Realization of a perceptron in VHDL

Project for the exam of Digital System of prof. Fanucci

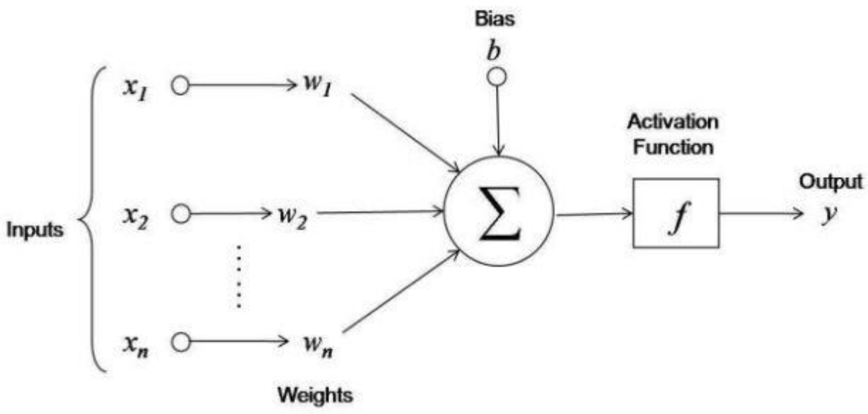
Dott. Ing. Borgioli Niccolò

Goal of the project

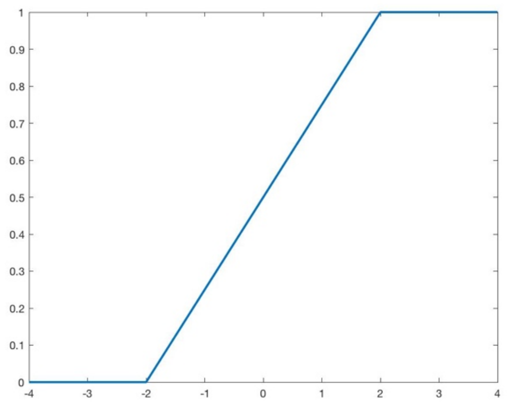
The project aims to realize a single perceptron in VHDL and make an implementation for a Zynq-7000 FPGA using the Vivado tool and analyze that implementation.

Specifications

The perceptron network takes $N_{in} = 10$ inputs x_n represented using $b_x = 8$ bits. The network multiplies each input with a coefficient w_n and adds all of the results with a bias b.
b and w_n are represented using $b_w = 9$ bits. x_n, w_n and b are in the range [-1; 1].



The result of this sum a is given as input to an activation function which tf is graphed below:



The output of this function is given as system output y and is represented by using $b_{out} = 16$ bits.

Matlab model

To have a comparison with the results of the testbench I realized a Matlab model of the activation function (file activation.m) and of the whole perceptron (file perceptron.m). I've then realized a test to check that the realized model fits the specification (file perceptron_test.m).

I've defined as a the output of the sum of all of the products with the bias and I've passed it as input to the activation function.

To build the activation function I've divided it into three different regions:

1. $a = (-\infty;\ -2]\quad y = 0;$
2. $a = [-2; 2]\quad y = \frac{a}{4} + 0.5;$
3. $a = [2;\ +\infty)\quad y = 1;$

Digital representation

To represent values in digital domain I choose to use fixed point representation, so I have to compute LSB to convert real values into discretized domain.

$LSB_x = \frac{1}{2^{b_x - 1}} = \frac{1}{2^{8-1}} = 0.0078\ ;\qquad LSB_w = \frac{1}{2^{b_w - 1}} = \frac{1}{2^{9-1}} = 0.0039$

To do product there is no need to align LSB so we can compute:

$b_{mult} = b_x + b_w = 8 + 9 = 17$ bits and have $LSB_{mult} = LSB_x * LSB_w = 3.0518 * 10^{-5}$

To get a we have to sum 11 terms, ten of which have the same number of bits (only bias have fewer bits) so the number of bits needed to represent sum output is:

$b_a = b_{mult} + ceil(\log_2 10) = 21$ bits

Consider now just the sum between products results: all of these values have the same LSB so no alignment is required. To sum the bias instead we need to align LSB:

