A NOVEL PARALLEL-SERIAL ARCHITECTURE FOR NEURAL NETWORKS IMPLEMENTED IN FPGAS

Ernest Jamro Kazimierz Wiatr
AGH University of Sience and Technollogy
al. Mickiewicz 30, 30-051, Poland
Acaddemic Computer Centre CYFRONET
ul. Nawojki 11, Kraków 30-950, Poland
jamro@agh.edu.pl wiatr@agh.edu.pl

Abstract. *This article presents a novel Parallel-Serial Architecture for Neural Networks (PSAN) optimized for digital hardware implementation, especially in FPGAs. The PSAN architecture is strongly parameterized by two parameters: the parallel P and serial S. These parameters can be adjusted independently which allows for substantial circuit optimization. The PSAN is especially efficient for multi-layer Neural Networks (NN) with different numbers of neurons in each layer, or for the NNs, for which the required throughput (calculation time) makes neither fully parallel nor serial architecture suitable. The PSAN was developed and tested for already trained feed-forward NN.*

1 Introduction

The NNs require significant computation power and therefore they are very often implemented in dedicated hardware. A significant work has been done for analog implementation of NN e.g. [1, 2]. The NNs implemented in FPGAs require different design approach, and several hardware implementation has been adopted. One of them is a serial approach [3] for which only a single multiplication per a neuron is performed at the time. Another approach denoted as time-multiplexed interconnection was presented in [4]. In this approach only a single NN layer is implemented in hardware and it is shared by different NN layers at different time slots. Hardware implementations of NN were presented in e.g. [5, 6, 7, 8], nerveless these papers usually focus on hardware implementation rather than the optimal NN hardware architecture. They also consider different aspects of NN like e.g. the Activation Function (AF).

In this paper only a feed-forward NNs are considered which architecture and weights are defined (constant) before the implementation process, i.e. at first the NN is trained on the PC and then the resultant NN is implemented in hardware. The training process might include finding the optimal number of layers, number of neurons in each layer, data width (number of bits each data is presented), and the values of weights.

In this paper several assumptions were made. Firstly, when calculating the sum of products within each neuron a constant offset value might be implemented without

multiplication. Nevertheless in this paper the offset value employs the multiplier. As a result, the offset value does not require additional memory block to be stored in.

When counting the number of NN layers the input layer is usually included. In this paper however the input layer for which no calculation is carried out is not taken into account. Summing up, the number of layers is defined by the number of hidden layers and the output layer.

At the first part of this paper the previous work is described. Then the parameters parallel *P* and serial *S* are introduced and examples of circuits for different parameters values are given. Then the optimal choice of these parameters for different throughput is considered. At the end, the example of hardware implementation is presented.

2 Standard parallel and serial NN architectures

A block diagram of the fully parallel neuron is given in *Fig. 1*. The parallel neuron calculates a basic function - sum of products for which all inputs can be changed at every clock cycle and the output is valid in the same clock cycle or with a few clock cycles latency in the case of a pipeline architecture. It should be noted that the Activation Function (AF) is inserted and will be considered

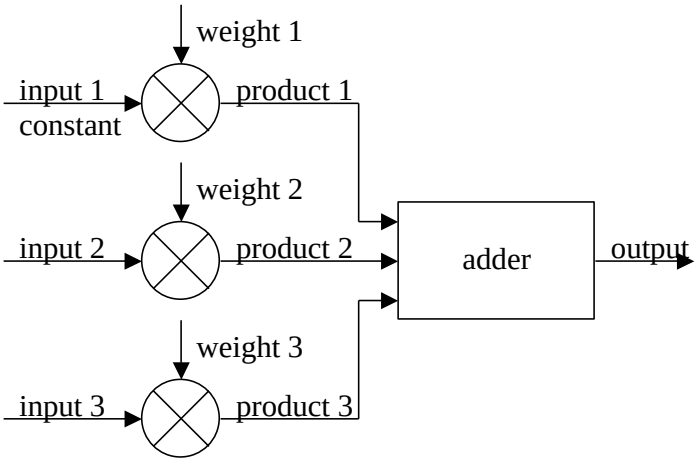

Fig. 1. Block diagram of a fully parallel neuron with 3 inputs

A whole fully-parallel Neural Network (NN) can be built from the parallel neurons presented in *Fig. 1*. An example of a single layer is given in the Fig. 2. This layer can be further used to build a whole multi-layer NN.

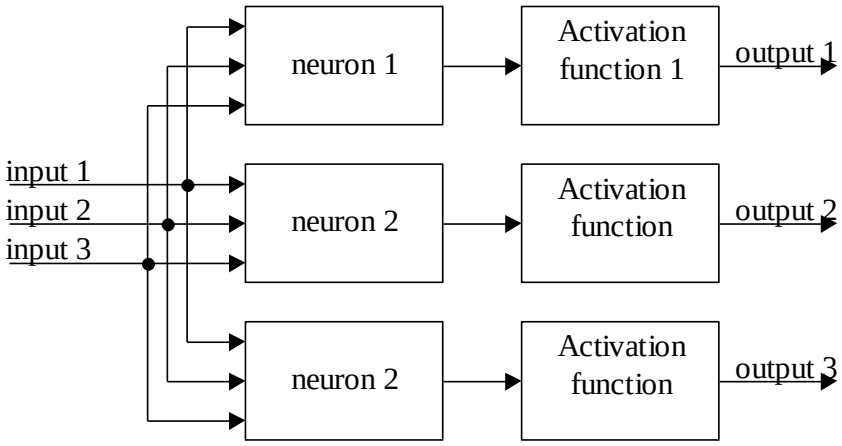

Fig. 2. A fully-parallel 3-input 3-output neural network layer

The fully-parallel NNs can be relatively easily built however they occupy a great number of FPGA resources. The number of multipliers and Activation Functions (AFs) required for a single layer is given in eq. 1.

$$n_m = n_i \cdot n_o \quad (1a)$$

$$n_{af} = n_o \quad (1b)$$

where: n_m – the number of multipliers in a single NN layer, n_{af} – the number of AF modules, n_i – the number of inputs to the NN layer (constant offset included), n_o – the number of outputs of the NN layer.

