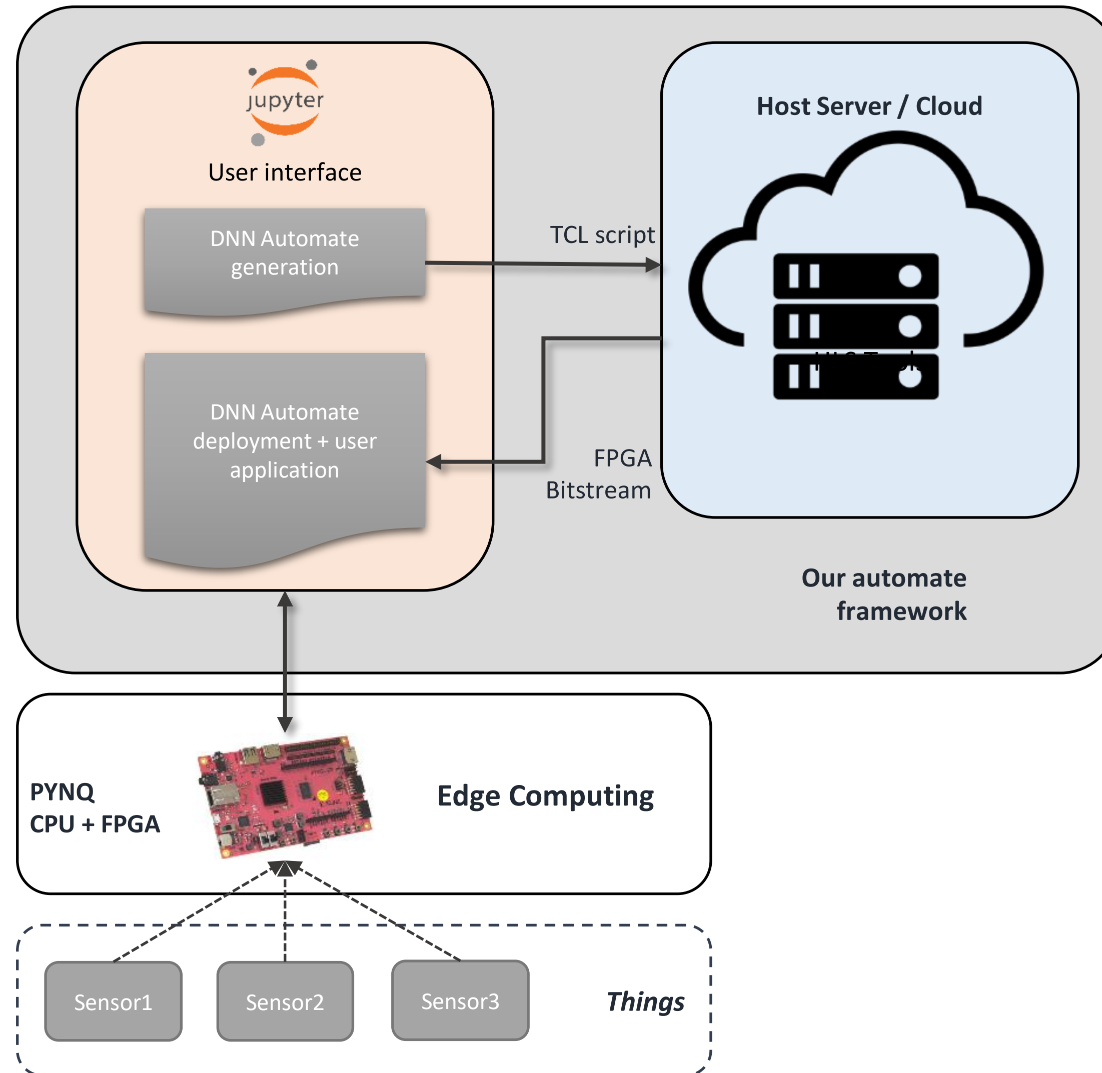
Configure layer(layer=2, α=4, pipeline=True, data protocol="AXI Stream", weights protocol="AXI-MM")

Configure layer(layer=3, data protocol = "AXI Stream", weights protocol ="AXI-MM")