$LSB_{sum} = LSB_{mult} = LSB_x * LSB_w = \frac{1}{2^{b_x - 1}} * \frac{1}{2^{b_w - 1}} = \frac{1}{2^{b_x - 1 + b_w - 1}}$ while $LSB_{bias} = LSB_w = \frac{1}{2^{b_w - 1}}$

So before to sum bias I have to add $b_x - 1 = 7$ zeroes bits at right of bias, then I need to add further $b_a - [b_w + (b_x - 1)] = 21 - [9 + 7] = 5$ bits at left of bias by replicating the MSB (which is the sign).

After this sum the activation function will receive as input a 21 bits value representing a, this value have to be compared with 2 and -2 to determine in which of the three regions of the activation function we are. To achieve this goal is needed to compute the representation of these two values in digital domain:

$repr_2 = ceil\left(\frac{2}{LSB_{sum}}\right) = 65536\qquad$ and $\qquad repr_{-2} = -repr_2 = -65536$

Then I have to compute the values for the output of the activation function, I compute these values with the precision allowed by the sum output (so 21 bits) and then I'll truncate to fit in y:

$repr_1 = ceil\left(\frac{1}{LSB_{sum}}\right) = 32768\qquad repr_{0.5} = ceil\left(\frac{0.5}{LSB_{sum}}\right) = 16348$

When implementing in VHDL I've further converted these values into binary representation. To perform the division by four for the linear part formula I've done two right shifts of a.

All of these computations can also be found on the attached file *bits_counts.m*

VHDL implementation

With all of the previous considerations I've wrote the VHDL code to implement the system:

- perceptron.vhd – contains the implementation of the simple perceptron
- perceptron_tb.vhd – testbench for the perceptron implementation above
- perceptron_wrapper.vhd – wrapper adding two flip flops one at the perceptron input and one to the perceptron output to make the system synchronous
- perceptron_wrapper_zybo.vhd – wrapper used to adapt the system for an implementation into a Zybo board: the only inputs becomes x, while the w and b are stucked at 0.5 (010000000). As the previous wrapper it have two flip flops: one at the input of the perceptron and one at the output to make system synchronous
- flip_flop_n_bit.vhd – flip flop used in the wrapper
- perceptron_wrapper_tb.vhd – testbench for the perceptron_wrapper_zybo implementation

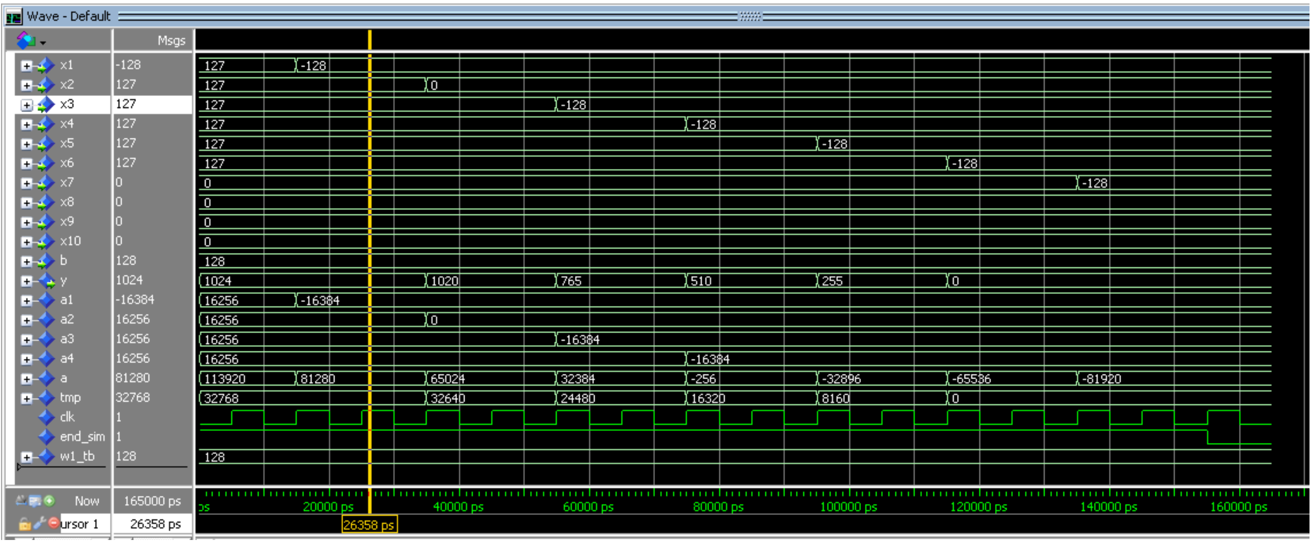
All of these files are attached to this paper and are divided in two folders contained into vhd folder: tb (contains only the testbench) and src (contains all of the other files).

