Integration of neural networks on FPGAs

Giulio Bianchini

Referent teacher: Santocchia Attilio
Tutors: Mirko Mariotti, Daniele Spiga

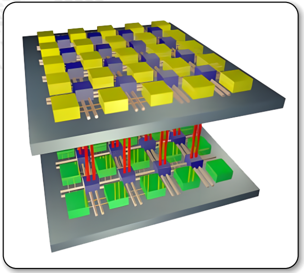
Research Doctoral programs PON for the academic year 2021/2022 - XXXVII cycle

Outline

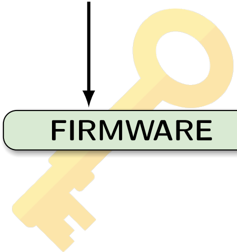
Introduction

FPGA accelerator

A field programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable.



- Parallel computing
- Highly specialized
- Energy efficient



FIRMWARE

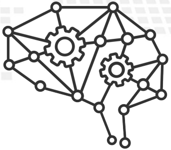
- Array of programmable logic blocks
- Logic blocks configurable to perform complex functions
- The configuration is specified with the hardware description language

Integration of neural networks on FPGA

FPGAs are playing an increasingly important role in the industry sampling and data processing.

AMD

XILINX



Deep Learning

intel

ALTERA

In the industrial field

- Intelligent vision;
- Financial services;
- Edge devices;
- Life science and medical data analysis;

In the scientific field

- Real time deep learning in particle physics;
- Hardware trigger of LHC experiments;
- And many others ...

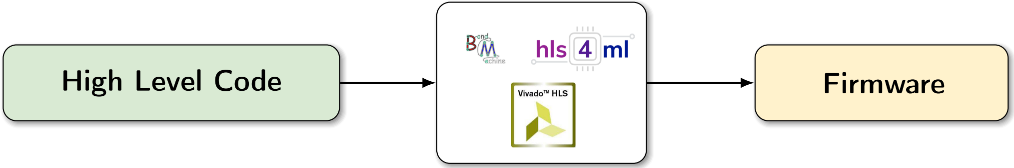
How FPGA are programmed?

Programming an FPGA is not an easy task...

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

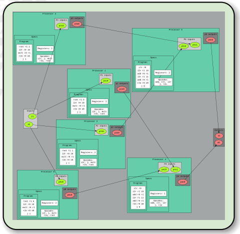
Hardware description language (HDL) represents the biggest barrier to using this device.

For this reason so many HLS (High Level Synthesis) tools has been developed.

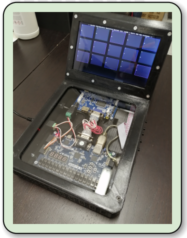


BondMachine

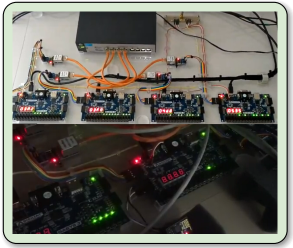
The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized on FPGA.



The main feature of the BondMachine is that processors forms an heterogeneous set of computing units.



As a standalone device



As clustered devices

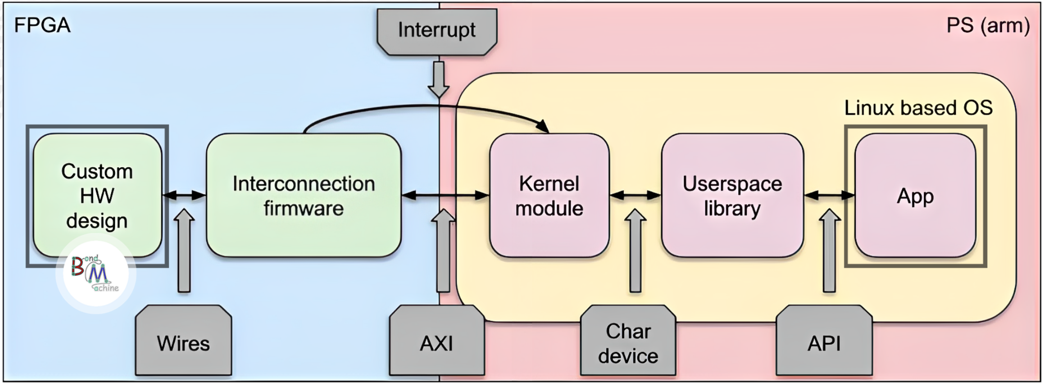
Expected results and achieved results

- Build of a system to use standard neural network models on FPGAs using BondMachine;
- Registry of pre-trained neural network models to perform a specific task that can be used through cloud services;
- Development of an accelerated system on hybrid processor;
- Benchmark of the proposed systems;