The latest FPGA contains up to 444 multipliers (Xilinx XC2VP100) however according to eq. 1a these resources can be very quickly used even by a relatively small NN. Alternatively the multipliers can be implemented in general purpose logic – Configurable Logic Blocks (CLBs), however these resources are also limited. Furthermore, the fully-parallel NN offers a great computational power which often cannot be fully used as the NN throughput is often limited by the input/output data transfers from e.g. external memory. Consequently a serial approach is often adopted. An example of a serial NN is given in *Fig. 3* and *Fig. 4*.

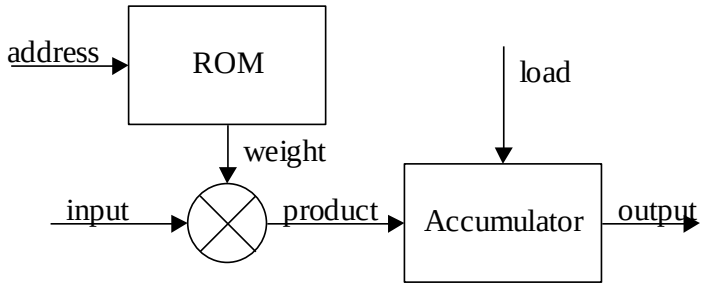

Fig. 3. Block diagram of a single neuron (serial architecture)

For the serial architecture presented e.g. in [3] there is one multiplier and one 2-input adder (accumulator) per neuron and one activation function (AF) module per layer. The inputs are fed serially to the neuron and the corresponding weights are stored in the ROMs whose address is incremented at every clock cycle to match the input and the weight index. In most cases when the number of neuron weights is relatively small (less than 32÷64), ROMs can be effectively implemented inside the Look-Up Table (LUT) memory in CLBs. The number of resources used is significantly reduced in comparison to the fully-parallel version, however to compute a single output vector, the n_i (n_i – number of inputs) clock cycles are required. The valid neurons' output are latched in flip-flops (FFs) every n_i-*th* clock cycle, and then time-multiplexed to provide proper data sequence for the next NN layer. In addition, the AF is placed at the multiplexer output. Therefore even when the considered NN layer is the last (output) layer, the multiplexer should be also employed as this reduces the number of the AFs.

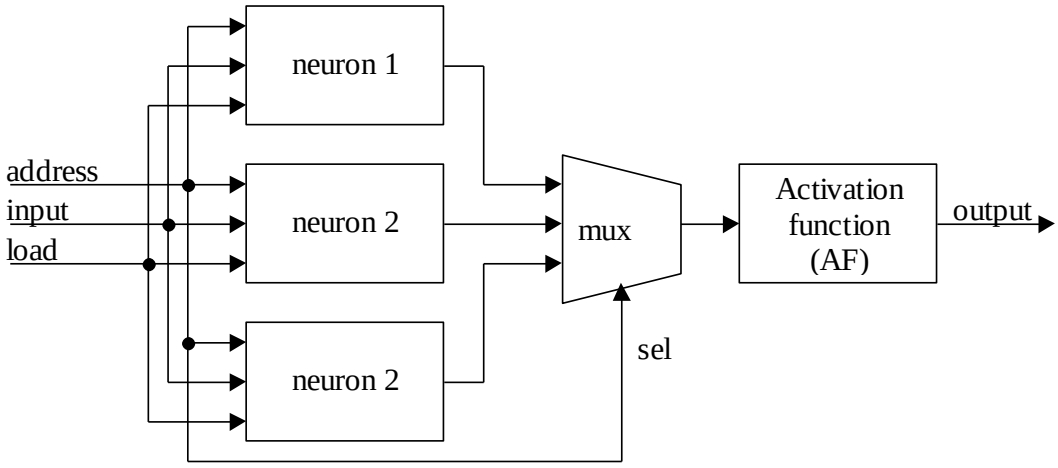

Fig. 4. Block diagram of a layer consisting of 3 neurons (serial architecture).

In the case of the multi-layer NN, the serial architecture has a severe drawback: the number of clock cycles required to compute a single output vector is limited by the slowest layer – the layer with the greatest number of inputs. The quicker NN layers must insert wait states in order to synchronize with the slowest layer.

Example 1
For a 2 – 6 – 1 NN (2 – inputs (one variable input and the offset), 6 – hidden (offset included), 1 – output node, 6 clock cycles are required to compute each output value, 2 clock cycles for the first layer and 6 clock cycles for the second layer. The first layer has to insert 4 wait states until the second layer finishes the calculation process; consequently the first layer is used only in 33% clock cycles.

The actual hardware utilization is very often specified by the external interface and its input/output throughput.

Example 2
Let us consider the previous example of the 2 – 6 – 1 NN, and 3-clock cycles external interface throughput (calculation process should take 3 clock cycles per input/output vector). The above 3 clock cycles throughput might be defined e.g. by the external memory throughput: 1- input transfer (2 clock cycles for a read) and 1-output transfer (1-clock cycle for a write). In this case neither fully parallel (1-clock throughput) nor fully-serial (6-clocks) solution is optimal.

In *Ex. 2,* a hybrid architecture might be a better solution, for which the first layer employs the serial architecture and the second layer employs the fully-parallel architecture. In this case however the first layer is used in 67% and the second layer only in 33% clock cycles. Furthermore this hybrid architecture requires activation function (AF) modules to be placed after every neuron in the first layer.

3 Novel Architecture

3.1 Introduction

In order to resolve the above drawbacks a novel Parallel-Serial Architecture for NN (PSAN) is proposed. Each layer of the PSAN is defined by two independent parameters: serial *S* and parallel *P*. The serial parameter *S* defines how many times each neuron is used to calculate a single output vector for the specified layer, i.e. instead of using *S* independent neurons (parallel or serial described in the previous section) a single neuron is used *S* – times to calculate *S* independent outputs within a single output vector. The parallel parameter *P* defines the number of multipliers implemented in a single neuron. Consequently the PSAN architecture is equivalent to the serial architecture [3] when *S=1* and *P=1*. Similarly, if *S=1* and $P=n_i$ (n_i – number of inputs – the size of the input vector with offset included) then the standard fully-parallel architecture is obtained. For $S= n_o$ (n_o – number of outputs) and *P=1* only a single multiplier is required for the whole NN layer; and this architecture works in a similar way as a multiply and accumulate (MAC) processor does.