Testbench verification in Modelsim

To verify the correctness of the implementation I checked the behavior of the system in with eight different inputs. These inputs are chosen in order to verify the behavior of the activation function in all of the three regions. For simplicity I have chosen to use just the first seven *x* inputs and set all weights and bias to the same value of 0.5. The testbench verification has been done directly on the perceptron model without any wrapper.

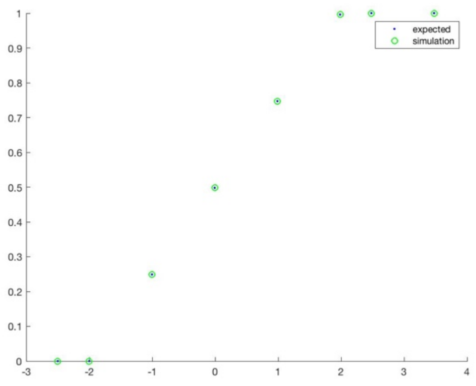
The check is done by comparing the values of *a* with the ones computed in Matlab, then the same is done with *tmp* and the output of the activation function in Matlab. The comparison is performed with *tmp* instead that with *y* because *tmp* is not truncated and so the two values should perfectly overlap.

The picture below shows the results of running the testbench on Modelsim:

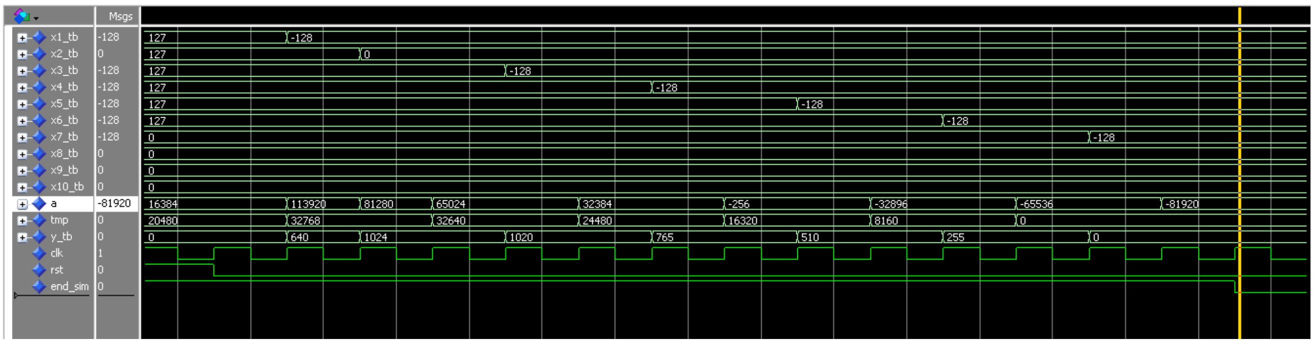

```
Wave - Default
          Msgs
x1        -128     127      -128
x2        127      127      0
x3        127      127      -128
x4        127      127      -128
x5        127      127      -128
x6        127      127      -128
x7        0        0        -128
x8        0        0
x9        0        0
x10       0        0
b         128      128
y         1024     1024     1020     765      510      255      0
a1        -16384   16256    -16384
a2        16256    16256    0
a3        16256    16256    -16384
a4        16256    16256    -16384
a         81280    113920   81280    65024    32384    -256     -32896   -65536   -81920
tmp       32768    32768    32640    24480    16320    8160     0
clk       1
end_sim   1
w1_tb     128      128

Now       165000 ps   ps  20000 ps  40000 ps  60000 ps  80000 ps  100000 ps  120000 ps  140000 ps  160000 ps
Cursor 1  26358 ps    26358 ps
```