Pseudo-code 4: IP With HLS Pragmas

#pragma **AXI-lite** for control
#pragma **AXI-Stream** for X(k-1-th)
#pragma **AXI-MM** for W(k-th)
#pragma **AXI-Stream** for X(k-th)
#pragma **AXI-MM** for b(k-th)
#pragma **AXI-MM** for Wb(k-th)
for $i \leftarrow 0$ **to** Q **do**
  #pragma for **partition** B **memory** by factor=α
  #pragma for **multiply MUL** by factor=α
  #pragma for **unrolling** by factor=α
  #pragma for **pipeline** $II = N$
  $B[i] = Wb[i] * b$
end
for $i \leftarrow 0$ **to** P **do**
  for $j \leftarrow 0$ **to** Q **do**
    #pragma for **partition** S **memory** by factor=α
    #pragma for **multiply MUL** by factor=α
    #pragma for **multiply ACC** by factor=α
    #pragma for **pipeline** $II = N$
    #pragma for **unrolling** by factor=α
    $S[j] += X(k-1-th)[i] * W[j]$
  end
end
for $i \leftarrow 0$ **to** Q **do**
  #pragma for **pipeline** $II = N$
  $S[i] += B[i]$
  $X(k-th)[i] = AF(S[i])$
end

3- Automated Environement

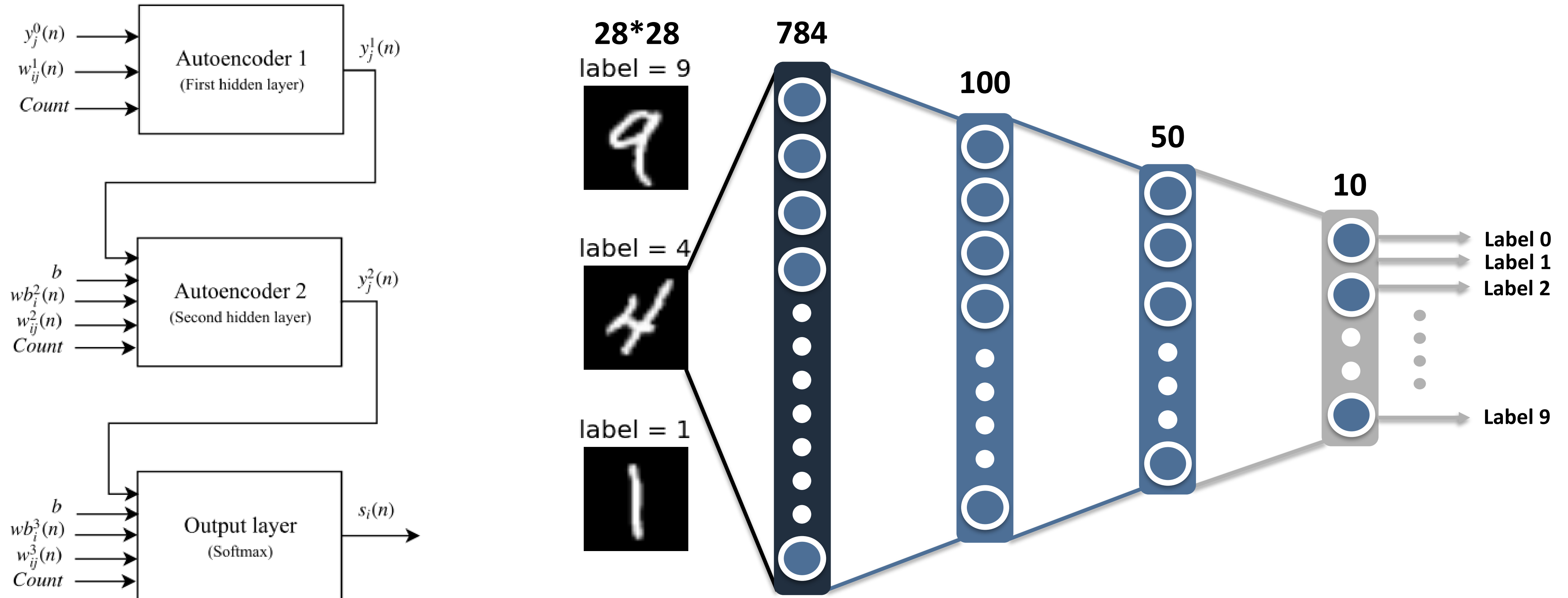


Outline

1. Introduction, motivations and objectives
2. DNN's hardware implementation
3. Automated Environement
4. Results
5. Conclusion
6. Perspectives