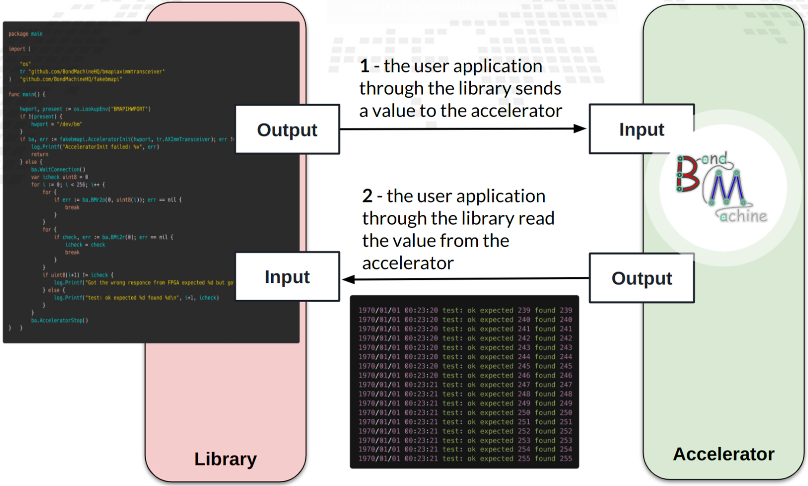
An accelerated system

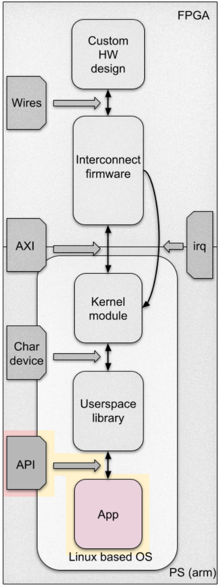
The whole accelerated system overview

Worked on the development of an accelerated system, starting from firmware up to the high level application.



Accelerated application: an example





An example

- Definition of an example

- Benchmark of the execution

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \left[c_i\right]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = [c_i]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$


```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = [c_i]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$

```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

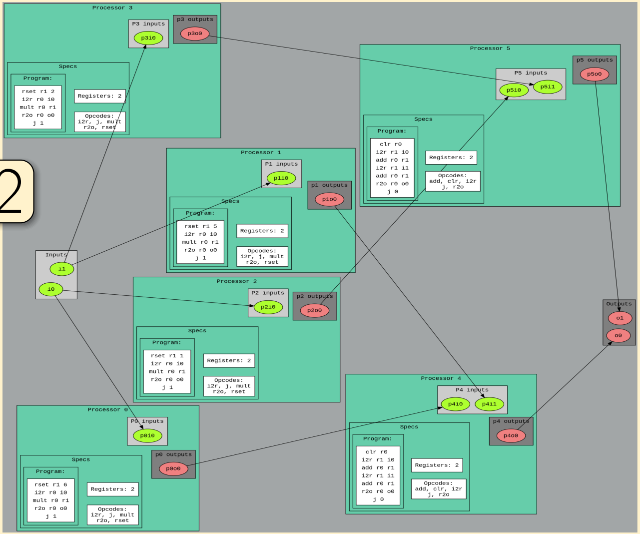
```
matrixwork -constants constants.json -constant-matrix A -numerical-type uint8 ...
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$

2x2

matrixwork



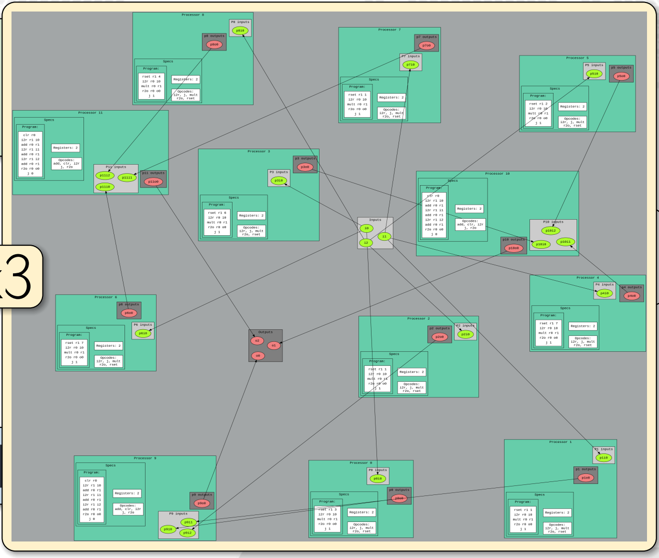

```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

Squared Matrix-vector multiplication

$$\left[\begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{matrix}\right.$$

