Comparison of FPNNs Models Approximation Capabilities and FPGA Resources Utilization

Martin Krcma, Zdenek Kotasek, Jakub Lojda
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: ikrcma@fit.vutbr.cz, kotasek@fit.vutbr.cz, ilojda@fit.vutbr.cz

***Abstract*—This paper presents the concepts of FPNA and FPNN, used for the approximation of artificial neural networks in FPGAs and introduces derived types of these concepts used by the authors. The process of transformation of a trained artificial neural network to an FPNN is described. The diagram of the FPGA implementation is presented. The results of experiments determining the approximation capabilities of FPNNs are presented and the FPGA resources utilization are compared.**

I. Introduction

The artificial neural networks [9] are one of the important models of sofcomputing and artificial intelligence. Their structure is inspired by the structure of the human brain and they dispose of a high capability of learning and memorizing to solve various types of tasks. Basically, the goal of the artificial neural network is to learn the relation between two sets of data vectors, to generalize the relation, to determine its features and to use it for the determining the relation of the unknown vectors belonging to the same problem. This capability can be used for classification tasks, for timeseries and functional prediction, to control tasks, to image recognition, clustering and other tasks.

Neural networks are composed of a set of *neurons* computing the *activation function* over the *weighted sum* of their inputs. The neurons are interconnected with the weighted connections called *synapses*. The learning of the neural network is basically a process of setting the weights.

The networks have been implemented in various kinds of devices starting from analog computers to the most modern processors, VLSIs, graphical processing units and FPGAs. This paper deals with one of the possible implementations of artificial neural networks in FPGAs - FPNA/FPNN.

The concept of Field Programmable Neural Arrays/Networks (FPNAs/FPNNs) [1], [2] in design is meant to simplify the implementation of artificial neural networks in FPGAs by adjusting its properties to be more suitable for the implementation into their logic. The simplification originates from its main feature - a highly customizable structure which makes it possible to establish resource sharing between the original synaptic connections of the neural network. This is done by using its customizability to simplify the interconnection model. The concept were used for implementing large scale spiking networks [11], [12].

The FPNNs are not the same structures as neural networks, although they can be constructed in that way. The FPNNs represent a different model which can structurally differ from the implemented neural network. They can also have different capabilities, which means that they are not only an implementation of the neural networks, they are an approximation of neural networks as well. Since the FPNNs can be constructed in various ways and types, the approximation accuracy can be different.

The goal of this paper is to describe the types of FPNNs and compare the approximation capabilities of these types. The FPGA resources utilization of the FPNNs is compared as well.

The FPNNs were formally defined in [1], [2]. In order to follow the original definitions, the presented work is based on these definitions and on definitions derived from them. For our purposes we modified the original definitions in order to suit them to our way of using the concept. This step allowed us to use different level of the approximation accuracy. Our further definitions specify special derived types of FPNNs. Also, they allow us to describe easily the algorithms of mapping trained neural networks to FPNNs.

In our paper published at NORCAS 2015 conference [7], we described the concept of Field Programmable Neural Networks for artificial neural networks implementation in FPGA. We also presented a model of fault tolerant FPNNs and various fault tolerance improving techniques based on the model. Experimental results were also provided. Now, in this paper we describe how we continue in our research - the formal definitions of the FPNA/FPNN concept are presented. The problem of process of direct transformation of the trained neural network to FPNN together with the related algorithms are described. The goal of experiments presented in the paper is to determine the approximation capabilities of different FPNNs of the reduced and the full type, the results are described in the paper. In the earlier paper [6] we dealt with the mapping of the neural networks to FPNNs of the most simple type with a number of methods. This paper follows this work by extending it to other types of FPNN with more detailed description of the models, methods and algorithms.

The paper is organised as follows - the first section introduces the FPNA/FPNN concept. The second section deals with the problem of neural networks transformation to FPNNs and describes our transformation algorithms while the third section presents the diagrams of FPGA implementation of FPNNs. In the fourth section experiments and results are described. The last section summarizes the paper.

II. FPNN

For purposes of our research we developed a new definition of an FPNN (see Definition II.1, original definition by B. Girau [1], [2]). According to this definition, an FPNN is a structure composed of two types of units (together called *neural resources*). The units of the first type are called *activators* (the set N) and represent original neural network neurons. They perform the same actions as neurons - they iteratively gather input data into *potential*, then apply an activation function to obtain the output. The activation function is represented by the *function operator* "f" and the *iteration operator* "i" is responsible for input data processing to provide the input to the function operator.

The interconnections between activators are realized by the other unit type called *links* (the set L). The links perform approximation of the original synaptic weights (they compute the weight multiplication) according to the rest of the FPNN parameters. The actual data interconnection model is presented by an oriented graph (N, E), where E is a set of valued edges interconnecting the activators. Every edge is usually split up to a sequence of links which allows us to construct various structures. The more we split the edges into links, the more flexibility we obtain.