4- Results

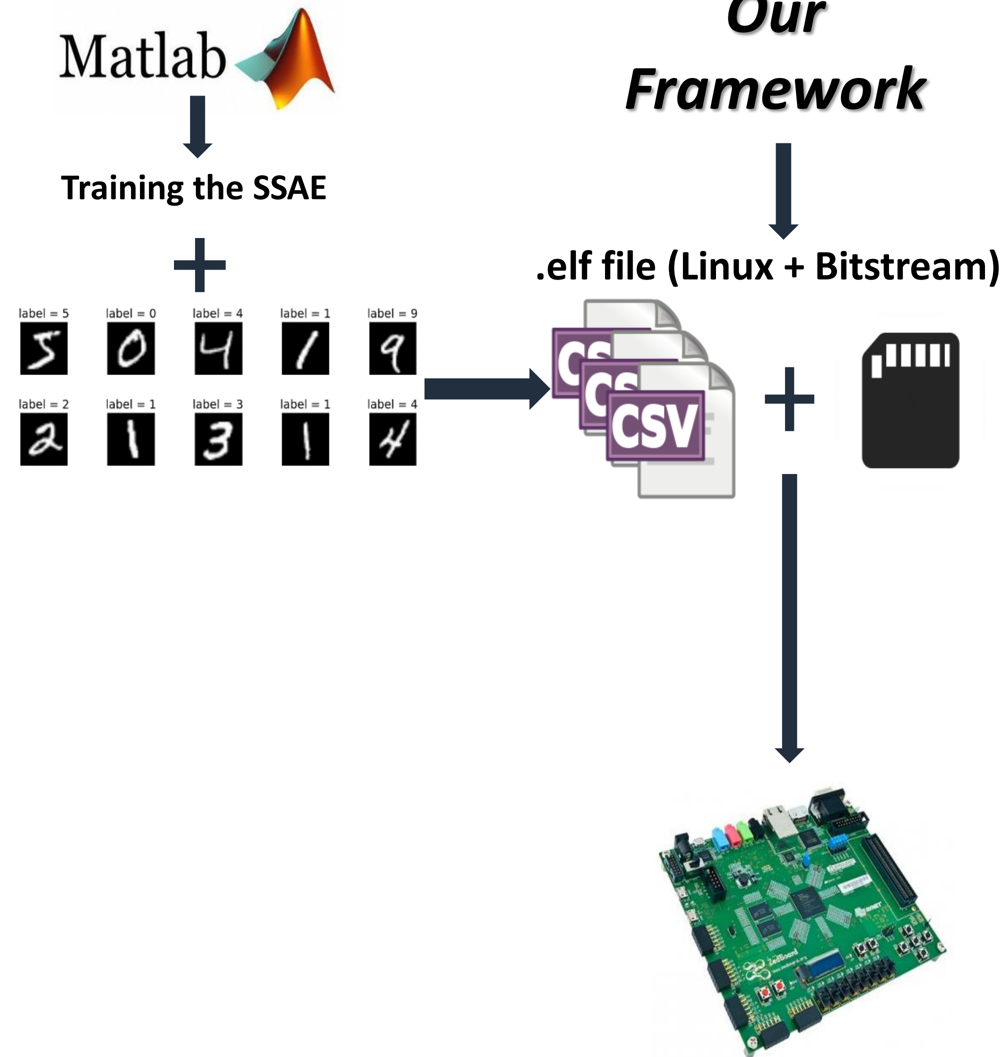
The proposed SSAE taopology presented by an input layer (784 data inputs), two hidden layers (100 and 50 inputs) and an output layer (10 outputs).



4- Results

Test bench:

1. Training the DNN on MATLAB to generate the weights and bias
2. Transform 10 MNIST exemples and results in CSV files
3. Generate full system (Linux + Bitstream) by our framework in .elf file
4. Copy the .elf and CSV files in SDcard
5. Put the SDCard in ZedBoard, run and the full system and print the results