3x3

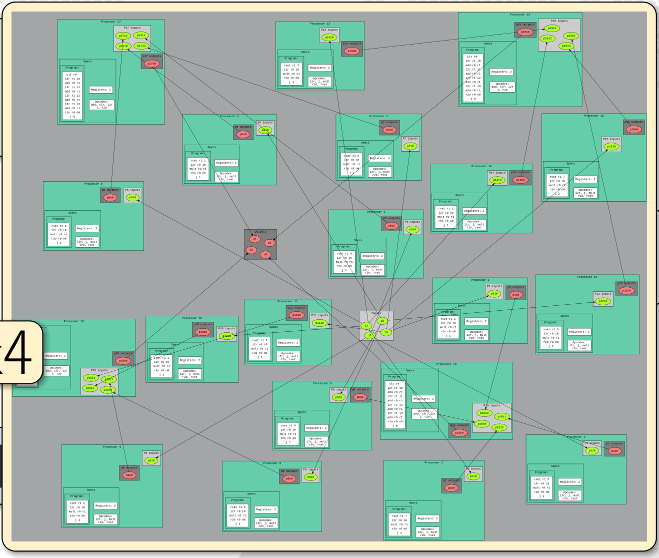
`matrixwork`




```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$



4x4

matrixwork

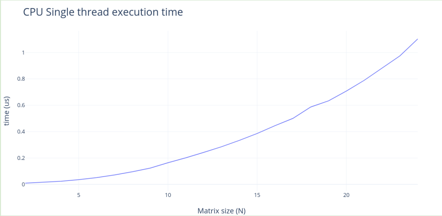

```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

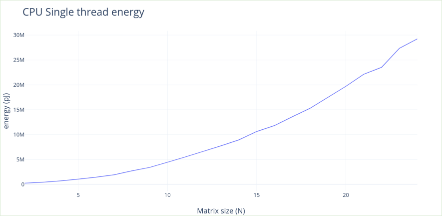
Benchmark: the CPU (Golang)

```go
func matrixtest(n int, iter int64) float32 {

//...

    start := time.Now()

    for k := 0; int64(k) < iter; k++ {
        for i := 0; i < n; i++ {
            output[i] = uint8(0)
        }

        for i := 0; i < n; i++ {
            for j := 0; j < n; j++ {
                output[i] += input[j] * matrix[i+j*n]
            }
        }
    }
    return float32(time.Since(start).Microseconds()) / float32(iter)
}
func main() {
    for i := 2; i <= 32; i++ {
        fmt.Println(i, ",", matrixtest(i, 100000000))
    }
}
```

- Time measures: built-in golang facilities
- Energy measures: perf
- Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz
- Go 1.18.2

	N	single op time (us)	single op energy (pJ)	Energy eff
1	2	0.00940269	259200	3.85802E-06
2	3	0.01631968	454200	2.20167E-06
3	4	0.02399984	722200	1.38466E-06
4	5	0.03632906	1070400	9.3423E-07
5	6	0.05176583	1471400	6.79625E-07
6	7	0.07183811	1935800	5.16582E-07
7	8	0.09597738	2737000	3.65364E-07
8	9	0.12337612	3429200	2.91613E-07
9	10	0.16450378	4465900	2.23919E-07
10	11	0.20173332	5530300	1.80822E-07
11	12	0.24265632	6643300	1.50528E-07
12	13	0.28554472	7762600	1.28823E-07
13	14	0.33400026	8954000	1.11682E-07
14	15	0.3861176	10633500	9.40424E-08
15	16	0.44550508	11832200	8.45151E-08
16	17	0.5004054	13604700	7.3504E-08
17	18	0.5867083	15324500	6.5255E-08
18	19	0.63376665	17524400	5.70633E-08
19	20	0.708154	19716100	5.072E-08
20	21	0.7893206	22133800	4.51798E-08
21	22	0.8830365	23525100	4.25078E-08
22	23	0.97467226	27346700	3.65675E-08
23	24	1.1031791	29235100	3.42055E-08

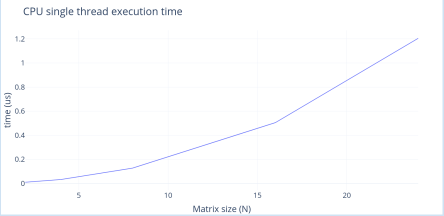


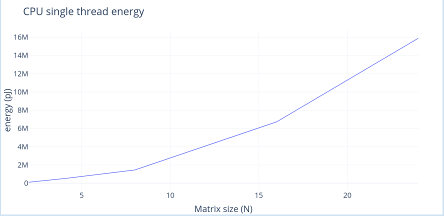


Benchmark: the CPU (C)

- Time measures: time
- Energy measures: perf
- Intel(R) CPU I5-8500 v5 @ 3GHz
- gcc with -O0

	n	single op energy (pJ)	single op time (us)	energy eff
1	2	100000	0.01	0.000008333333333
2	4	500000	0.033	0.000002702702703
3	8	1450000	0.127	0.0000005524861878
4	16	6720000	0.505	0.0000001326259947
5	24	15880000	1.205	0.00000006854009596



