Triple Modular Redundancy Used in Field Programmable Neural Networks

Martin Krcma, Zdenek Kotasek and Jakub Lojda
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
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 discusses the usage of TMR technique in order to reach a fault tolerance. The diagrams of the FPGA implementation are presented. The results of experiments determining the FPGA resources utilization with different usage of the TMR technique are provided.

1 Introduction

The artificial neural networks [14] are one of the important models of softcomputing and artificial intelligence. They are structure inspired by the human brain with 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 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 *basis* function (often 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 - Field Programmable Neural Arrays/Networks (FPNAs/FPNNs).

In our paper published at NORCAS 2015 conference [12], 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.

This paper is organised as follows - the first section introduces the FPNA/FPNN concept. The second section describes the implementation of FPNNs into FPGAs. The third section deals with fault tolerance techniques. The fourth section presents the experimental results and the last section summarizes the whole paper.

2 Field Programmable Neural Networks

The concept of FPNNs [4] is meant to simplify the implementation of artificial neural networks in FPGAs by adjusting their properties to be more suitable for implementation into them. 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 and to simplify the interconnection model. The FPNNs are composed of dedicated interconnected units called neural resources which approximate the original neurons and synaptic interconnections. The units of the first type are called *activators* and represent the original neural network neurons. The other units are called *links* and serve as an approximation of the original synaptic interconnection. Every link disposes of a set of weights serving as an approximation

978-1-5386-3299-4/17/$31.00 ©2017 IEEE

of the original synaptic weights.

An example of a grid FPNN can be seen in Fig. 1. The circles in the figure 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.

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 - with different structure and properties, they can provide similar results as the networks. The accuracy is the main problem here. Since the FPNNs can be constructed in various ways and types, the approximation accuracy can be different. We dealt with the approximation accuracy in [11].

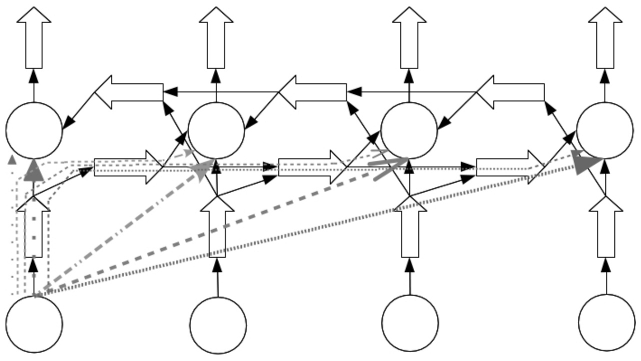
Figure 1. Synapses approximation in a grid FPNN

2.1 Implementation of FPNNs into FPGAs

The VHDL implementation of both types was created according to the original design and schematic [4]. 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 activator number and it is used in links to select the proper weight to multiply the input data width. The links then propagate the flag to all connected links.

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 SEL 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 SEL 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.

- **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 SEL block - 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 SEL 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.

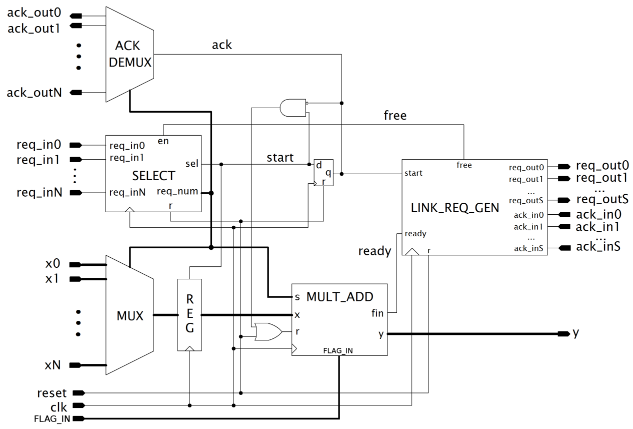

Figure 2. Diagram of a link implementation - the interconnection of the inner units

3 Fault tolerant FPNN using TMR

The neural networks are parallel structures with lot of redundancy performing an approximate (soft) computing. Therefore they dispose of inherent fault tolerant properties which differ with every network though. There are a number of techniques designed to increase the fault tolerant properties. Some do it using an additional redundancy on different levels (for example TMR on the level of the whole network [18], or adding redundant neurons [1]) in order to build the network to be fault tolerant. Others modify the process of training [2] in order to train the network to be fault tolerant (for example [16, 5, 6, 9] uses weight minimization during the training, [8] describes a usage of fault injection during the training) or use retraining after a fault occurs [3]. Others modify the basis function [15] or activation function [17, 10]. All approaches are combined as well.