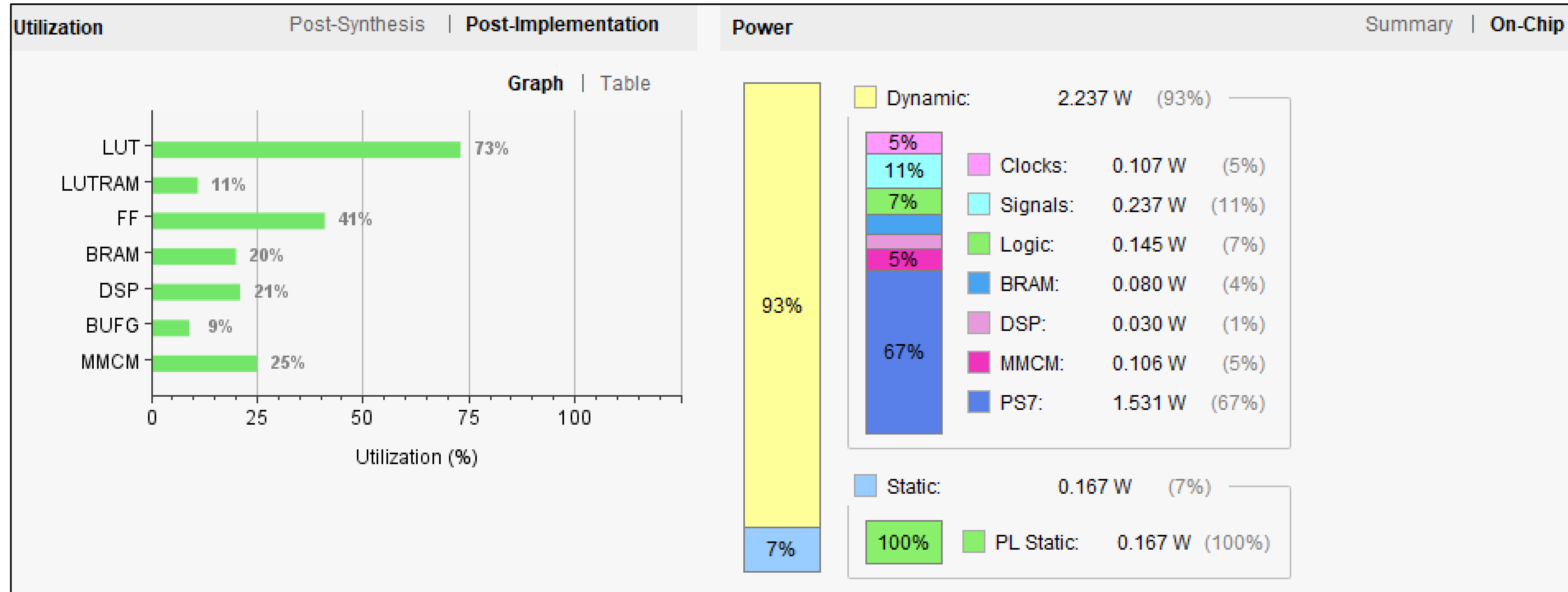


Problems | Console | Properties | SDx Log | SDx Terminal

```
Connected to: Serial ( COM3, 115200, 0, 8 )

Output[4]= 0.000000
Output[5]= 0.447000

Output[6]= 0.337492
Output[7]= 0.000001

Output[8]= 0.034521

Output[9]= 0.080442
Image 10:

Output[0]= 0.000000
Output[1]= 0.000000

Output[2]= 0.000009

Output[3]= 0.000108

Output[4]= 0.001639

Output[5]= 0.000000

Output[6]= 0.000000

Output[7]= 0.008561
Output[8]= 0.000637

Output[9]= 0.989045


***************** SOFT RESULTS ******************

(Soft) Average number of CPU cycles of Autoencoder is: 5.63263e+07
(Soft) Average time of Autoencoder is: 0.0844779


***************** HARD RESULTS ******************
(Hard) Average number of CPU cycles of Autoencoder is: 5.77783e+06
(Hard) Average time of Autoencoder is: 0.0086669


*************** SPEED UP RESULTS ****************
Speedup time is: 9.7472

Speedup number cycles is: 9.7487
```

4- Results

Hardware area occupation and power consumption



Data test set	Accuracy	Max Frequency	Time/1 image	Total On-chip Power
10000 MNIST images	99,2%	100 MHz	8,6 ms	0,380 W

4- Results

Comparison of performance between DNN topologies and three implementation alternatives auto-generated with the proposed framework, at a frequency of 100 MHz

DNN topology	Latency (ms)		Speed-up	Accuracy (%)	Power consumption (W)	Frequency (MHz)
	CPU alone	CPU + FPGA				
784-32-32-10	164.9	2.75	59.9 ×	96.2	0.266	100
784-100-50-10	514.6	8.6	59.8 ×	99.2	0.380	100
784-100-50-20-10	519	8.67	59.7 ×	99.2	0.430	100

1. Introduction, motivations and objectives
2. DNN’s hardware implementation
3. Automated Environement
4. Results
5. Conclusion
6. Perspectives

5- Conclusion

- The proposed framework solves hardware development's problems by bringing hardware acceleration closer to the software developer and user
- Our framework offers an easy way to adapt the auto-generated bitstream of the FPGA architecture with the Python user application.
- Our work presents an Edge-to-Edge Python framework for DNN- based FPGA acceleration.

1 Introduction, motivations and objectives
2 DNN's hardware implementation
3 Automated Environement
4 Results
5 Conclusion
6 Perspectives

6- Perspectives

- Our new Edge-to-Edge environment can integrate pure FPGAs design-flow to automate hardware implementation on another boards, other SoCs, and other brands by simply adopting a unique TCL template for each commercial tool-chain.

- Our future work is developing new IP templates that cover other DNN layer types to generate more popular topologies like CNNs, RNNs, GANs, among others.

Thanks

For your attention...!