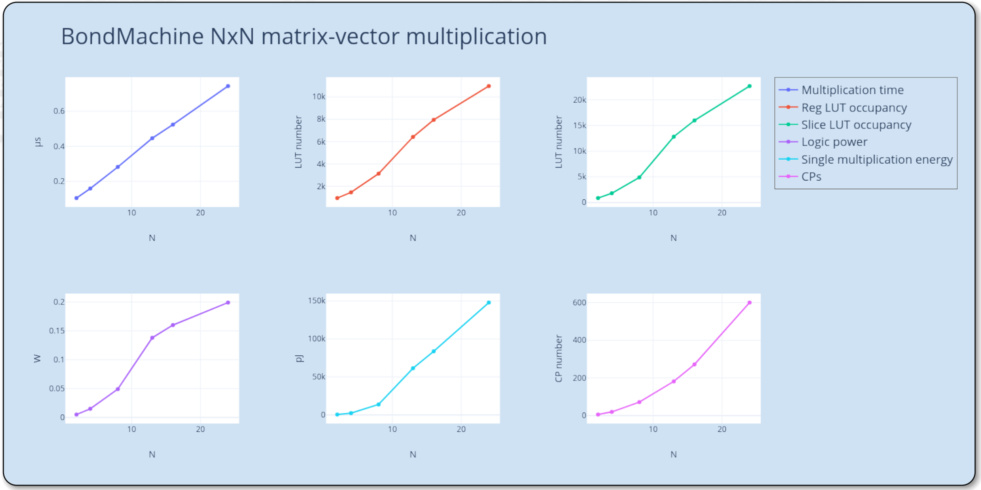


Benchmark of the accelerated system

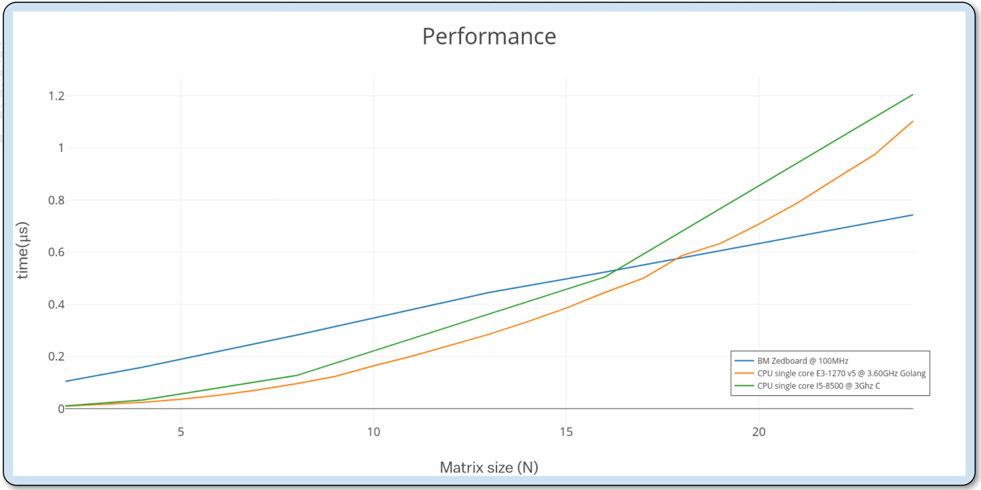
FPGA benchmark summary

	N	single op time (us)	Register LUTs	Slice LUTs	Power	single op energy (pJ)	CPs
1	2	0.1044	947	875	0.005	522	6
2	4	0.1587	1457	1813	0.015	2380.5	20
3	8	0.2819	3131	4897	0.049	13813.1	72
4	13	0.4456	6422	12819	0.138	61492.8	182
5	16	0.5234	7950	15979	0.160	83744	272
6	24	0.7432	10974	22669	0.199	147896.8	600