Definition II.1 (FPNN [1])**.** We say that structure (N, L, E, ϕ, ω) is an *FPNN* if the following statements hold true:

1) N is a set of units called *activators* that dispose of:
   a) An iterative variable t_n: $\forall n \in N : \exists t_n \in \mathbb{R}$
   b) A default value of t_n: $\forall n \in N : \exists o_n \in \mathbb{R}; t_{n_0} = o_n$
   c) A number of iterations: $\forall n \in N : \exists a_n \in \mathbb{N}$
   d) An iterative operator (x_n is an input data): $\forall n \in N : \exists i_n : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$; $t_{n_a} = i_n(t_{n_{a-1}}, x_n); a = 1..a_n$
   e) A function operator: $\forall n \in N : \exists f_n : \mathbb{R} \to \mathbb{R}$
2) L is a set of units called *links* that dispose of:
   a) A set of *link operators* $\forall l \in L : \exists A_l$: $A_l = \{\alpha_n(x) | \alpha_n(x) = W_n \times x; W_n \in \mathbb{R}; n = 1..c\}$
3) E is a set of valued oriented edges: $(m, n) \in E; m, n \in N)$. The edge value is defined: $\forall e \in E : \exists W_e \in \mathbb{R}$
4) (N, E) is an oriented graph denoting the interconnection between activators.
5) ϕ is a function $E \to L^+$, so that: $\forall e \in E : \phi(e) = (l_1..l_n); l_1..l_n \in L; n > 0$
6) ω is a function $E \to L^+$, so that: $\forall e \in E : \phi(e) = (l_1..l_n); l_1..l_n \in L; \omega(e) \subseteq \phi(e); 0 < n \leq |\phi(e)|$
7) Edge-to-operator functions $\sigma_l : E \to A_l; l \in L$: $\forall e \in E \wedge \forall l \in \phi(e) : \sigma_l(e) = \alpha_l^x; \alpha_l^x \in A_l$
8) Operator determining $\psi_l : E^+ \to A_l, l \in L$: $\forall l \in L : \psi_l(e_1..e_n) = \alpha_x \Leftrightarrow \alpha_x \in A_l \wedge l \in \omega(e_1) \wedge .. \wedge l \in \omega(e_n) \wedge \sigma_l(e_1) = .. = \sigma_l(e_n) = \alpha_x$
9) A set of input nodes exists: $\exists N_i = \{n \in N \mid deg^+(n) = 0\}$ $\forall n \in N_i : i_n = \emptyset; f_n(x) = x$

The actual FPNN structure is determined by the (N, E) graph and the ϕ function. The edges are split up to the sequences of interconnected links, given by the ϕ functions which realize the interconnection between activators and the approximation of the edges weights. The edges weight approximation is determined by the ω, σ, ψ functions and realized by the *link operators*. Link operators are functions which are applied to the data passing through the links. Every link disposes of one or more link operators (the A_l sets). To determine a link operator which should be assigned to the particular edge (to the data which would originally pass through the edge) the σ functions are constructed. All the link operators in the sequence (realizing an edge) are applied to all the passing data (according to the σ functions). To establish the weight approximation, it has to be decided which links in the sequences will be used for the approximation by the construction of the ω function. The actual approximation is determined by the ψ functions which construct link operators for the assigned edges (by the ω, σ functions). This will be further explained in the section III.

To preserve the consistency with the original definition [1] we add the following: If only the graph (N, E), iteration and function operators are defined, the structure is called FPNA (Field Programmable Neural Array) [1] and it defines the whole class of possible FPNNs. Every FPNN can bee seen as an instance of some FPNA.

A. Grid FPNN

For our research purposes we developed a special type of FPNN based on the above provided definitions. *Grid FPNN* (definition II.2) is an FPNN with an enforced limitation of the structure causing it to form a grid shape. The reason for this is to make an FPNN suitable for the implementation in FPGAs due to the similarity of the grid FPNNs structure and FPGAs interconnection bus and the sharing of resources in links.

Definition II.2 (Grid FPNN)**.** We say that FPNN is the *grid FPNN* if the the following statements hold true:

1) The activators are organized into layers.
2) The two sequences of interconnected links exist in all layers composed of more than one activator. The number of links in every sequence is one less than the number of activators in the layer. The output of every link is connected to the input of the nearest activator. The sequences go in the opposite ways.
3) The output od every activator is connected only to a single link which provides the connection to the next layer. The output of the link is connected to the nearest activator and to the nearest links of one or both link sequences in the layer (which realizes connection to all other activators).

An example of a grid FPNN can be seen in Fig. 1. In the figure, the circles represent activators, wide arrows represent links and the thin arrows represent data interconnections. The orientation of the connection arrows shows the way of the passing data. The straight wide dashed/dotted arrows represent the original neural networks synapses. The thin dashed/dotted arrows represent the sequences of links approximating the particular synapses. The synapses and the particular sequences are drawn with the same line and arrow styles. As the picture

illustrates, there is only one link (called *initial*, Definition II.4) on the output of every activator which provides the connection to the following layer. It is directly connected to one successive activator in the following layer. The connection to the other activators goes through the sequence of links within the whole layer. Two sequences of the links are going the opposite ways. They are called *Interconnection sequence* (definitions II.3-II.5). Every layer with more than one activator has an interconnection sequence within.

Definition II.3. *A sequence of links* is generally a sequence of *directly interconnected* links.

Definition II.4. *An initial* is a link having no link predecessors. It has only an activator predecessor.

Definition II.5. *An interconnection sequence* is a sequence of links interconnecting activators within a layer. It is composed of two sequences going the opposite ways. The input of every link is connected to one or two preceding links, the output is always connected to the nearest activator and to the succeeding link in the sequence (if it exists).

B. Different levels of approximation

The approximation capabilities of the FPNN depend on the number of available link operators present in links and the number of edges assigned by the ω function for approximation to the links. Respectively, the ratio between the numbers of operators and assigned approximated edges is the essential parameter for the FPNN approximation abilities. The numbers can be equal. In that case, the ψ functions only determine the value approximating the given edge (the member of the multiplication sequence as described in the next subsection). Thus, every edge has its link operator counter part. In this case, the approximation of the original neural network weights (suppose that original synapses were directly transferred to edges, thus (N, E) graph is isomorphic to the original network) is accurate. We call the FPNN with these properties as *Full FPNN*.