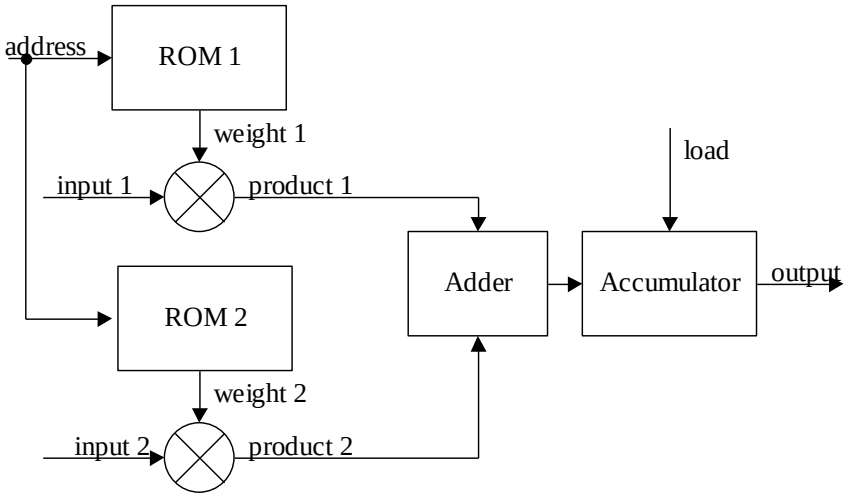

Fig. 5. Block diagram of a single neuron for PSAN (P=2)

An example of the PSAN for *S=1, P=2* for a 6-inputs 3-outputs NN layer is given in *Fig. 5* and *Fig. 6*. This architecture has two multipliers per neuron therefore two inputs has to be presented at the time. For the first input, at the first clock cycle, input data 1 is presented (constant value), at the second clock cycle input data 3, at the third clock cycle input data 5. Similarly the second input is fed with the input value 2, then 4 and 6. Accordingly the *ROM 1* stores weights 1, 3, 5, the *ROM 2* stores weights 2, 4, 6. The number of clock cycles required to calculate a single output is equal 3 and is halved in comparison to the serial architecture.

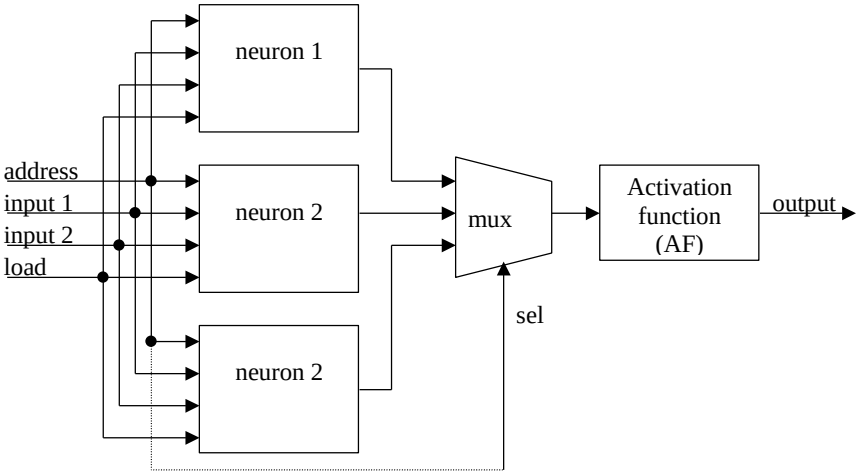

Fig. 6. Block diagram for a single layer 6-inputs, 3-outputs, PSAN (S= 1)

Let us consider another example of the same 6-input and 3-output NN layer but the different PSAN, *P=6* and *S=3.* For this architecture the whole NN layer contains only a single neuron given in *Fig. 7.* The neuron architecture is very similar to the fully-parallel version, nevertheless the same neuron is used several times to calculate three different output values for a single output vector. At the first clock cycle, the first output is calculated, therefore the weights values should be the same as for the first neuron. At the second clock cycle, the output for the second neuron is calculated and the ROMs should feed the multipliers with the second neuron weights, etc. This PSAN requires 3 clock cycles to calculate a single output vector and the input should be stable for that time. The stable input value is an advantage of this architecture as an additional Parallel In Serial Out (PISO) interface is not required as it is the case for the serial architecture.

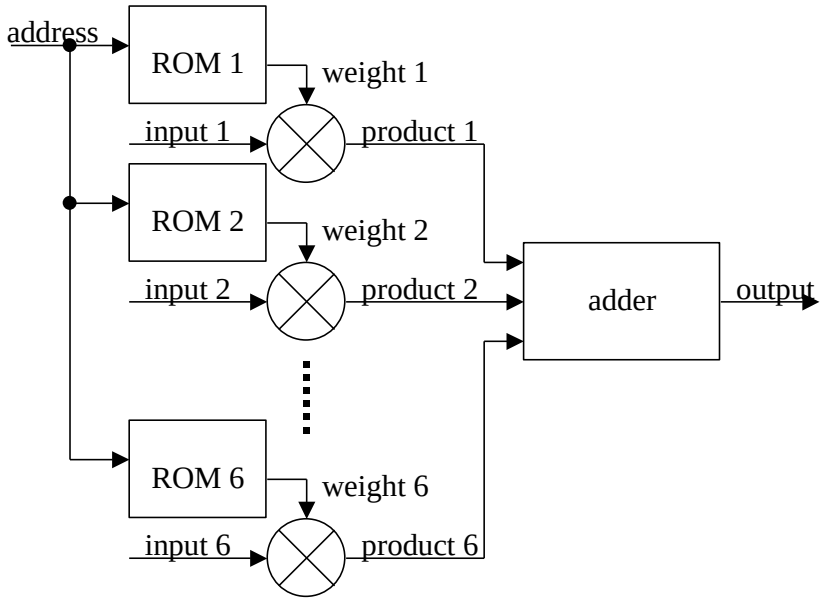

Fig. 7. The PSAN, S>1 and P=6

Let us consider another more complex example of the same 6-input and 3-output NN layer for the PSAN: *P=2* and *S=3.* This NN layer contains only a single neuron with 2 multipliers and requires 9 clock cycles to calculate a single output vector. The neuron architecture is similar as for the PSAN: *S=1*, *P=2* given in *Fig. 5*. The only difference is the

larger size of the ROMs, which must store values not only for different inputs but also for different neurons. Each ROM should store 9 different weights (3 for each neuron). The behavior of the PSAN (*S=3, P=2*) at different clock cycles is given in Tab. 1. At the first clock cycles the ROM address should be set to 1 (or to 0 when a real hardware implementation is considered), the *input 1* should be fed with the first input value, the *input 2* should be fed with the second input value, the *ROM 1* should provide the weight value for the first neuron and the first input, the *ROM 2* should provide the weight value for the first neuron and the second input, the accumulator *Load* signal should be *1*. The output provides the value of the third neuron of the previous output vector. The next clock cycles can be continued in the similar way as it is given in Tab. 1.