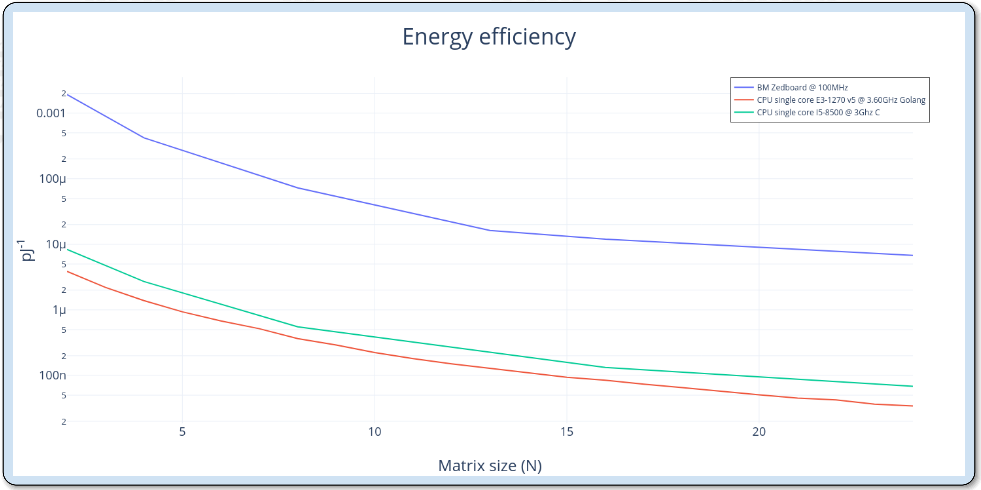
Benchmark FPGA



Comparisons: Performance



Comparisons: Energy



Machine Learning inference with BondMachine

Converting NN to HDL

What is the typical process for making FPGA inference?

hls4ml

High Level Code

HLS Conversion

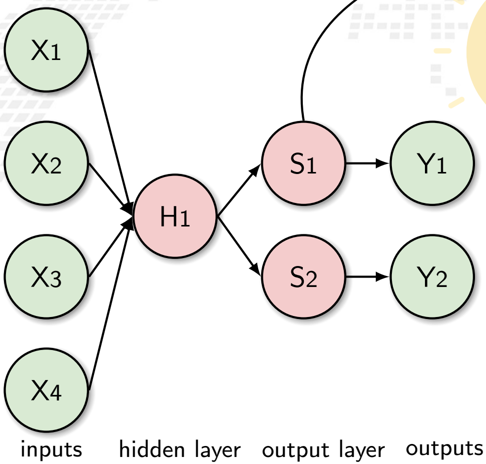
HLS Project

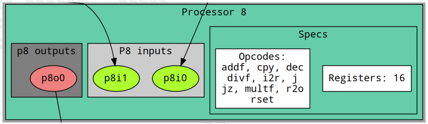
Firmware

The workflow starts by using high-level code and ML frameworks such as TensorFlow or PyTorch to train a NN model and subsequently synthesized as firmware in a complex process that involves the use of HLS tools (i.e. Vivado HLS) to generate the HDL code

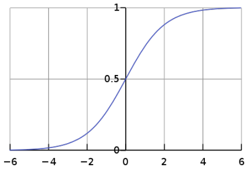
BM inference: the idea

A neuron of a neural network
can be seen as Connecting Processor of BM





$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

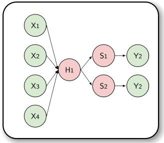



```
%section softmax .romtext iomode:sync
        entry _start    ; Entry point
_start:
 mov r8, 0f0.0
{{range $y := intRange "0" .Params.inputs}}
{{printf "i2r r1,i%d\n" $y}}
        mov     r0, 0f1.0
        mov     r2, 0f1.0
        mov     r3, 0f1.0
        mov     r4, 0f1.0
        mov     r5, 0f1.0
        mov     r7, {{$.Params.expprec}}
loop{{printf "%d" $y}}:
        multf   r2, r1
        multf   r3, r4
        addf    r4, r5
        mov     r6, r2
        divf    r6, r3

        addf    r0, r6

        dec     r7
        jz      r7,exit{{printf "%d" $y}}
        j       loop{{printf "%d" $y}}
exit{{printf "%d" $y}}:
{{$z := atoi $.Params.pos}}
{{if eq $y $z}}
 mov r9, r0
%endsection
```

From idea to implementation

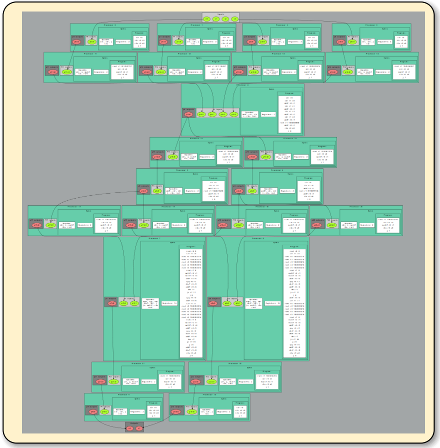
Starting from High Level Code, a NN model trained with **TensorFlow** and exported in a standard interpreted by **neuralbond** that converts nodes and weights of the network into a set of heterogeneous processors.