This other picture below shows the comparison between the results of simulation and expected ones, how is possible to see they perfectly overlaps:



The same procedure is done with the testbench made for the perceptron_wrapper_zybo and as we can see from the testbench simulation below the results are the same as expected:



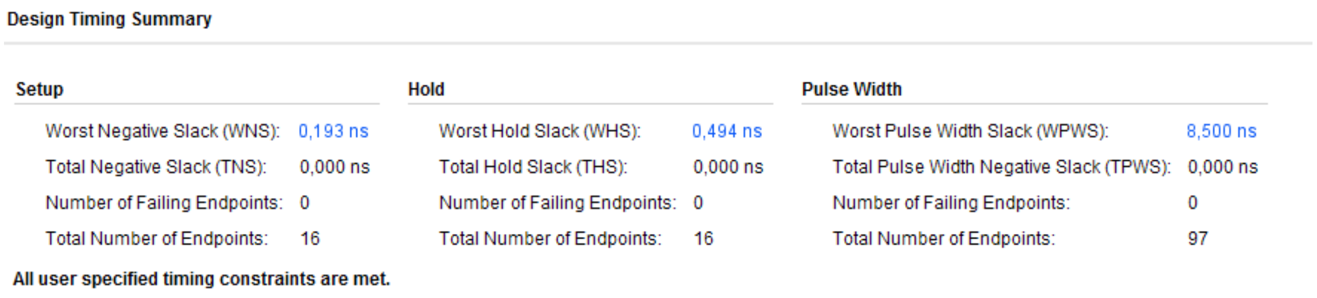
Vivado synthesis and implementation

After verification the VHDL code of the system has been imported into Vivado tool to make synthesis and implementation for a Zybo Board.

After running synthesis without any constraints to check for any error I selected 10 ns of clock period to analyze the slack and consequently adjust the clock frequency: there were a worst negative slack of about 7 ns so I choosed to increase clock period to 18 ns in order to have also a little of margin for further steps because at synthesis phase the program does not consider delay due to routing.

After that I mapped all input and outputs to package pins. Because the Zybo board have only 100 I/O pins in the wrapper I have had to reduce the number of inputs by making all weights and the bias get stuck at 0.5 and having as only inputs just the x, rst and clk and as output y, this way are used 98 out of the 100 pins available. Once mapped all signals to pins I ran again the synthesis to check for correctness in pins assignment and to verify again that the timing constraints are meet.

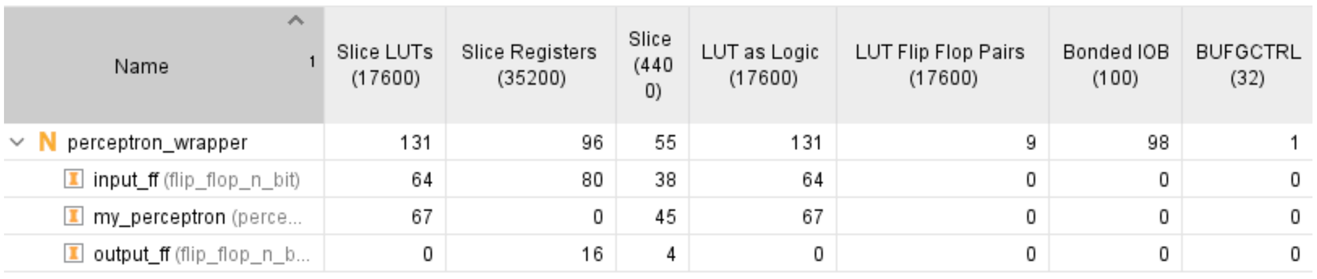
After running implementation stage the program gave no errors and I checked again that timing constraints are met (see picture below).


Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	0,193 ns	Worst Hold Slack (WHS):	0,494 ns	Worst Pulse Width Slack (WPWS):	8,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	16	Total Number of Endpoints:	16	Total Number of Endpoints:	97

All user specified timing constraints are met.

The final implementation made by Vivado just signals one warning which is due to the fact that the Arm processor which is present on the board is not used, but it's fine. In the picture below we can see the cells of the FPGA that are needed by the system:



Name 1	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT Flip Flop Pairs (17600)	Bonded IOB (100)	BUFGCTRL (32)
perceptron_wrapper	131	96	55	131	9	98	1
input_ff (flip_flop_n_bit)	64	80	38	64	0	0	0
my_perceptron (perce...	67	0	45	67	0	0	0
output_ff (flip_flop_n_b...	0	16	4	0	0	0	0

Timing and power considerations

The slack time is defined as: $T_{slack} = T_{clk} - (T_{sup} + T_{cq} + T_{pd})$

T_{sup} and T_{cq} depends on the logic and technology used while T_{pd} depends on the design. So the only thing that we can directly adjust without changing technology or design is the clock period:

$$T_{slack} = T_{clk} - (T_{sup} + T_{cq} + T_{pd}), \quad T'_{slack} = T'_{clk} - (T_{sup} + T_{cq} + T_{pd})$$

$$T_{slack} - T'_{slack} = T_{clk} - (T_{sup} + T_{cq} + T_{pd}) - [T'_{clk} - (T_{sup} + T_{cq} + T_{pd})] = T_{clk} - T'_{clk}$$

So if we want to have the theoretical maximum clock frequency allowed without having a timing violation we have to set $T'_{slack} = 0$. As we have seen the final worst negative slack computed is 0.193ns with a clock period of 18ns so theoretically the maximum clock frequency we can use is:

$$T'_{clk} = T_{clk} - T_{slack} = 17.8070\ ns \quad \rightarrow \quad F'_{clk} = \frac{1}{T'_{clk}} = 56.158\ MHz$$

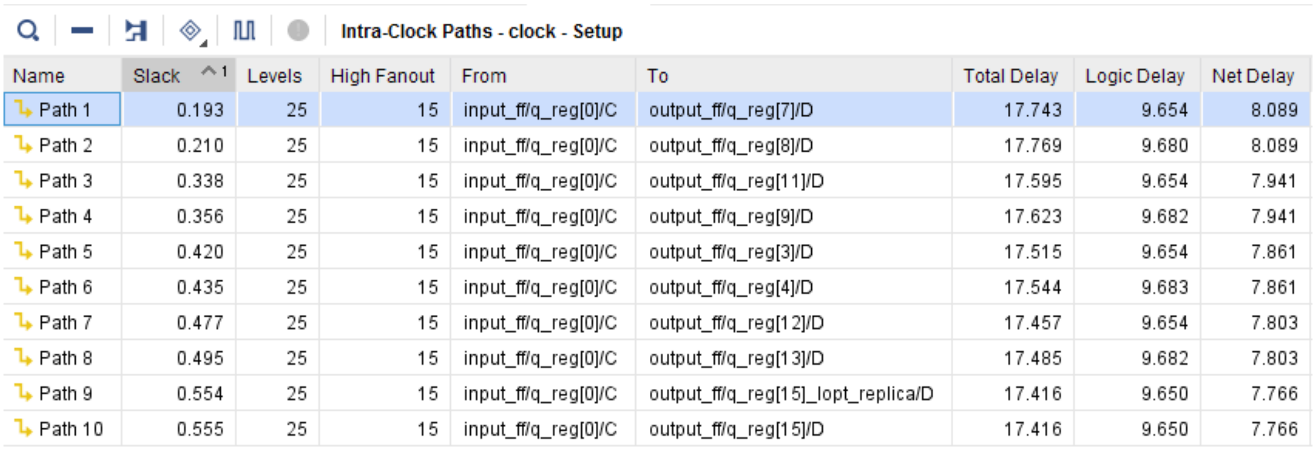
This result is just theoretical because we are not taking into account the delay due to routing which are known only when we will physically implement the logic on the FPGA.