The definition allows us to reduce the number of link operators. In that case, the ψ function is surjective and its purpose is to find a *compromise* between a number of edges mapped to one link operator. In this case, the approximation accuracy suffers from the decrease caused by *sharing* the link operators between multiple edges. However, this kind of sharing reduces the FPGA resources utilization (the main goal of the FPNN concept) since, the memories containing the link operators are smaller as well as related logic (multiplexors, possibly multipliers etc.). And the accuracy decrease does not have to be necessary critical since the neural networks are potentially robust against some weights losses. The usage of this technique of resource utilization reduction is a matter of preference and depends on concrete situation and a way of usage.

We distinguish two other types of FPNN. The *reduced* FPNN and the *light* FPNN. The meaning of light FPNN is simple - every link disposes of only one link operator ($\forall l \in L : |A_l| = 1$). In this case, the link multiplier turns into a constant multiplier which offers the highest spare of resources. However, the accuracy suffers the most.

The last type mentioned in this paper is the reduced FPNN which disposes of the same number of link operators as it has the number of direct link predecessors in the link sequences it belongs to. This type approximation capabilities are determined by an FPNN structure as the number of link operators is directly dependent on the number of existing link sequences and their interconnection. The explanation based on Fig. 1 would be the most appropriate. Consider the third (the rightmost) link of the lower part of the interconnection sequence (the sequence heading to the right). This link approximates three edges originating in the three leftmost activators. However, it has only two direct link connected predecessors. So, in the case of reduced FPNN, it would have two link operators. The first one approximates the edge originating in the third leftmost activator. Since there is no sharing of this link operator, the approximation of the edge is accurate. However, the second link operator is shared between two edges originating in the first two leftmost activators. An approximation of these edges would be hence less accurate due to sharing the link operators. But in case of the full FPNN, the approximation would be accurate since there would be three affine operators, one for every edge.

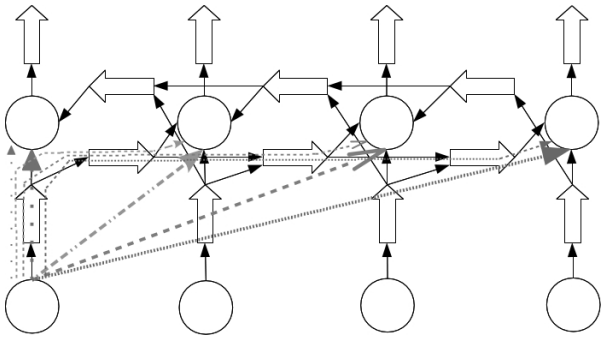

Fig. 1. Synapses (edges) approximation in a grid FPNN

Definition II.6 (Light FPNN)**.** We say that FPNN is a *light FPNN* if the folowing statemet holds true: $\forall l \in L : |A_l| = 1$.

Definition II.7 (Full FPNN)**.** We say that FPNN is a *full FPNN* if the following statement holds true:
$\forall l \in L \wedge \forall e \in E : |A_l| = |\{e | e \in \omega^{-1}(l)\}|$.

Definition II.8 (Reduced FPNN)**.** We say that FPNN is a *reduced FPNN* if the following statements hold true:

1) The edge equivalence is defined:
$\forall e_1, e_2 \in E; l \in L : e_1 \equiv_l e_2 \Leftrightarrow \phi(e_1) = l_1^1..l_x l..l_n \wedge$
$\phi(e_2) = l_1^2..l_y l..l_m; l_x = l_y$
2) $\forall l \in L$ the size of A_l is equal to the number of the equivalence classes generated by the $\equiv_l$.

III. Mapping of neural networks to FPNNs

Mapping is a process of direct transfer of an artificial neural network into an FPNN without using a training data set and without the need of learning (other works [3] deal with training of FPNNs). Mapping uses information obtainable from an original neural network such as weights, biases, activation functions and the network structure.

The first phase of mapping would be the construction of an appropriate FPNN. The first step is construction of (N, E) graph which should be (but does not have to be) isomorphic

to the original neural network. This step contain the mapping of neurons to activators - the basis functions are mapped to the iteration operators, the activation functions to the function operators and the biases to the o_n parameters. The second step is the links creation according to the intended shape of the FPNN. The next step is to assign the edges to the sequence of links (constructing the ϕ function) which specify the concrete shape of the FPNN. According to the approximation accuracy preferences, the A_l sets of link operators needed to be constructed now.

Since all data between activators will flow through the sequences of links (given by the ϕ function) interconnecting them, the data will be modified using link operators of all links in the sequences. Therefore, the σ functions have to be constructed to determine the relations between edges and the link operators.

However, not all link operators have to be used for the edges weights approximations. Some of them (or all but one) can be used as data passers which can be possibly shared between edges. For better understanding, consider an edge e, its weight $W_e = 3$ and $\phi(e) = l_1 l_2 l_3 l_4$. Using σ functions we obtain a sequence of link operators $\alpha_1\alpha_2\alpha_3\alpha_4$ assigned to the edge. If it would be our intention to use all these operators to approximate the W_e, the approximation sequence would then be most likely composed of three operators with value of 1 and one operator with value of 3. Therefore, we would need to have an extra 1-valued operator in all the three links. Which would be easy to use, but costs more resources. I could be more wise to use other existing operators (approximating other edges) in some of links and special operators used for e approximation in others. In our case, if σ functions map the e to the operators in $l_1..l_3$ links with values of $1.5, 2, 2$, the link operator in l_4 used for approximation of the e would have the value of 0.25 (since $1.5 \times 2 \times 2 \times 0.25 = 3$). In this case, only one operator was used for the e approximation and the others were shared with other edges without influencing them (they were used for other edges approximation in the same way). To specify which link operators are used for approximation of concrete edges, the ω function is constructed. If sharing is not in our intention (for example for matter of data type accuracy which could suffer from multiple multiplications), the $\omega(e) = \phi(e)$ for all edges.