High Level Code




```
neuralbond -net-file banknote.json -neuron-lib-path neurons -save-basm working_dir/bondmachine.basm -config-file
neuralbondconfig.json ; basm  -o working_dir/bondmachine.json working_dir/bondmachine.basm neurons/*.basm
```

Firmware



A simple example

- A first example
- Check correctness of predictions
- Analysis of the main metrics

First test

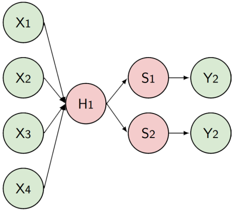
Dataset info:

- **Dataset name**: Banknote Authentication
- **Description**: Dataset on the distinction between genuine and counterfeit banknotes. The data was extracted from images taken from genuine and fake banknote-like samples.
- **N. features**: 4
- **Classification**: binary
- **Samples**: 1097

Neural network info:

- **Class**: Multilayer perceptron fully connected
- **Layers**:
  1. An hidden layer with 1 **linear** neuron
  2. One output layer with 2 **softmax** neurons

Graphic representation:



Make predictions and check correctness

```
In [138]: from pynq import Overlay
          from pynq import MMIO
          import os
          import numpy as np
          import struct
          import time

In [149]: # SETTINGS
          project_name  = "proj_ebaz4205_neuralbond_expanded"
          firmware_name = project_name+".bit"
          n_input       = 4
          n_output      = 2
          benchcore     = True

In [153]: # LOAD OVERLAY
          overlay = Overlay(os.getcwd()+"/"+firmware_name)

In [154]: # GET MEMORY ADDRESS OF IP
          bm_starting_address = (overlay.ip_dict["bondmachineip_0"]["phys_addr"])
          print(" Starting memory address of Bondmachine IP is (in dec): ", bm_starting_address)
          print(" Starting memory address of Bondmachine IP is (in hex): ", hex(bm_starting_address))

           Starting memory address of Bondmachine IP is (in dec):  1136656384
           Starting memory address of Bondmachine IP is (in hex):  0x43c00000

In [155]: # GET THE OBJECT NECESSARY TO INTERACT WITH ML IP
          spi0 = MMIO(bm_starting_address, 128)

In [156]: # LOAD BANKNOTE TESTSET
          X_test = np.load('banknote-authentication_X_test.npy')
          Y_test = np.load('banknote-authentication_y_test.npy')
          # get the first 32 samples
          X_test = X_test[0:10]
          y_test = Y_test[0:10]
          print(" Example of first two input:  ", X_test[:1])
          print(" Example of first two output: ", Y_test[:1])

           Example of first two input:   [[ 0.39886742  0.76609776 -0.39003127 -0.58781728]]
           Example of first two output:  [[ 1.  0.]]

In [157]: # IN THIS CASE I WANT TO SEND ONLY THE FIRST X INPUT SAMPLE
          idx = 0
          results_to_dump = []
          for xSample in X_test:
              offset = 0
              for feature in list(xSample): #
                  binToSend = get_binary_from_float(feature)
                  decToSend = int(binToSend, 2)
                  spi0.write_mm(offset, decToSend) # WRITE THE FEATURE TO THE CORRESPONDING INPUT
                  offset = offset + 4 # 4 BYTE = 32 BIT
              time.sleep(0.5)
              out = np.asarray(read_output())
              #print(" #",idx," -> classification: ", np.argmax(out[0:2]))
              classification = np.argmax(out[0:2])
              if (benchcore == True):
                  results_to_dump.append([out[0], out[1], classification, out[2]])
              else:
                  results_to_dump.append([out[0], out[1], classification])
              idx = idx + 1
              #break

          print(results_to_dump)

          [[0.68957698345184326, 0.31042304635047913, 0, 10402.0], [0.57489913702011108, 0.42510083317756653, 0, 10522.0], [0.
          60581505298614502, 0.39418494701385498, 0, 8728.0], [0.78091907501220703, 0.21908088028430939, 0, 5440.0], [0.692104
          93564605713, 0.30789506435394287, 0, 14502.0], [0.63284581899642944, 0.36715418100357056, 0, 12267.0], [0.6657641530
          0369263, 0.33423587679862976, 0, 6191.0], [0.69290423393249512, 0.30709570646286011, 0, 10592.0], [0.637744188308715
          82, 0.36225578188896179, 0, 11427.0], [0.67555481195449829, 0.32444518804550171, 0, 15070.0]]

In [115]: import csv
          fields = ['probability_0', 'probability_1', 'classification', 'clock_cycles']

          with open(project_name+".csv", 'w') as f:
              write = csv.writer(f)
              write.writerow(fields)
              write.writerows(results_to_dump)
```

- Thanks to PYNQ we can easily load the bitstream and program the FPGA in real time.
- With their APIs we interact with the memory addresses of the BM IP to send data into the inputs and read the outputs
- Dump output results for future analysis

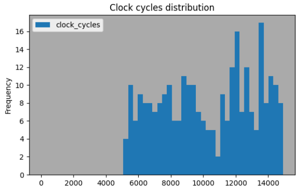
Software			BondMachine		
prob0	prob1	class	prob0	prob1	class
0.6895	0.3104	0	0.6895	0.3104	0
0.5748	0.4251	0	0.5748	0.4251	0
0.4009	0.5990	1	0.4009	0.5990	1

The output of the bm corresponds to the software output:
it works!