clock (address)	input 1	input 2	ROM 1		ROM 2		Accum. Load	Output
			neuron	index	neuron	index		
1	1	2	1	1	1	2	1	3
2	3	4	1	3	1	4	0	
3	5	6	1	5	1	6	0	
4	1	2	2	1	2	2	1	1
5	3	4	2	3	2	4	0	
6	5	6	2	5	2	6	0	
7	1	2	3	1	3	2	1	2
8	3	4	3	3	3	4	0	
9	5	6	3	5	3	6	0	
10 (1)	1	2	1	1	1	2	1	3

Tab. 1. The neuron input index, ROMs output values, Load signal and the output index for PSAN S=3, P=2

3.2 Choice of parameters *S* and *P*

The above examples explain how the PSAN operates for different parameters *S, P*. Now, let us consider the hardware requirements and calculation time for a single NN layer. The number of clock cycles required to calculate a single output vector, t_c is as follows

$$t_C = S \cdot \left\lceil \frac{n_i}{P} \right\rceil = S \cdot t_n \qquad (2)$$

where: n_i- the number of inputs (the size of the input vector, the offset is included);
t_n – the number of clock cycles required to calculate a single output value for a neuron:

$$t_n = \left\lceil \frac{n_i}{P} \right\rceil \qquad (2a)$$

It should be noted that the same calculation time t_c can be obtained by several different pairs of the parameters *S* and *P*, i.e. when both parameters *S* and *P* are increased (or decreased) proportionally the calculation time t_c is roughly constant. Similarly, the circuit area is roughly the same as far as the ratio $\frac{S}{P} = const$.

Consequently, the user has to define one of the parameters *P* or *S*. When the parameter *P* is defined, the number of neurons implemented in hardware N_n is as follows:

$$N_n = \left\lceil \frac{n_o}{\left\lfloor \frac{t}{t_n} \right\rfloor} \right\rceil \qquad (3)$$

where t – the user defined number of clock cycles required to calculate a single output vector (required design throughput).

There is a slight difference between the numbers of clock cycles t_c which are taken to calculate a single output vector and the number of clock cycles t forced by the user and defined mostly by the input/output interface requirements. When $t>t_c$ then idle cycles are inserted and therefore the circuit efficiency decreases.

The number of address locations of each ROM is the same as the number of clock cycles t_c. Each neuron is used S-times to calculate a single output vector, therefore the parameter S can be calculated as follows:

$$S = \left\lceil \frac{n_o}{N_n} \right\rceil . \qquad (4)$$

The number of multipliers (and ROMs) inside a single neuron is equal P, consequently the total number of multipliers N_m within a NN layer is as follows:

$$N_m = P \cdot N_n . \qquad (5a)$$

The only exception from eq. 5a is for the PSAN: $S=1$ and $P=n_i$ for which the fully-parallel architecture is obtained – this architecture does not require ROM memories.

The number of 2-input adders N_{2a} inside a single neuron is as follows:

$$N_{2a} = P . \qquad (5b)$$

The number of multipliers P within a single neuron cannot be greater than the number of inputs to the neuron:

$$1 \le P \le n_i . \qquad (6)$$

The parameter S shall not be greater than the number of outputs:

$$1 \le S \le n_o . \qquad (7)$$

When selecting optimal values of the parameters P and S, fractions in eq. 2, 3 and 4 should give integers – rounding functions should be avoided. Besides the number of clock cycles t_c taken by the circuit to calculate a single output vector should be the same as the number of clock cycles t forced by an user. The forced parameter t is defined mostly by the external interface or other layers of the same NN. The circuit works correctly when $t>t_c$ but this significantly influences circuit efficiency. In many cases, the external interface of the NN layer forces not optimal conditions, thus the hardware redundancy R should be defined:

$$R = P \cdot N_n - \frac{n_i \cdot n_o}{t} \qquad (8)$$

The redundancy R defines the number of implemented multipliers $P \cdot N_n$ minus the theoretical number of required multiplications $n_i \cdot n_o / t$ carried out in a single clock cycle. Alternative definition of the redundancy R_t is the total number of additional multiplications carried out by the NN layer to calculate a single output vector:

$$R_t = R \cdot t = P \cdot N_n \cdot t - n_i . n_o \qquad (9)$$

The redundancy R (eq. 8) should be used when the hardware requirements are considered. Conversely the total redundancy R_t (eq. 9) is usually employed when the whole circuit efficiency is taken into account.
By employing eq. 2 and eq. 4, the redundancy R_t can be expressed as followings:

$$R_t = \frac{t}{t_c} \cdot (N_n \cdot \left\lceil \frac{n_o}{N_n} \right\rceil) \cdot (P \cdot \left\lceil \frac{n_i}{P} \right\rceil) - n_i . n_o . \qquad (10)$$

It can be seen from the above equation that if the fractions n_o/N_n and n_i/P give integers (the ceiling function is not used) and $t_c=t$ the redundancy is equal zero. Consequently when selecting optimal value of the parameters P and S, the ceiling functions in eq. 10 should be avoided and no idle cycle $(t\text{-}t_c)$ should be inserted. In the cases when it is not possible, the parameters P and S should be selected in such a way that the redundancy R_t is minimized.