The last step is to determine the ψ functions. These functions serve for finding a compromise if more then one edges are mapped to a single link operator for approximation. If all the edges dispose of exclusive operators of their own (the full FPNN), no ψ functions are needed. In other cases (the reduced and light FPNNs) they have to be constructed. There are plenty of way of finding the compromises - arithmetic average, median, weighted average and others. We have described the results of using different compromises (ψ functions) in our paper at DDECS 2015 [6].

There are several possible ways how to determine the ω and the σ, ψ functions. Using evolution algorithms and optimization algorithms could be one of them. In this paper however we would like to describe a systematic approach of mapping trained layered feedforward neural network (perceptron like) to grid FPNNs. We suppose the (N, E) graph to be isomorphic to the original neural network and the L set and the ϕ function to be constructed to form a grid FPNN according to the definition II.2. The ω functions is constructed according to the equation 1. According to it, every edge is approximated by the link operator of the last link in the sequence given by the ϕ function. The σ functions need to be constructed to create groups of edges according to the equivalence classes generated by the $\equiv$ function from the definition II.8. The ψ functions were chosen as the arithmetic average (equation 2). The $opSeq_e$ is the sequence of link operators assigned to the edge e except the last one. The P_e is the value of the product of the link operators in the $opSeq$ set. It denotes the actual multiplication value in the last link before $\omega(e)$. According to the value and the value of the edge weight, the value A_e needed to accurate approximation is computed. In full FPNN, this value would be directly assigned to the link operator. In the presented equation, the arithmetic average is applied to all A_e values mapped to the link operator.

$$\forall e \in E : \omega(e) = l_n \Leftrightarrow \phi(e) = l_1..l_n \tag{1}$$

$$\begin{aligned} opSeq_e &= (\alpha_l^e | \alpha_l^e = \sigma_l(e) \wedge l \in \phi(e) \setminus \omega(e)) \\ P_e &= \prod_{\alpha_l \in opSeq_e} \alpha_l \\ A_e &= \frac{W_e}{P_e} \\ \forall l \in L : \psi_l(e_1..e_n) &= \frac{\sum_{e_x \in \{e_1..e_n\}} A_e}{n} \end{aligned} \tag{2}$$

$$\forall e \in E : A_e = \prod_{l \in \phi(e); \alpha_l^e = \sigma_l(e)} \alpha_l^e \tag{3}$$

A. Mapping algorithms

On the base of the presented principles we implemented the following algorithms which perform a mapping of trained neural network to the grid FPNN. The construction algorithm of the FPNN will not be described in explicit details in this paper, however the main idea was presented in the preceding section. The algorithms use the definitions and equations presented above as well as the declaration in Table I.

At first, the auxiliary variables have to be initialized in the **Algorithm 1**. Then, the ordered set of link sequences must be constructed using the **Algorithm 2**. The set is called $chains$ and it is constructed for each layer separately using the ϕ function and ordering the resulting link sequences by their length (ascending order). The reason why to order the sequences is that it is suitable to start mapping with the shortest sequences of the length 1 (initials - always present in the grid FPNN) and continue with longer sequences in the next steps, determining one additional operator (member of the multiplication sequence) in the sequence in the each subsequent step. This means that the links (their operators) are mapped one by one creating longer mapped sequences in each step.

In every step a new link is selected and its operators determined. In order to compute the values of the operators, it is needed to construct the groups of edges related to each link operator. Determining of these groups differs in case of

each type of FPNN. In case of reduced FPNN, the edges are separated according to the link predecessor they pass trough, as the definition specifies. The groups are constructed as the equvalence classes of the $\equiv_l$ function from the definition II.8. This is done by the **Algorithm 3**. In case of light FPNN, the equation on the second line of the **Algorithm 3** shall be replaced by the equation 4 assigning all the edges to the one group which will be mapped to the single link operator. If a full FPNN is being mapped, the line should be replaced by the equation 5 assigning every edge to the separated group.

$$groups \leftarrow \omega^{-1}(l) \tag{4}$$

$$groups \leftarrow \{\{e\} | e \in \omega^{-1}(l)\} \tag{5}$$

After the groups edges are constructed, the values of partial products P_e from the equation 2 as well as the approximation values A_e are computed for every edge in every group in the **Algorithm 4**. As the last step, the related link operator is computed for each group using the ψ function. The operators computation is complete and the link is removed from the chain a algorithm continues with the successive link.