Inference evaluation

Evaluation metrics used:

- **Inference speed**: time taken to predict a sample i.e. time between the arrival of the input and the change of the output measured with the **benchcore**;
- **Resource usage**: luts and registers in use;
- **Accuracy**: as the average percentage of error on probabilities.



- σ: 2875.94
- Mean: 10268.45
- Latency: 102.68 µs

Resource usage

resource	value	occupancy
regs	15122	28.42%
luts	11192	10.51%

Is it possibile to **optimize** this solution? (Yes)

Tons of ways for optimizations

BondMachine is highly customizable and a lot of optimizations can be made. For example, you can scale the size of the registers (32,16,8 .. bit), choose whether to collapse processors up to customize the single neuron (and these are just some examples).

Customizing neurons

Remember the **softmax function**?

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

`K` can be customize as needed

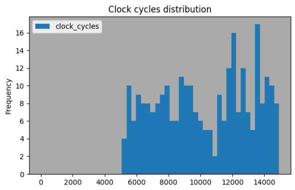
benefit → **Improves latency**

drawback → **Decreases accuracy**

tradeoff

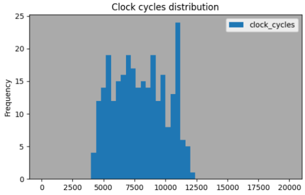
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

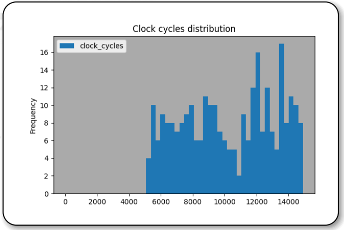


- K: 16
- σ: 2106.32
- Mean: 7946.16
- Latency: 79 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

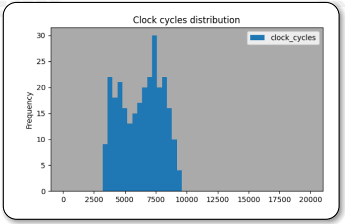
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

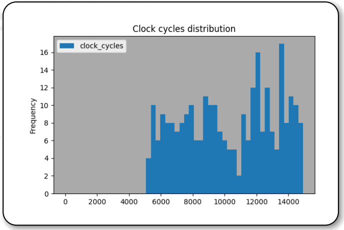


- K: 13
- σ: 1669.88
- Mean: 6312.26
- Latency: 63 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

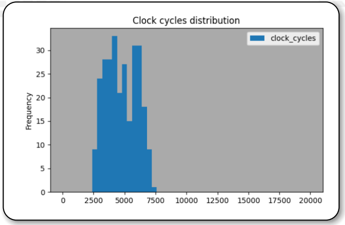
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

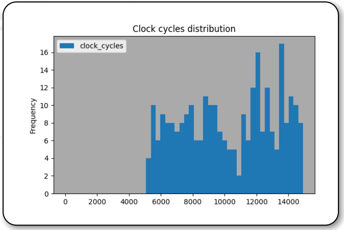


- K: 10
- σ: 1232.47
- Mean: 4766.75
- Latency: 47 µs
- Prediction: 100%

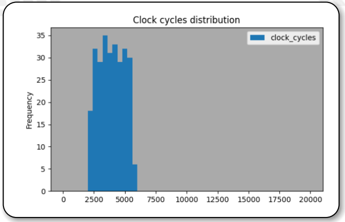
	mean	σ
prob0	$1.6162e^{-07}$	$1.1013e^{-07}$
prob1	$1.6525e^{-07}$	$1.1831e^{-07}$

Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%



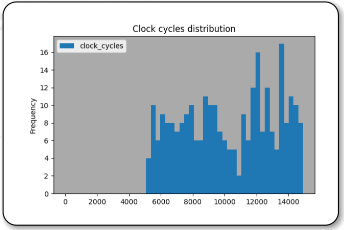
- K: 8
- σ: 1015.50
- Mean: 3913.66
- Latency: 39 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

	mean	σ
prob0	$6.5562e^{-05}$	$1.7607e^{-05}$
prob1	$6.6098e^{-05}$	$1.7609e^{-05}$

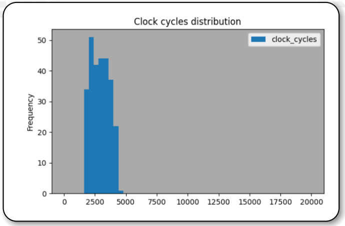
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