Eq. 10 can be also employed to calculate the redundancy R_t for the fully serial ($N_n=1$, $P=1$) and parallel ($P=n_i$, $N_n=n_o$) NN and can be simplified to the following formula:

$$R_t = \frac{t - t_c}{t_c} \cdot n_i . n_o \qquad (11)$$

In general, eq. 11 can be used to calculate the redundancy R_t for the PSAN for which $R_t(t=t_c)= 0$, i.e. n_o/N_n and n_i/P give integers. For example, eq. 11 can be used for the circuit with no redundancy for $t= t_c$, for which the number of clock cycles t increases but the circuit is not changed.

In some cases the relative redundancy R_1 might be employed for which the redundancy R_t is scaled by the number of required multiplications $n_i \cdot n_o$:

$$R_1 = \frac{R_t}{n_i \cdot n_o} = \frac{P \cdot N_n \cdot t}{n_i . n_o} - 1 \qquad (12)$$

The redundancy R_1 presents the number of redundant multiplications in comparison to the number of required multiplications.

In most cases the NN parameters: the number of inputs n_i, number of outputs n_o and required calculation time t are forced and only the values of the parameters P or S can be selected. In this case the algorithm given in Listing 1 might be employed in order to find the optimal P_{opt}, S_{opt} values.

Listing 1. The algorithm searching for the optimal solution

1) Initialize $P_0 = \left\lceil \frac{n_i}{t} \right\rceil$, calculate S_0 (from eq. 3 and 4), $k=1$.
2) Only a single solution $(S, P)= (n_o, 1)$?
   if $S_0>n_o$ *then* $S_0=n_o$ *and* ***finish.***
3) Initialize the optimal values $P_{opt}=P_0$, $S_{opt}=S_0$, $R_{opt}=R_t(S_0, P_0)$.
4) Next iteration: increment P_k: $P_k= P_k+1$.
5) Check for the proper P_k range *if* $P_k>n_i$ *then* ***finish;***
6) Calculate the new $N_{n,k}$ S_k (from eq. 3 and 4)
7) Eliminate insignificant solutions: *if* $N_{n,k}=N_{n,k-1}$ *then go to point 4*
8) Check S_k range: *if* $S_k>n_o$ *then* ***finish***
9) Calculate the redundancy: $R_k= R_t(S_k, P_k)$ (or the implemented circuit efficiency)
10) Select the lowest redundancy circuit:
   if $R_k<R_{opt}$ *then* $R_{opt}=R_k$, $S_{opt}= S_k$, $P_{opt}= P_k$
11) Go to the next iteration: $k= k+1$, *go to point 4.*

The point 1 of the above algorithm initialize parameters *S, P* to the minimum suitable values. The point 2 checks whether only a single solution can be found because $t \geq n_i \cdot n_o$, and only a single multiplication is carried out in the NN layer. The most important part of the above algorithm is the point 7, which eliminates insignificant pairs *(P, S)*. Increasing the parameter *P* by 1 often gives a trivial solution which does not reduces the number of clock cycle $t_n = \left\lceil \frac{n_i}{P} \right\rceil$ required to calculate an output value of a single neuron. Consequently only the number of multipliers *P* increases but the calculation time t_n is not reduced. Furthermore, even if the value of t_n decreases, the number of neurons N_n might not decrease (see eq. 3) and consequently only the number of multipliers increases.

Example 3
Let us consider a NN layer with the following external interface parameters: n_i=5 (offset included), n_o=8, t=6.
The following pairs *S, P* will be now considered:
$P_0=1, S_0=1, t_n=5, N_n=8, t_c=5, R_0=8$ - *consider*
$P_1=2, S_1=2, t_n=3, N_n=4, t_c=6, R_1=8$ – *consider*
$P_2=3, S_2=3, t_n=2, N_n=3, t_c=6, R_2=14$ – *consider*
$P_3=4, S_3=3, t_n=2, N_n=3, t_c=6, R_3=32$ – *disregard* ($S_3=S_2$)
$P_3=5, S_3=4, t_n=1, N_n=2, t_c=4, R_3=20$ – *consider*
The optimal circuit is obtained for *(P, S)= (1,1)* or *(2,2).* However as it will be explained in the next sections also pairs *(3, 3)* and *(5, 4)* might be considered for the total circuit optimization.

3.3 Input Output Interface

The choice of the parameters *S* and *P* is determined not only by the circuit redundancy *R*, but also by the input output interface. The parameters *S, P* determine the input output data sequence and therefore it is beneficial to select the parameters *S, P* in such the way that the input output data sequence converter is not required. The converter consists mostly from Serial-In Parallel-Out (SIPO), Parallel-In Serial-Out (PISO), First-In Multiple-First-Out (FIMFO) modules, which occupy relatively small area, however in the case when several pair of *(S, P)* have similar redundancy *R*, the choice might be defined by the input output interface.

For the parameter *S* greater than 1, the same data should be fed several, non-consecutive times (see e.g. Tab. 1). Consequently a special module denoted as the First-In Multiple-First-Out (FIMFO) is employed. The FIMFO is a modification of the First-In First-Out (FIFO) for which input data are stored inside the FIFO memory as in the standard FIFO. The only difference is at the output where the same data are fed *S*-times in n_i-packets. Consequently for the $n_i=4$, $S=2$ and the input data sequence:

$d_{00}, d_{01}, d_{02}, d_{03}, d_{10}, d_{11}, d_{12}, d_{13}, d_{20}$ …

the output sequence is as follows:

$d_{00}, d_{01}, d_{02}, d_{03}, d_{00}, d_{01}, d_{02}, d_{03}, d_{10}, d_{11}, d_{12}, d_{13}, d_{10}, d_{11}, d_{12}, d_{13}, d_{20}$ …

As a result, the slight modification of the FIFO output logic is required. The output counter which addresses the FIFO dual-port memory is normally incremented after every output transfer. For the FIMFO, this counter is also loaded (*S-1)*-times with the address of the first data in the sequence, i.e. with the address of data d_{00}, d_{10}, d_{20}, … Summing up, the additional hardware for the FIMFO in comparison with FIFO is insignificant.