TABLE I. DECLARATIONS

Declaration	Description
$FPNN = (N, L, E, \phi, \omega)$	a grid FPNN
$layers \in \{N^*\}^*$	The set of FPNN activators layers.
$chains \subset E^*$	An ordered collection of link sequences.
$sortByLength : E^n \rightarrow E$	Sorting by path length.
$firstNodeOf : E^n \rightarrow E$	Chain's first node.
$\forall e \in E : \exists P_e \in \mathbb{R}$	Partial product of the operators sequence.
$\forall e \in E : \exists A_e \in \mathbb{R}$	Approximation value for the operator computation.

```
1: procedure INITIALIZE(NN, FPNN)
   /* Init of the variables:          */
2:     for all ∀e ∈ E do
3:         P_e ← 1.0
4:         A_e ← 1.0
5:     end for
6: end procedure
```

Algorithm 1. Initialization algorithm

The presented algorithms represent the very basic mapping method. In our previous research [6] we developed a set of additional methods for mapping the light FPNNs which can be used to map the reduced FPNNs as well. The algorithms differ in the way of computing the ψ function. They are based on different usage of weighted algorithms with different weights

```
1: function DETERMINECHAINS(lr = {n1..n_n} ∈ N^n)
2:     chains ← ∅
3:     for all n ∈ lr ∧ s ∈ N do
4:         for all (n, s) ∈ E do
5:             chains ← chains ∪ φ((n, s))
6:         end for
7:     end for
8:     sortByLength(chains)
9:     return chains
10: end function
```

Algorithm 2. Chain determination algorithm

```
1: procedure DETERMINEGROUS(l ∈ L)
2:     groups ← [e]_{≡_l} ∀e ∈ ω^{-1}(l)
3:     return groups
4: end procedure
```

Algorithm 3. Initialization algorithm

```
1: procedure MAPFPNN(NN, FPNN)
2:     INITIALIZE(NN, FPNN)
3:     for all layer ∈ layers do
4:         chains ← DETERMINECHAINS(layer)
           /* Mapping path by path:      */
5:         for all r ∈ chains do
6:             l ← firstLinkOf(r)
               /* Multiplicands comput.:  */
7:             for all g ∈ DETERMINEGROUPS(l) do
8:                 for all e ∈ g do
9:                     P_e = ∏_{α_l ∈ opSeq_e} α_l
10:                    A_e ← W_e / P_e
11:                end for
                   /* Computing the link
                   operators:             */
12:                α_{σ(e∈g)} = ψ(g) = (Σ_{e∈g} A_e) / |g|
13:            end for
               /* Shortening the chain:  */
14:            r ← r \ {l}
15:            chains ← {p | p ∈ chains ∧ p ≠ ∅}
16:        end for
17:    end for
18: end procedure
```

Algorithm 4. Reduced FPNN mapping algorithm

determination as well as on more advanced principles. They also use different ways of results optimization. However, in order to explain the problem we used the basic method only since the other methods are more complicated and their results could depend more on the concrete network.

We implemented these algorithms into our framework [5] dealing with the FPNNs. The framework allow us to construct FPNNs and map the neural networks to FPNNs as well as simulating the computation of the FPNN and generating the VHDL design for every FPNN. Using the framework, the mapping is very fast, depending on the methods, FPNN size and used optimizations it takes seconds to minutes to be executed.

IV. IMPLEMENTATION OF FPNNS INTO FPGAS

The VHDL implementation of both types was created according to the original design and schematic [1]. Another implementation was proposed in [10]. Both, activators and links were designed as separated units communicating with signals. The communication is based on the asynchronous *request - acknowledgement* model. Every neural resource generates requests for all units directly connected to its output (successors) when its computation is done. Once a successor starts to process the request, it sends the acknowledgement back to the original resource. When the original resource receives acknowledgements from all successors, it selects a new input request to process, sends the acknowledgement and begins the computation. The activators also send a *flag* together

with the requests. The flag is a constant number and it is used by links to select the proper weight to multiply width the input data. The links then propagate the flag to all connected links. Only the full FPNNs use flags. The reduced and light FPNNs implementation do not contain the logic related to flags processing and transition.

The implementations of both types of neural resources are similar, however they differ in used computational units. The diagram of standard link implementation is illustrated in Fig. 2 and the diagram of the activator in Fig. 3. Both types are composed of a multiplexor, demultiplexor, register, computation units and units for processing requests. The meaning of common units is described bellow:

- **SELECT** selects one of the active requests for processing using the *Round&Robin* algorithm. The requests from preceding neural resources are indicated by the set bits on its input. When the request is selected, it sets the $start$ signal up.
- **MUX** is an input data vectors multiplexer. It is controlled by the SELECT unit.
- **REG** is a register storing the selected data vector.
- **ACK_DEMUX** delivers an acknowledgement (generated by the $start$ signal) to the proper predecessor. It is controlled by the SELECT unit.

These units are present in both links and activators. They serve for input requests processing and delivery of the input data to the computation part of the unit. Computation part of links and activators is composed of different units:

- **MULT_ADD** applies the weights to the data. The key to select the proper weight is the flag associated with the request. The flag is selected from all of the flags at the input $FLAG\_IN$ by the value at the input s. The weights are stored in the memory inside this unit. Full FPNNs contain significantly more weights than reduced or light FPNNs.
- **ITER** iteratively computes the sum of all input data (simulates the neuron basis function). After a predefined number of iterations, it transmits the result to the TRANS unit and activates it using the fin signal. After every iteration it activates the $next$ signal which starts the processing of another request.
- **TRANS** computes the activation function (the output of the activator). The input is gained from the ITER unit. The activation function were sigmoid like function suitable for hardware implementation [13].

All computation units take the input data from the register REG, perform the computation of the result and transmit it to the neural resource output. They also activate the signal $ready$ which is an input of the output requests generators:

- **LINK_REQ_GEN** generates the requests to the connected successors when the $ready$ signal is set. It also receives the acknowledgements from the successors. Using the $free$ signal it controls the SELECT unit - it enables (when all acknowledgements are received) or disables (new request was selected - $start$ signal is up) its function.
- **ACT_REQ_GEN** is similar to the LINK_REQ_GEN, but it allows to activate the $free$ signal using the $next\_req$ signal without the requests generation.