So in the implementation I've left a clock period of 18ns in order to have a margin that would guarantee that even with the additional delay due to routing the system won't have a setup violation.

Other possible solutions to increase clock frequency of our device could be:

- choose a different technology (changing so the FPGA used) to reduce T_{sup} and T_{cq}; this will have a direct impact on costs because choosing a faster technology would be more expensive and could be not ever possible, for example in the choice of our FPGA we are also constrained by the number of I/O pins needed.
- pipeline the logic of the system by adding (for example) an additional flip flop just before the activation function. This way we will reduce the propagation delay of the critical path, so will reduce T_{pd} in spite of an increasing of the complexity of the system (additional logic is required so additional consumption).
- change the design to reduce the critical path delay; not easy.

The critical path is computed by the Vivado tool and is the one that goes from the LSB of x1 to the 7th bit of the y as we can see from the picture below:


Intra-Clock Paths - clock - Setup

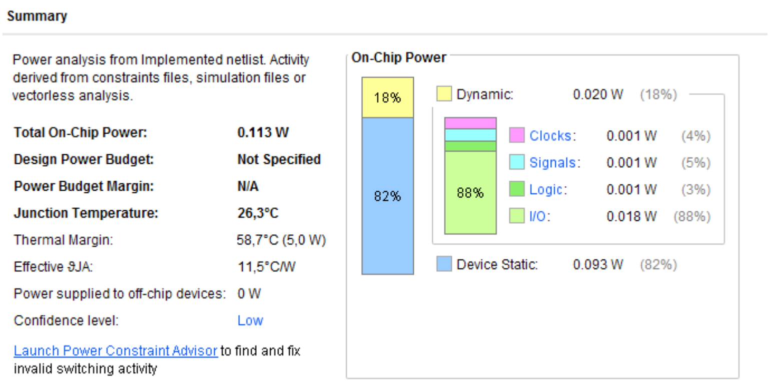
Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	0.193	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[7]/D	17.743	9.654	8.089
Path 2	0.210	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[8]/D	17.769	9.680	8.089
Path 3	0.338	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[11]/D	17.595	9.654	7.941
Path 4	0.356	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[9]/D	17.623	9.682	7.941
Path 5	0.420	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[3]/D	17.515	9.654	7.861
Path 6	0.435	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[4]/D	17.544	9.683	7.861
Path 7	0.477	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[12]/D	17.457	9.654	7.803
Path 8	0.495	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[13]/D	17.485	9.682	7.803
Path 9	0.554	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[15]_lopt_replica/D	17.416	9.650	7.766
Path 10	0.555	25	15	input_ff/q_reg[0]/C	output_ff/q_reg[15]/D	17.416	9.650	7.766

From the point of view of power consumption we have to remember that increasing the clock frequency will lead to an increase of the dynamic power consumption following the relation:

$$P_{dyn} = C_l * V_{dd}^2 * P_{0\to1} * F_{clk}$$

Moreover, as previously mentioned, if we increase the logic to pipeline the system we also have to take into account an increase of the capacitance of the system and so an increase of the dynamic power consumption.

The power consumption of the implemented system has been computed by Vivado and is:


Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.113 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,3°C**
Thermal Margin:	58,7°C (5,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		0.020 W	(18%)
	Clocks:	0.001 W	(4%)
	Signals:	0.001 W	(5%)
	Logic:	0.001 W	(3%)
	I/O:	0.018 W	(88%)
Device Static:		0.093 W	(82%)

Possible applications

This system could be used in smart environment to give an hardware acceleration for computation of neural networks. For example in a smartphone could be embedded an FPGA programmable by the processor on the fly to set the proper weights and do the needed computation; this will allow to reduce the load for the processor increasing so the overall performance because this customized logic allows to obtain lower power consumption and faster computation in calculation of activation functions with respect to a CPU. Since iPhone 7, Apple started introducing low power FPGA into his devices to run machine learning algorithms.