As the result the parameter $S>1$ requires special interface and is seldom used when the input interface defines the choice of the optimal parameters S, P. The only exception is for $P=n_i$ for which all inputs should only be stable for S clock cycles – a single input value per input does not require additional data sequence conversion.
The optimum input output interface is summarized in Tab. 2. For example, for serial input interface, i.e. every input value is fed serially one at the time, the optimum PSAN solution (when the input-output interface is only taken into account) is for parameters $S=1$, $P=1$, for which the PSAN does not require input data sequence conversion.

	input	output
serial	$S= 1, P=1$	$S= n_o$
parallel	$P= n_i$	$S=1$

Tab. 2. Input output interface and the choice of the parameters *S, P*

The input-output interface is often also defined by the activation function (AF) modules which hardware requirements are considerable. If the AFs are placed at the NN layer outputs, the serial output interface is preferable as only a single AF module is required. Furthermore, even if the output interface is parallel, it is still worthy to implement a PISO, a single AF and a SIPO module in order to reduce the number of AF modules.

3.4 Multi-layer PSAN

Motivation for the PSAN

The primary advantage of the PSAN is that parameters S and P are independent and can be adjust accordingly to the user needs. This argument is particularly important when multi-layer NN is considered.

Let us define a number of nodes of the NN: n_0 – the number of inputs to the whole NN and number of inputs to the first PSAN layer 1 (offset included), n_j*-1* – the number of outputs from the layer j; n_j – the number of inputs to the layer $j+1$ (offset included), n_L – the number of outputs from the whole NN and from the last layer L, L- the number of the PSAN layers.

Example 4

Let us consider the example of the 2 – 6 – 1 NN ($n_0= 2$, $n_1= 6$, $n_2= 1$, $L=2$) already given for the hybrid serial-parallel NN in example 2. The required calculation time $t=3$. Consequently for the first layer: $n_{i1}=n_o= 2$, $n_{o1}=n_1-1=5$, for $P_1= 2$ employing eq. 3 and 4 we obtain: $N_{n1}=2$, $S_1= 3$. Consequently, employing eq. 9, the redundancy for the first layer is: $R_{t1}=2$. Similar results can be obtained for the second layer and are given in Tab. 3. As a comparison also the fully parallel and hybrid (serial for the first layer and parallel for the second layer) results are presented in Tab. 3. It can be seen that the PSAN architecture performs only 2 redundant multiplications per calculation cycle in comparison to the other architectures which require 32 (parallel) and 17 (hybrid) redundant multiplications. The number of required multiplications is 16 per calculation cycle. It should be noted that for the some multiplications are not nese

Archit.	layer 1				layer 2				total
	P	N_n	S	R_t	P	N_n	S	R_t	R_t
PSAN	2	2	3	2	2	1	1	0	2
Parallel	2	5	1	20	6	1	1	12	32
Hibrid	1	5	1	5	6	1	1	12	17

Tab. 3. Results for the 2 – 6 – 1 NN, for PSAN, parallel and hybrid solution given in example 2.

each NN layer is used in every clock cycle, the circuit has no redundancy *R* and no input output data sequence conversion is required. It should be noted that the same example was considered for the fully serial and parallel NN, and these NNs have significant overheads.

Input output interface

The interface between different layers of the NN plays the key role when selecting optimal *S, P* parameters. When the AF is considered the preferable output interface is serial, which requires $S= n_o$. Nevertheless for $S>1$ and $P<n_i$ input data sequence is more complex than for other options (FIMFO should be used).

Let us consider now a very basic NN: $n=n_0=n_1=n_2=n_3$, and $t= n$ for which the optimal architecture is shown in Fig. 8. Two different options are considered: parallel Fig. 8a and serial Fig. 8b and Fig. 8c. For the serial option data sequence conversion can be implemented either after the neurons layer ($S=1$, $P=1$) (Fig. 8b) or before the neurons layer ($S=n_o$, $P=n_i$) (Fig. 8c). It should be noted that architecture in Fig. 8b is identical with the serial NN.

Both architectures presented in Fig. 8a and Fig. 8c have one idle clock cycle per a calculation cycle ($R_t=n_i$) as the constant offset value need not be calculated ($n_i= n_o+1$), this does not apply to the last layer. Consequently the architecture presented in Fig. 8b is slightly more efficient in comparison to the architecture presented in Fig. 8c.

In this paper the offset value is calculated in a separate clock cycle, nevertheless offset can be implemented as accumulator initial value and consequently no additional clock cycle is required to calculate the offset. The latest approach results that $n_i=n_o$ for Fig. 8 and consequently Fig. 8b and Fig. 8c are equally efficient.

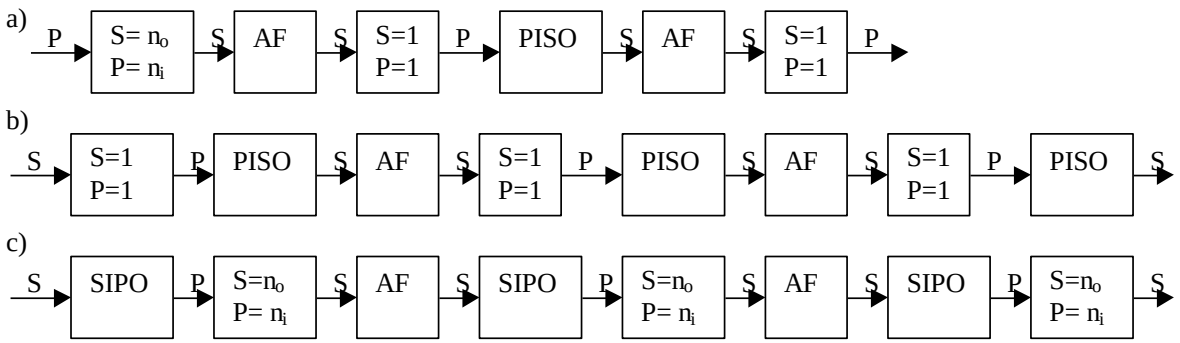

Fig. 8. Block diagram for 3 PSAN layers NN for parallel (P) or serial (S) input output interface

In the case when the calculation time *t* is shortened i.e. $k \cdot t=n$ (k – an integer), the above architecture can be straightforward modified. Instead of the PSAN $S=n_o$, $P=n_i$, the PSAN $S=n_0/k$, $P=n_i$ should be used. Similarly, the PSAN $S=1$, $P=1$ should be replaced with the PSAN $S=1$, $P=k$. Besides instead of a single AF per NN layer, k AFs should be used. The case is more complicated when n/k does not give an integer. In this case either idle cycles can be inserted or the input output interface is disregarded when selecting the optimal parameters *S* and *P*.