These units are responsible for the control of the neural resource. When the processing of the selected request is started, they block the SELECT unit preventing it from selecting another request before the actual one is processed. After the computation is done, they generate output requests and hold the entire neural resource inactive until all requests are successfully received by the successors.

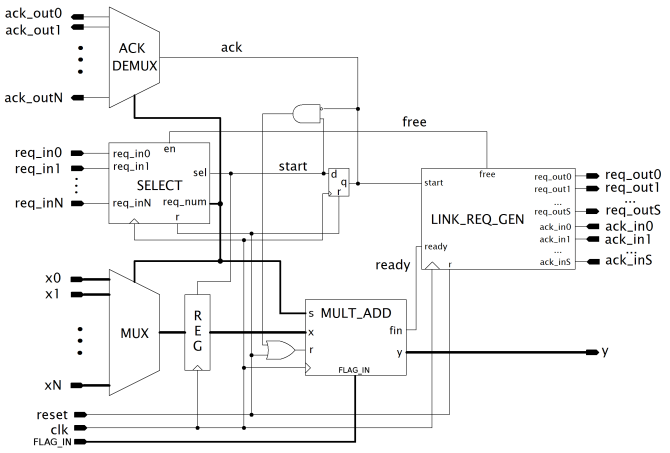

Fig. 2. Diagram of a link implementation - the interconnection of the inner units

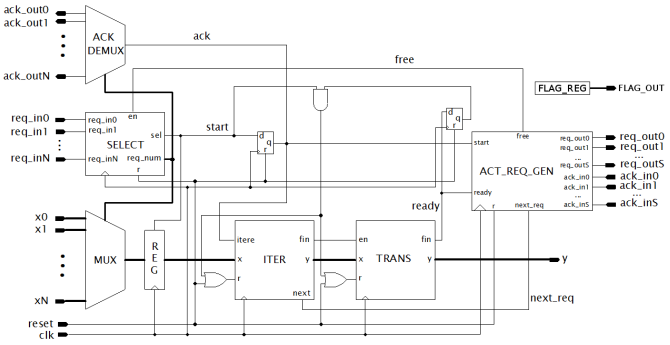

Fig. 3. Diagram of an activator implementation - the interconnection of the inner units

V. Experimental results

We experimented with the presented models and algorithms, the experiments and results will be now described and summarized. The experiments were focused on the approximation capabilities of the reduced and full FPNNs model, and on their FPGA resource consumption. The goals of the experiments were to show and compare the capabilities of both models and their space complexity. To perform the experiments we used our framework to simulate the FPNNs in order to get

the approximation accuracy. The VHDL design of every FPNN was generated using the framework as well.

The core of the experiments was a set of neural networks, and a set of structurally corresponding FPNNs of both types. Each trained neural network and the particular FPNN were both tested on a set of testing input vectors and their outputs were compared to each other to determine how the FPNN output differs from the reference neural network output. Since the FPNN serves as an approximation of the network, the match between their outputs is the essential information.

We worked with 15 neural networks of different structures trained for 3 classification tasks originating in the *Proben1* neural networks set of benchmarks [4]. The selected tasks were *Diabetes*, *Thyroid* and *Two Spiral*. The referential neural networks were constructed with respect to obtaining the set of networks with different structures not too big for the implementation in the selected FPGA, with no respect to their classification capabilities irrelevant for the FPNNs approximation quality determination.

Table II contains the information about reference neural networks. The *Name* column contains the network name (which is derived from the particular network task), the *Structure* column describes the network structure as numbers of neurons in each layer separated by the dashes. The *Neurons* and *Weights* columns summarize the numbers of neurons and weights of the network. The last column contains the number of links of the particular FPNN. The number of activators is equal to the numbers of neurons.

The experiments were run ten times and the best and the worst results of the approximation are presented in table III. Table III contains the approximation accuracy test results. The *Name* column refers to the particular FPNN (and the reference network), the *Reduced best* column contains the approximation accuracy of the reduced type FPNN output. The best case results are evident from this column. The *Reduced worst* column contains the approximation accuracy of the worst case results. It is the rate of the identically classified input vectors by both the network and the FPNN. The the rate of match of the Full FPNNs is 100%.

The created FPNNs were implemented using VHDL and were synthesized using the Xilinx ISE 14.4 tool. The target FPGA was the Xilinx Virtex-7 device *xc7v2000t-2flg1925*. All computations were implemented in fixed point form with 8 bits of the integer part and 8 bits of the fractional part [8]. The utilization of slice registers, slice LUTs, DSPs and minimum recommended clock period after synthesis were measured. The result are summarized in tables IV and V. The columns contain the utilization of the particular FPGA resources in the form of the total number and the percentual usage of the total available resources. The last column contains the minimum recommended clock period.

As the table III shows, the reduced FPNNs reached different levels of the approximation capabilities. Five of the reduced FPNNs were approximating the original network with the accuracy higher than 90 %. Three other FPNNs outreached the level of 70 % accuracy. However, some FPNNs did not cross the level of 50 % accuracy. The worst case results, which were in most cases very different from the best case results, were few times close to the 0% accuracy. These particular networks are unable to be approximated using the reduced FPNN and the full FPNN is the only choice. These findings show, how the mapping process is dependent on the concrete situation, the concrete neural network and the set of its weights. It shows that in some cases the mapping can be successful and the particular neural network can be approximated with the reduced FPNN, in other cases it is not possible. However, only the basic mapping method was used in these experiments. We developed a set of additional mapping methods which could provide better results. We presented and compared these methods and their optimizations in [6].