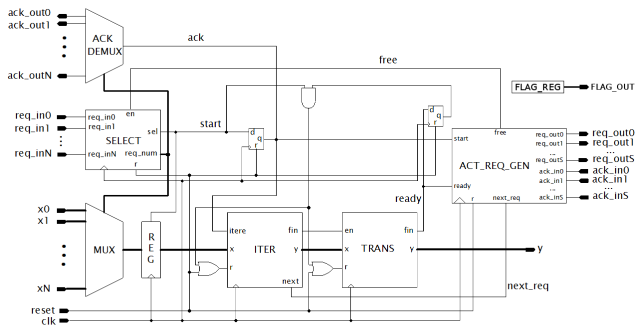

Figure 3. Diagram of an activator implementation - the interconnection of the inner units

Our approach based on extension of the implemented neural network fault tolerance (or substitution if no fault tolerant technique was used on the network). Different approaches can be used to make an FPNN fault tolerant. The approaches can utilize replication or can use other principles. We introduced a fault tolerance technique which does not use replication in [12]. The homogeneous structure of FPNNs is suitable for using replication based techniques as well as for the recovery using the online reconfiguration. The asynchronous model of communication is suitable for this type of recovery as well since the FPNN can be simply put on hold until the recovery is finished and then resume its function without a need of resetting the whole FPNN.

TMR is a well known fault tolerant technique based on triple replication of the secured unit and comparison of the triplet output data in order to determine the major result which is then used as the output of the whole triplet. This technique can be used on different levels with FPNNs. In this paper we focus on two levels - the level of inner units (we will refer to this type as *type A*) and the level of the whole neural resources (*type*

B).

On the level of inner units (type A), there are six (link) or seven (activator) main units which can be secured using the TMR technique. Using the technique on this level has several advantages. It allows us to choose which units (if not all) will be secured. Therefore it allows us to adjust the security/overhead ratio. Also, if we focus on the smaller inner units, the recovery from fault using the dynamic reconfiguration will be easier and faster than in the case of reconfiguration of the whole neural resources. However, on this level the interconnection between units will not be secured. This will occur on the level of the whole neural resources (type B). On this level, the fault tolerance will be generally higher because of duplication the whole resources but overhead will be higher as well. Also, recovery from fault using dynamic reconfiguration will be more complicated and slower due to larger reconfigured area.

We decided to compare these two levels in the meaning of area utilization in order to have a base for decision which level of TMR will be used. There are other criteria to evaluate the fault tolerant techniques. The power consumption, maximum clock frequency and latency belong to the most important. However, in this paper we deal with area (resource utilization) only, with other criteria we shall deal with in our future research.

4 Experimental results

In order to determine the area usage (in the number of slice registers and LUTs) of the neural resources and their inner units in both the unsecured and the TMR versions, we implemented them in VHDL and synthesized them using the Xilinx ISE 14.7 tool. The target FPGA was the Xilinx Virtex-6 device *xc6vlx240t-1-ff1156*. All computations were implemented in fixed point form with 8 bits of the integer part and 8 bits of the fractional part [7]. The voters were implemented using bit operations, therefore the voting was performed on the level of bits. All neural resources were implemented to be connected with three predecessors and two successors. The number of connected neural resources affects the size of the communication units. The link has three weights with real values. The use of DSP blocks was switched off. The optimization level was left on default but the *Equivalent registers removal* option was switched off to avoid the drop of the duplicated units. All results were provided by the synthesis only.