In the case when the required calculation time *t* is increased: $t=k \cdot n$, Fig. 8 can also be straightforward modified. The following blocks should be exchanged: ($S=1$, $P=1$) with ($S=k$, $P=1$), and ($S=n_o$, $P=n_i$) with ($S=n_o$, $P=n_i/k$). Unfortunately both ($S=k$, $P=1$) and ($S=n_o$,

$P=n_i/k)$ require data sequence converter which causes that the input-output interface can be disregarded when selecting the optimal parameters *S, P*. It should be noted that the increase of the required calculation time *t* does not influence the number of AFs – this architecture still requires one AF per NN layer.

In more general case when: $n_0 \neq n_1 \neq n_2 \neq n_3$ the input output interface should be considered individually for each layer and in most cases can be disregarded when selecting the optimal PSAN architecture.

4 Implementation results

In order to present the implementation results of the PSAN, the example of the 2-6-1 NN with parallel input is considered. The data width (the number of bits) for different arithmetic path is as follows:

- input data width: 8
- weight width (ROM width): 8
- multiplier (input × weight = product): 8×8= 16
- accumulator: 17 (the first layer), 19 (the second layer)
- activation function input: 12 (disregard the LSBs of the accumulator)
- activation function output: 8

The AF is implemented as a simple saturation function given in Fig. 9.

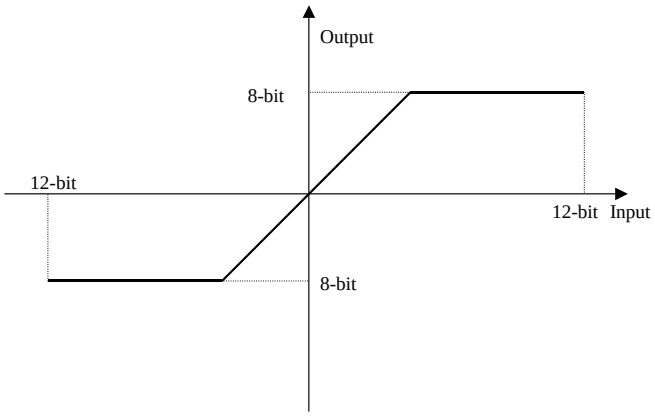

Fig. 9. Activation Function (AF) used for the presented implementation results

Tab. 4 presents implementation results for different required calculation time *t* and the resultant optimal PSAN architecture. Column 2 and 3 presents the number of 4-input LUTs and the number of Flip-Flop (FF) used for the input interface. The input interface is used only for *t=4*, *S=2* and *P=1* when the parallel data format is converted to serial (PISO) and the same input data are fed twice (FIMFO). The data format conversion between the first and the second PSAN layer is implemented within the first layer. Column 4,5 and 11,12 present the optimal *S* and *P* parameters for the first and the second layer respectable. Column 6 and 17 presents the number of implemented neurons for the first PSAN layer and the total number of implemented neurons – for the second layer only a single neuron is always implemented. Column 7, 13 and 18 presents the number of multipliers N_m for the first, second layer and the total. It should be noted that the multipliers are implemented employing CLB logic – the built-in multiplier blocks, e.g. in Virtex II, are not used. Column 8, 14 and 19 presents redundancy *R* for the first, second layer and the total, respectably. Column 9,10 and 15, 16 and 20, 21 presents the number of 4-input LUTs and the number of flip-flops (FFs) for the first, second layer and the total, respectably.

The pipeline architecture is parameterized. The presented implementation results are presented for an middle pipeline mode (*p=2* – pipeline FF are inserted after 2 layers of LUT logic), consequently the architecture can be further speed-up by increasing the number of FFs (*p=1)* or the number of the FF can be reduced by the cost of lower clock frequency (*p>2*). The implemented circuit can be clocked with the frequency 100 MHz (for *p=2* and Xilinx Spartan 2E: XC2S300E -pq208 -6), the clock frequency is roughly the same for every *t.*

	Input		Layer 1							Layer 2						Total				
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	LUT	FF	S	P	N_n	N_m	R	LUT	FF	S	P	N_m	R	LUT	FF	N_n	N_m	R	LUT	FF
1	0	0	1	2	5	10	0	895	447	1	6	6	0	553	272	6	16	0	1448	719
2	0	0	2	2	3	6	1	652	341	1	3	3	0	299	154	4	9	1	951	495
3	0	0	3	2	2	4	0.67	445	235	1	2	2	0	209	154	3	6	0.67	654	389
4	0	0	3	2	2	4	1.5	445	235	1	2	2	0.5	209	107	3	6	2	654	342
	31	15	2	1	3	3	0.5	387	213	1	2	2	0.5	209	107	4	5	1	627	335
5	0	0	5	2	1	2	0	235	114	1	2	2	0.8	209	107	2	4	0.8	444	221
6	0	0	5	2	1	2	0.33	240	130	1	1	1	0	117	59	2	3	0.33	357	189
7	0	0	5	2	1	2	0.57	240	130	1	1	1	0.14	117	59	2	3	0.71	357	189
8	0	0	5	2	1	2	0.75	240	130	1	1	1	0.25	117	59	2	3	1	357	189
9	0	0	5	2	1	2	0.89	240	130	1	1	1	0.33	117	59	2	3	1.22	357	189
10	0	0	5	1	1	1	0	148	82	1	1	1	0.4	117	59	2	2	0.4	265	141

Tab. 4. Hardware implementation of the 2-6-1 NN for different required calculation time t.

Two different solutions are considered for the first layer and for *t=4*: (*S, P*) = (3,2) and (2,1). For (*S, P*) = (3,2) a larger redundancy *R* is obtained but no additional input data format converter is required. According to the implementation result it is beneficial to implement the latter circuit (*S, P*) = (2,1) for which the input interface is required but the redundancy is lower.