As the tables IV and V show, the results of the FPGA resources utilization experiments differ in case of both FPNN types. The slice registers consumption does not differ much, other results however differ significantly. As expected, the full FPNNs consume more resources than reduced FPNNs. Considering the number of consumed LUTs, the difference is in some cases only a few percent (diabetes1, twoSpiral1). In some other cases, the full FPNNs consume multiple number of resources than their reduced equivalents (diabetes4, diabetes5, thyroid2 and others). Also, full FPNNs consume multiple times more DSPs than the reduced FPNNs. This was expected since the multipliers are supposed to be more complex due to higher

TABLE II. THE LIST OF NEURAL NETWORKS AND THEIR PROPERTIES

Network name	Structure	Neurons	Weights	Links
diabetes1	8-16-8-2	34	272	78
diabetes2	8-64-2	74	640	200
diabetes3	8-64-32-2	106	2624	294
diabetes4	8-32-32-2	74	1344	198
diabetes5	8-32-32-32-2	106	2368	292
diabetes6	8-96-2	106	960	296
diabetes7	8-16-32-16-2	74	1184	196
diabetes8	8-16-32-64-2	122	2816	340
diabetes9	8-16-32-16-32-16-2	122	2208	336
thyroid1	21-21-3	45	504	86
thyroid2	21-42-3	66	1008	149
thyroid3	21-63-3	87	1512	212
thyroid4	21-84-3	108	2016	275
thyroid5	21-21-42-21-3	108	2268	271
thyroid6	21-63-21-3	108	2709	273
twoSpiral1	2-32-1	35	96	96
twoSpiral2	2-64-1	67	192	192
twoSpiral3	2-96-1	99	288	288
twoSpiral4	2-128-1	131	384	384
twoSpiral5	2-16-32-16-1	67	1072	188
twoSpiral6	2-16-32-48-1	99	2128	284

TABLE III. THE REDUCED FPNNS APPROXIMATION ACCURACY

FPNN name	Reduced best [%]	Reduced worst [%]
diabetes1	69.712	60.052
diabetes2	72.062	67.885
diabetes3	72.584	67.624
diabetes4	69.451	34.203
diabetes5	71.279	31.331
diabetes6	70.496	33.942
diabetes7	73.107	31.592
diabetes8	60.835	26.370
diabetes9	56.919	26.631
thyroid1	93.498	6.640
thyroid2	93.498	18.644
thyroid3	93.414	2.222
thyroid4	93.136	47.513
thyroid5	73.214	3.111
thyroid6	91.386	2.472
twoSpiral1	56.770	52.604
twoSpiral2	54.166	51.562
twoSpiral3	49.479	46.875
twoSpiral4	53.645	53.645
twoSpiral5	52.604	48.437
twoSpiral6	74.479	63.541

TABLE IV. THE RESULTS OF THE SYNTHESIS OF THE REDUCED FPNNS

Name	Regs (%)	LUTs (%)	DSPs (%)	MinPer [ns]
diabetes1	4726 (0%)	18235 (1%)	182 (8%)	12.725
diabetes2	11165 (0%)	45604 (3%)	464 (21%)	12.725
diabetes3	16252 (0%)	67049 (5%)	686 (31%)	12.725
diabetes4	11229 (0%)	45908 (3%)	462 (21%)	12.725
diabetes5	17270 (0%)	69225 (5%)	684 (31%)	12.719
diabetes6	16254 (0%)	67405 (5%)	688 (31%)	12.725
diabetes7	6028 (0%)	23996 (1%)	238 (11%)	12.725
diabetes8	18865 (0%)	77224 (6%)	796 (36%)	12.725
diabetes9	18699 (0%)	76614 (6%)	792 (36%)	12.725
thyroid1	5738 (0%)	20059 (1%)	182 (8%)	12.725
thyroid2	9164 (0%)	33763 (2%)	329 (15%)	13.629
thyroid3	12538 (0%)	48346 (3%)	476 (22%)	12.725
thyroid4	15898 (0%)	62788 (5%)	623 (28%)	12.725
thyroid5	15763 (0%)	62679 (5%)	619 (28%)	12.738
thyroid6	15906 (0%)	61993 (5%)	621 (28%)	12.733
twoSpiral1	5217 (0%)	21713 (1%)	228 (10%)	12.725
twoSpiral2	10209 (0%)	43543 (3%)	452 (20%)	12.733
twoSpiral3	15201 (0%)	64994 (5%)	676 (31%)	12.745
twoSpiral4	20193 (0%)	85449 (6%)	900 (41%)	12.874
twoSpiral5	10329 (0%)	42877 (3%)	448 (20%)	12.725
twoSpiral6	15413 (0%)	63528 (5%)	672 (31%)	12.725