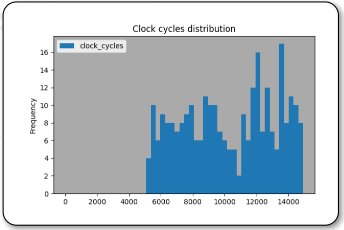


- K: 5
- σ: 740
- Mean: 2911
- Latency: 29 µs
- Prediction: 100%

	mean	σ
prob0	$3.1070e^{-05}$	$7.5290e^{-05}$
prob1	$3.1070e^{-05}$	$7.5290e^{-05}$

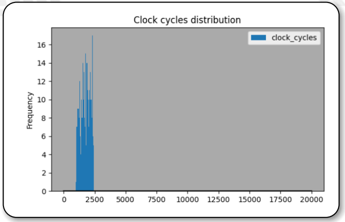
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

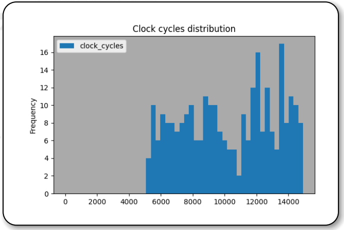


- K: 3
- σ: 394.10
- Mean: 1750.93
- Latency: 17 µs
- Prediction: 100%

	mean	σ
prob0	0.0053	0.0090
prob1	0.0053	0.0090

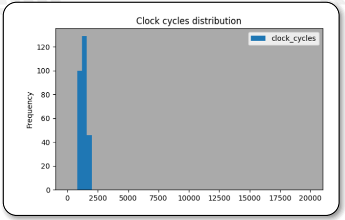
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$

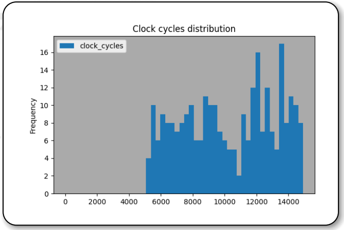


- K: 2
- σ: 268.69
- Mean: 1311.11
- Latency: 13.11 µs
- Prediction: 100%

	mean	σ
prob0	0.0193	0.0232
prob1	0.0193	0.0232

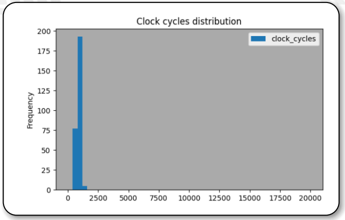
Results of optimization

Changing number of K iterations of the softmax function...



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

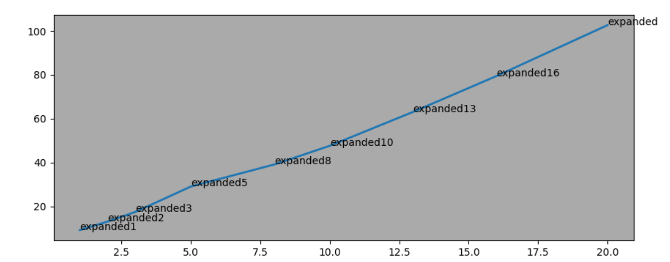
	mean	σ
prob0	$1.6470e^{-07}$	$1.2332e^{-07}$
prob1	$1.6623e^{-07}$	$1.2142e^{-07}$



- K: 1
- σ: 173.25
- Mean: 923.71
- Latency: 9.23 µs
- Prediction: 100%

	mean	σ
prob0	0.0990	0.1641
prob1	0.0990	0.1641

Results of optimization



K	Inference time
1	9.23 μs
2	13.11 μs
3	17.50 μs
5	29.11 μs
8	39.13 μs
10	47.66 μs
13	63.12 μs
16	79.46 μs
20	102.68 μs

Reduced inference times by a factor of 10 ... only by decreasing the number of iterations.

Next steps...

We are still at the beginning ...

- **More datasets**: test on other datasets with more features and multiclass classification;
- **Optimizations**: optimize the solution trying to reduce the use of resources while maintaining accuracy and inference time;
- **Integrations with existing tools**: combine the advantages of the proposed solution with other tools that make inference on FPGAs such as HLS4ML;
- **Neurons**: increase the library of neurons to support other activation functions;
- **Boards**: make the solution heterogeneous and vendor independent;
- **Evaluate results**: compare the results obtained with other technologies (CPU and GPU) in terms of inference speed and energy efficiency;
- **Training on FPGA**: train a neural network on FPGA.

Schools and courses, conferences and docs

Courses and schools:

- Machine Learning Techniques With FPGA Devices For Particle Physics Experiments (11/2022)
- SOSC2022 (11/2022)
- Third ML-INFN Hackathon: Advanced Level (11/2022)

Papers:

- Work in progress: BM Machine Learning Paper 2022

Documentation:

- https://github.com/BondMachineHQ/ml-ebaz4205
- https://github.com/BondMachineHQ/bondmachine_ebaz4205_buildroot_example
- https://github.com/BondMachineHQ/ml-zedboard
- https://github.com/BondMachineHQ/bondmachine_ebaz4205_buildroot