The resources utilization of the unsecured units are shown in Table 1. In the table, the *Unit* column identifies the units by their name. The columns *Slice Regs.* and *Slice-LUTs* contain the resources utilization. The columns *LUTs of act.* and *LUTs of link* compare the LUTs utilization of the unit with the utilization of the whole neural resources in order to illustrate the area portion of the inner units.

Table 1. Utilization of unsecured blocks

Unit	Slice Regs.	Slice-LUTs	LUTs of act.	LUTs of link
SELECT	8	17	1%	1%
ITER	49	104	6%	0%
TRANS	1	1478	86%	0%
MULT-_ADD	0	1389	0%	88%
REQ-_GEN	2	5	0.3%	0.3%
ACTI-VATOR	126	1723	100%	0%
LINK	57	1582	0%	100%

As the table illustrates the most significant portion of FPGA resources are utilized in the computation units MULT_ADD and TRANS. The communication and control units utilize around one percent of resources and around 5%-10% of resources are utilized by the units interconnection. This shows that the computation units are the best candidates for using the TMR technique as the probability of failure is the highest with them. On the other hand, the communication and control units are essential for the data flow, therefore for the functionality of the whole FPNN and the failure in these units could stop the operation of the whole FPNN, while the failure in the computing unit could only cause the degraded precision. Moreover according to their low resources utilization, it could be suitable to apply the TMR technique to secure them.

Table 2 summarizes the resource utilization of the building blocks secured using the TMR technique. The columns *Regs.* and *LUTs* have the same meaning as in Table 1, the other columns contain the percent increase of the resources utilization. As expected, the resource utilization has increased approximately three times or less in most of the units. The neural resources are marked by the used TMR type. The utilization of the type A activator (all units were TMR secured although not all of them are listed in the table) has increased around 2.7 times and the utilization of the type A link around 2.9 times (registers) and 2.3 times (LUTs). The type B resources LUTs utilization has increased even more but their registers utilization has increased less than in case of type A resources.

Table 2. Utilization of TMR-secured blocks

Secured unit	Regs.	LUTs	Increase of Regs.	Increase of LUTs
SELECT	28	40	250%	135%
ITER	179	331	265%	218%
TRANS	83	4291	83	190%
MULT-_ADD	83	3641	83	162%
REQ-_GEN	6	6	200%	20%
ACTIV-ATOR_-TYPE_A	346	4699	174%	173%
LINK_-TYPE_A	166	3737	191%	136%
ACTIV-ATOR_-TYPE_B	311	5611	147%	226%
ACTIV-ATOR_-TYPE_B	163	4299	185%	172%

Table 3. Comparison of different TMR levels

Neural resource (secured using type B)	Increase of registers utilization vs. type A	Increase of LUTs utilization vs. type A
Link	-2%	15%
Activator	-11%	19%

Table 3 compares the utilization of neural resources secured by the both TMR types. As the table illustrates, the type B neural resources consumes less registers than neural resources of type A. This is due to the number of voters consuming the registers in the type A resources. However, the type B neural resources consume more LUTs. This is due to interconnection included into the duplicates. It is needed to consider that there is around twice more registers than LUTs available in the FPGA. From this point of view the type A neural resources seems to be more resource and area efficient than type B resources. However, the type B resources are secured including the interconnection between inner units, therefore their fault tolerance should be higher.

5 Conclusions and future research

In this paper we briefly described the concept of FPNN serving for the implementation of artificial neural networks in FPGAs. We also described the implementation using the schematics and explained the construction of neural resources and the communication model. The fault tolerance techniques were considered and two levels of application of the TMR technique on the neural resources were discussed.

The application of the TMR technique on the inner units of the neural resources (type A) has proven to be less consuming in the meaning of the number of consumed LUTs, although this type consumed more registers. However, the registers are more available than LUTs, so this type seems to be more resource efficient. Due to smaller areas secured using TMR it might be more effective to use the dynamic reconfiguration in order to recover from fault.

The type of the TMR that secures the whole neural resources (type B) consumes less registers but more LUTs. In the meaning of the available resources, this type is less resource effective. Also, using the dynamic reconfiguration to recover from fault might be less effective and slower due to larger area needed to be reconfigured.