TABLE V. THE RESULTS OF THE SYNTHESIS OF THE FULL FPNNS

Name	Regs (%)	LUTs (%)	DSPs (%)	MinPer [ns]
diabetes1	5108 (0%)	29564 (2%)	376 (17%)	11.519
diabetes2	11530 (0%)	69785 (5%)	904 (41%)	20.312
diabetes3	18078 (0%)	392493 (32%)	2160 (100%)	37.806
diabetes4	11536 (0%)	111880 (9%)	1608 (74%)	18.685
diabetes5	18630 (0%)	325759 (26%)	2159 (99%)	21.249
diabetes6	16618 (0%)	103702 (8%)	1352 (62%)	30.555
diabetes7	11460 (0%)	103727 (8%)	1448 (67%)	14.937
diabetes8	22166 (0%)	460280 (37%)	2159 (99%)	20.416
diabetes9	21418 (0%)	284021 (23%)	2159 (99%)	18.780
thyroid1	6780 (0%)	47190 (3%)	600 (27%)	11.519
thyroid2	13546 (0%)	132290 (10%)	1776 (82%)	13.402
thyroid3	13546 (0%)	132290 (10%)	1776 (82%)	13.402
thyroid4	17594 (0%)	215282 (17%)	2159 (99%)	34.575
thyroid5	18027 (0%)	286474 (23%)	2160 (100%)	16.469
thyroid6	18151 (0%)	391568 (32%)	2159 (99%)	17.423
twoSpiral1	5311 (0%)	17672 (1%)	228 (10%)	11.519
twoSpiral2	10303 (0%)	35684 (2%)	452 (20%)	11.406
twoSpiral3	15295 (0%)	53188 (4%)	676 (31%)	19.025
twoSpiral4	20287 (0%)	70903 (5%)	900 (41%)	25.345
twoSpiral5	10393 (0%)	93953 (7%)	1332 (61%)	14.937
twoSpiral6	17051 (0%)	223580 (18%)	2160 (100%)	18.481

number of weights in the full FPNNs. While links in the reduced FPNNs contain usually up to three weights, the links in full FPNNs can dispose of tens of weights. Reduced FPNNs also can generally operate on higher clock frequencies.

VI. CONCLUSIONS AND FUTURE RESEARCH

In this paper, the formal definitions of the FPNA/FPNN concept were presented. The definitions of the new derived types were introduced. The process of direct transformation of the trained neural network to FPNN and the related algorithms were described. The diagrams of the FPGA implementation were presented. The experiments determining the approximation capabilities of different FPNNs of the reduced and the full type were run and their results were presented in this paper. The results show that in some cases, the reduced FPNN type is capable of good approximation performance. However, this depends on the concrete neural network and its weight values and their combinations. Therefore, the reduced FPNN are not suitable for all neural network implementations.

One of the main ideas of this paper was to show the possible trade-off between neural network approximation accuracy and the FPGA resources consumption. The experiments showed that reduced FPNNs consume significantly less resources than full FPNNs and that they are faster as well. On the other hand, the reduced FPNNs offer limited approximation accuracy compared to the accurate full FPNNs.

During the future research, we are going to perform experiments using our more advanced mapping techniques to increase the usability of reduced FPNNs as well as develop new methods and optimizations. Also we are going to include more types of neural networks into our research.

ACKNOWLEDGMENT

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602, ARTEMIS JU under grant agreement no 641439 (ALMARVI) and BUT project FIT-S-14-2297.

REFERENCES

[1] Girau, B.: FPNA: Concepts and Properties. In *FPGA Implementations of Neural Networks*, A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, pp. 71–123, http://dx.doi.org/10.1007/0-387-28487-7-3

[2] Girau, B.: Digital hardware implementation of 2D compatible neural networks. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 2000, ISSN 1098-7576, pp. 506–511 vol.3

[3] Girau, B.: On-chip learning of FPGA-inspired neural nets. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, 2001, ISSN 1098-7576, pp. 222–227 vol.1

[4] Prechelt, L. P.; Informatik, F. F.: — A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical report, Universitat Karlsruhe; 76128 Karlsruhe, Germany, 1994.

[5] Krcma, M.: *The neural networks acceleration in FPGA*. Master's thesis, Faculty of Information Technology, Brno University of Technology; Brno, 2014. https://wis.fit.vutbr.cz/FIT/st/rp.php/rp/2013/DP/15754.pdf

[6] KRCMA Martin, KASTIL Jan a KOTASEK Zdenek: *Mapping trained neural networks to FPNNs.* In: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. Belgrade: IEEE Computer Society, 2015, pp. 157–160. ISBN 978-1-4799-6779-7.

[7] Krcma, M.; Kotasek, Z.; Kastil, J.: Fault tolerant Field Programmable Neural Networks. In *Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC), 2015*, Oct 2015, pp. 1–4, 10.1109/NORCHIP.2015.7364381.

[8] Holt, J.; Baker, T.: Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume II, July 1991, pp. 121 –126 vol.2.

[9] Munakata, T.: Neural Networks: Fundamentals and the Backpropagation Model. In *Fundamentals of the New Artificial Intelligence*, editace T. Munakata, Texts in Computer Science, Springer London, 2007, ISBN 978-1-84628-839-5, pp. 7–36, http://dx.doi.org/10.1007/978-1-84628-839-5--2

[10] Bohrn, M.; Fujcik, L.; Vrba, R.: Field Programmable Neural Array for feed-forward neural networks. In *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, 2013, pp. 727–731

[11] Harkin, J.; McDaid, L.; Hall, S.: Programmable architectures for large-scale implementations of Spiking Neural Networks. In *IET Irish Signals and Systems Conference (ISSC 2008)*, June 2008, ISSN 0537-9989, pp. 374–379

[12] Harkin, J.; Morgan, F.; Hall, S.: Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks. In *2008 International Conference on Field Programmable Logic and Applications*, Sept 2008, ISSN 1946-147X, pp. 483–486

[13] Kwan, H.: Simple sigmoid-like activation function suitable for digital hardware implementation. *Electronics Letters*, 1992: pp. 1379–1380. http://link.aip.org/link/?ELL/28/1379/1