It can be seen from Tab. 4 for *t=1* and *t=2* that doubling *t* does not cause that the occupied area is halved. This can be explained by the fact that for *t=1* different multiplication method is used: constant multiplication (*t=1*) instead of the variable multiplication (*t>1*). It should be noted that the constant multiplication is not optimized (only VHDL synthesis optimization is used) and therefore further work is still required for this case. Besides for the first layer and *t=2* the redundancy *R>0.* In order to better illustrate the circuit efficiency for different *t*, Tab. 5 was introduced. The value $A \cdot t$ presents the product of area (number of 4-input LUTs) and required calculation time *t*. In addition the product $A \cdot t$ was scaled by the redundancy R_1 i.e. $(A \cdot t/(1+R_1))$ in order to make the circuit efficiency independent from the redundancy R_1. It can be seen from Tab. 5 that for the greater *t* the circuit efficiency is lower. This is caused by the additional control circuit which further influence the total circuit area for lager *t* (for grater *t* the active arithmetic area is lower). Besides additional circuit such as weights ROMs, Activation Functions (AFs) are not scaled proportionally with required calculation time *t.*

t	Layer 1			Layer 2			Total
	A·t	R_1	(A·t)/(1+R_1)	A·t	R_1	(A·t)/(1+R_1)	A·t
1	895	0	895	553	0.00	553	1448
2	1304	0.2	1087	598	0.00	598	1902
3	1335	0.2	1113	627	0.00	627	1962
4	1780	0.6	1113	836	0.33	627	2616
4	1548	0.2	1290	836	0.33	627	2508
5	1175	0	1175	1045	0.67	627	2220
6	1440	0.2	1200	702	0.00	702	2142
7	1680	0.4	1200	819	0.17	702	2499
8	1920	0.6	1200	936	0.33	702	2856
9	2160	0.8	1200	1053	0.50	702	3213
10	1480	0	1480	1170	0.67	702	2650

Tab. 5. A·t product for different t

For the developed VHDL code of the PSAN, the number of AFs in a single layer is the lower value of: the number of implemented neurons N_n or the number of parallel outputs. For the presented example for the first PSAN layer the number of AFs is equal N_n. In general the minimum number of AFs N_{AF_min} is as follow:

$$N_{AF\_\min} = \left\lceil \frac{n_o}{t} \right\rceil \qquad (13)$$

To obtain the number of AFs N_{AF} equal the minimum number of AF N_{AF_min}, an addition multiplexer / registers are required in some cases. This additional circuit was not implemented in the presented implementation results therefore one additional AF was required for *t=4*, S= 2 and *P=1* (see Tab. 4, for the first layer $n_o=5$).

When designing VHDL code of the PSAN layer, the most difficult part of the design was interface between different NN layers. This interface is strongly parameterized and therefore it is rather difficult to develop the optimal interface for every parameters set. For example in Tab. 4 for *t=4*, *S=2* and *P=1*, the input interface was used which occupied 31 4-inputs LUTs. A similar result might be obtained when only a 8-bit multiplexer and a modulo 2 counter were employed (the total area would be roughly 10 LUTs).

The PSAN architecture uses *strobe (stb)*, *acknowledge (ack)* handshake between different layers of NN, i.e. a NN layer drives *stb* signal high when its output data are valid and in reply, the next NN layer drives *ack* signal high when it is ready to accept the new input data. This approach causes that the *stb-ack* chain might grown very long and the propagation time would be defined by the control path. As a solution to this problem, FIFO buffers are inserted inside every PSAN layers (inside the interface logic). This causes that the *stb-ack* chain propagated only within a single PSAN layer. The FIFO buffer was also implemented inside the input interface converter in Tab. 4 for *t=4*, *S=2* and *P=1*, and this explains additional area requirements for this circuit. For serial parameter *S>1*, the FIFO buffer is replays with the FIMFO (First-In Multiple-First-Out) buffer (as it is the case for the *t=4*, *S=2* and *P=1*).

5 Conclusions

In this paper a novel Parallel-Serial Architecture for NN (PSAN) has been presented. The PSAN architecture is especially suitable for already trained feed-forward NN and for different number of neurons in each layer. One of the most important feature of the SPAN is that it is

strongly parameterized by the parallel *P* and serial *S* parameters and therefore different circuit can be selected for different input parameters. The required calculation time *t* (the number of clock cycles a single output vector is calculated) is the most important parameter when selecting optimal architecture. Usually fully parallel, serial or single multiply and accumulate (MAC) approaches were implemented. The SPAN broadens significantly the number of possible solutions by introducing additional circuits which are more or less parallel or serial. As a result the area occupied by the PSAN may be significantly reduced.

[1] Lasner J., Lehmann T. *An analog CMOS chip set neural networks with arbitrary topologies*, IEEE Trans. Neural Networks, 1993, Vol. 4. pp. 441-444

[2] Wilamowski B.M., Jaeger R.C., Kaynak M.O., *Neuro-Fuzzy Architecture for CMOS Implementation*, IEEE Trans. on Industrial Electronics, Vol. 46, No. 6, Dec. 1999, pp. 1132-1136

[3] Savran A., Unsal S., *Hardware Implementation of a Feedforward Neural Network using FPGAs*, International Conference on Electrical and Electronics Engineering, 3-7 Dec. 2003, Bursa, Turkey, http://eleco.emo.org.tr/eleco2003/ELECO2003/bsession/B1-17.pdf

[4] Beuchat J.L. Haenni J.O., Sanchez E., *Hardware Reconfigurable Neural Networks*, International Parallel and Distributed Processing Symposium, Orlando, Florida, Mar 30 - Apr. 3, 1998, http://ipdps.eece.unm.edu/1998/raw/haenni.pdf

[5] Ayala J. L., Lomena A. G., Lopez-Vallejo M., Fernandez A. *Design of a pipelined hardware architecture for real-time neural network computations,* EEE Midwest Symposium on Circuits and Systems, Tulsa (Oklahoma, USA), August 2002

[6] Steven A. Guccione and Mario J. Gonzalez, *A Neural Network Implementation Using Reconfigurable Architectures, "More FPGAs"*, Will Moore and Wayne Luk, Abingdon EE&CS Books, Abingdon, England, 1993, 443-451.

[7] Nordström, T. and B. Svensson, *Using and designing massively parallel computers for artificial neural networks, Journal of Parallel and Distributed Computing,* vol. 14, no. 3, pp. 260-285, 1992

[8] Jihan Zhu and Peter Sutton, *FPGA Implementations of Neural Networks – a Survey of a Decade of Progress*, in Proceedings of 13th International Conference on Field Programmable Logic and Applications (FPL 2003), Lisbon, Sep 2003.