In our future research we will deal with other fault tolerance techniques. Especially with techniques which do not use the replication but they are based on a change of parameters and on the robustness of the FPNN which we designed. We shall also perform experiments with fault injection. We shall measure how the techniques affect the consumption and the value of the frequency on which the system works as well.

Acknowledgement

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] A. Ahmadi, M. H. Sargolzaie, S. M. Fakhraie, C. Lucas, and S. Vakili. A low-cost fault-tolerant approach for hardware implementation of artificial neural networks. In *Computer Engineering and Technology, 2009. ICCET '09. International Conference on*, volume 2, pages 93–97, Jan 2009.

[2] B. S. Arad and A. El-Amawy. On fault tolerant training of feedforward neural networks. *Neural Networks*, 10(3):539 – 553, 1997.

[3] J. Deng, Y. Rang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu. Retraining-based timing error mitigation for hardware neural networks. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pages 593–596, March 2015.

[4] B. Girau. Fpna: Concepts and properties. In A. R. Omondi and J. C. Rajapakse, editors, *FPGA Implementations of Neural Networks*, pages 63–101. Springer US, 2006. 10.1007/0-387-28487-7-3.

[5] T. Haruhiko, K. Hidehiko, and H. Terumine. Partially weight minimization approach for fault tolerant multilayer neural networks. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1092–1096, 2002.

[6] T. Haruhiko, K. Hidehiko, and H. Terumine. Fault tolerant training algorithm for multi-layer neural networks focused on hidden unit activities. In *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pages 1540–1545, 2006.

[7] J. Holt and T. Baker. Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume ii, pages 121 –126 vol.2, jul 1991.

[8] T. Ito and I. Takanami. On fault injection approaches for fault tolerance of feedforward neural networks. In *Test Symposium, 1997. (ATS '97) Proceedings., Sixth Asian*, pages 88–93, Nov 1997.

[9] N. Kamiura, T. Isokawa, and N. Matsui. Learning based on fault injection and weight restriction for fault-tolerant hopfield neural networks. In *Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings. 19th IEEE International Symposium on*, pages 339–346, Oct 2004.

[10] N. Kamiura, Y. Taniguchi, T. Isokawa, and N. Matsui. An improvement in weight-fault tolerance of feedforward neural networks. In *Test Symposium, 2001. Proceedings. 10th Asian*, pages 359–364, 2001.

[11] M. Krcma, J. Kastil, and Z. Kotasek. Mapping trained neural networks to fpnns. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2015 IEEE 18th International Symposium on*, pages 157–160, April 2015.

[12] M. Krcma, Z. Kotasek, and J. Kastil. Fault tolerant field programmable neural networks. In *Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC), 2015*, pages 1–4, Oct 2015.

[13] H. Kwan. Simple sigmoid-like activation function suitable for digital hardware implementation. *Electronics Letters*, 28(15):1379–1380, 1992.

[14] T. Munakata. Neural networks: Fundamentals and the backpropagation model. In T. Munakata, editor, *Fundamentals of the New Artificial Intelligence*, Texts in Computer Science, pages 7–36. Springer London, 2007. 10.1007/978-1-84628-839-5–2.

[15] A. Rusiecki. Fault tolerant feedforward neural network with median neuron input function. *Electronics Letters*, 41(10):603–605, May 2005.

[16] H. Takase, H. Kita, and T. Hayashi. Weight minimization approach for fault tolerant multi-layer neural networks. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, volume 4, pages 2656–2660 vol.4, 2001.

[17] Y. Taniguchi, N. Kamiura, Y. Hata, and N. Matsui. Activation function manipulation for fault tolerant feedforward neural networks. In *Test Symposium, 1999. (ATS '99) Proceedings. Eighth Asian*, pages 203–208, 1999.

[18] F. Zarafshan, G. Latif-Shabgahi, and A. Karimi. Notice of retraction a novel weighted voting algorithm based on neural networks for fault-tolerant systems. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 9, pages 135–139, July 2010.