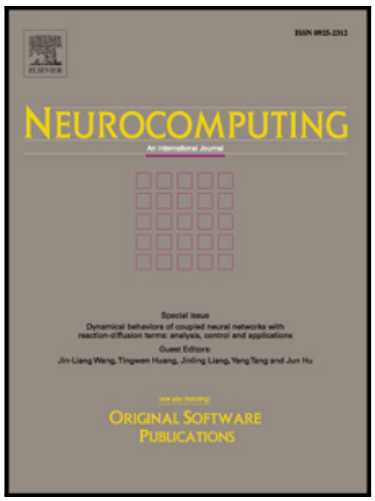

Reconfigurable FPGA implementation of neural networks

Zbigniew Hajduk

PII: S0925-2312(18)30539-3
DOI: 10.1016/j.neucom.2018.04.077
Reference: NEUCOM 19558

To appear in: *Neurocomputing*

Received date: 9 October 2017
Revised date: 16 March 2018
Accepted date: 30 April 2018

Please cite this article as: Zbigniew Hajduk , Reconfigurable FPGA implementation of neural networks, *Neurocomputing* (2018), doi: 10.1016/j.neucom.2018.04.077

Reconfigurable FPGA implementation of neural networks

Zbigniew Hajduk* zhajduk@kia.prz.edu.pl
Rzeszów University of Technology, ul. Powstańców Warszawy 12, 35-959 Rzeszów, Poland.
Corresponding author. *

Abstract

This brief paper presents two implementations of feed-forward artificial neural networks in FPGAs. The implementations differ in the FPGA resources requirement and calculations speed. Both implementations exercise floating point arithmetic, apply very high accuracy activation function realization, and enable easy alteration of the neural network's structure without the need of a re-implementation of the entire FPGA project.

Keywords

FPGA; Neural networks

1. Introduction

Most of the existing artificial neural networks (ANNs) applications, particularly for commercial environment, are developed as software. Yet, the parallelism offered by hardware may deliver some advantages such as higher speed, reduced cost, and higher tolerance of faults (graceful degradation) [1, 2]. Among various developed methods of ANNs implementations in field programmable gate arrays (FPGAs), e.g., [3 - 6], there is a breed of implementation which allows the structure of the ANN (i.e., the number of layers and/or neurons, etc.) to be altered without the need of re-synthesizing and re-implementation of the whole FPGA project. This feature increases the ANNs implementation flexibility to the similar level as offered by software, at the same time maintaining the advantages delivered by hardware. Unfortunately, existing solutions, e.g., [7 - 9], are based on fixed point arithmetic, have strongly limited calculations accuracy of the activation function, and require dedicated software tools for the formulation of a set of user instructions controlling the ANN calculations in the developed hardware. Some of them [9, 10] do not employ parallel architecture exploiting only a single neuron block for the calculations of the whole ANN. In

the case of [10] floating point (FP) arithmetic is used and a relatively high accuracy of the activation function is achieved, however the feasibility of the alteration of the ANN structure without reimplementation of the whole project is heavily compromised.

In this brief paper the FPGA implementations of feed forward ANNs, namely the resource-saving and parallel, are presented. The resource-saving implementation characterizes considerably lower calculations speed than the parallel one, but requires remarkably lower number of FPGA resources. Both implementations employ single precision floating point arithmetic and apply a very high accuracy algorithm for the activation function calculation with the Padé approximation of the exponential function. This enables the direct exploitations of the ANN's weights and biases values calculated off-line, e.g., by the Matlab software. The important feature of the proposed implementations is that the structure of the ANN can easily be changed (even on-line during the system operation) by the replacement of the FPGA Block RAM memory content without the usage of FPGA synthesis tools. The aforementioned traits of the developed implementations make them a solid and versatile candidate for a hardware accelerator of ANNs calculations. Particularly, the Padé approximation of the exponential function as well as the usage of the block RAM memory for the ANN's structure definition also constitute a novelty of the proposed solution.

2. Resources-saving implementation

The resource-saving implementation employs only one or two pairs of the floating point (FP) multiplier-adder blocks (actually, two versions of this implementation have been considered - the usage of two pairs of FP blocks shortens overall calculations time but simultaneously increases the FPGA resources requirement). It calculates the result in a serial manner - neuron by neuron and layer by layer. However, as many FP operations are performed simultaneously as possible, e.g., for the version with two FP multiplier-adder pairs, two additions and two multiplications can be accomplished at the same time.

The architecture of the resource-saving implementation with the single pair of the FP multiplier-adder blocks is presented in Fig. 1. The bold lines denote multi-bit buses. The architecture is comprised of the FP

adder (FPADD), multiplier (FPMUL) and divider (FPDIV), ROM memory block for the storage of certain values of the exponential function, distributed RAM block for the storage of the intermediate calculations results, Block RAM memory containing the weights and biases as well as the description of the ANN's structure, input multiplexer for the selection of the ANN's inputs $X_0,...,X_{K-1}$, output D flip-flops ($FD_0,...,FD_{K-1}$) for the storage of the ANN's output values $Y_0,...,Y_{K-1}$, and the control unit (CU). The ND external signal activates the calculations, whereas the output RDY signal indicates the readiness of the calculations result. Each of the FP blocks has three data buses dubbed *x*A (the first operand), *x*B (the second operand), *x*Y (the operation result), and two control signals named *x*ND (activation of the operation) and *x*RDY (readiness of the operation result), where here the *x* prefix denotes the type of the operation (ADD, MUL, DIV). The AOP signal for the FPADD block determines whether the addition (AOP=0) or subtraction (AOP=1) operation is performed. The Block RAM memory block as well as the distributed ROM block have only two buses, namely the address bus (BADR and ROMADR for the former and latter aforementioned blocks, respectively) and output data bus (BDO, ROMDO). The distributed RAM has two separate ports: the first one for the synchronous write operation and the second one for the asynchronous read operation. For both ports, the address buses (WRADR, RDADR) consist of the concatenation of the SEL signal or its negation (the MSB bit of the buses) and the DADR or MS signals (the LSB bits of the buses). The DDI is the data input, whereas the DDO bus denotes the data output. The DWE signal enables the write operation.

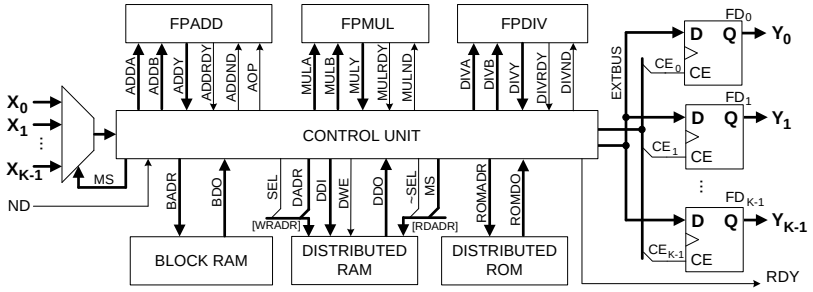

Fig. 1. Architecture of the resources-saving implementation

Detailed algorithmic state machine (ASM) diagram, describing the CU operations, is presented in Fig. 2. The signals' names denoted using capital letters relate to external signals and buses showed in Fig. 1, whereas non-capital letters are exploited to mark the internal variables (registers).

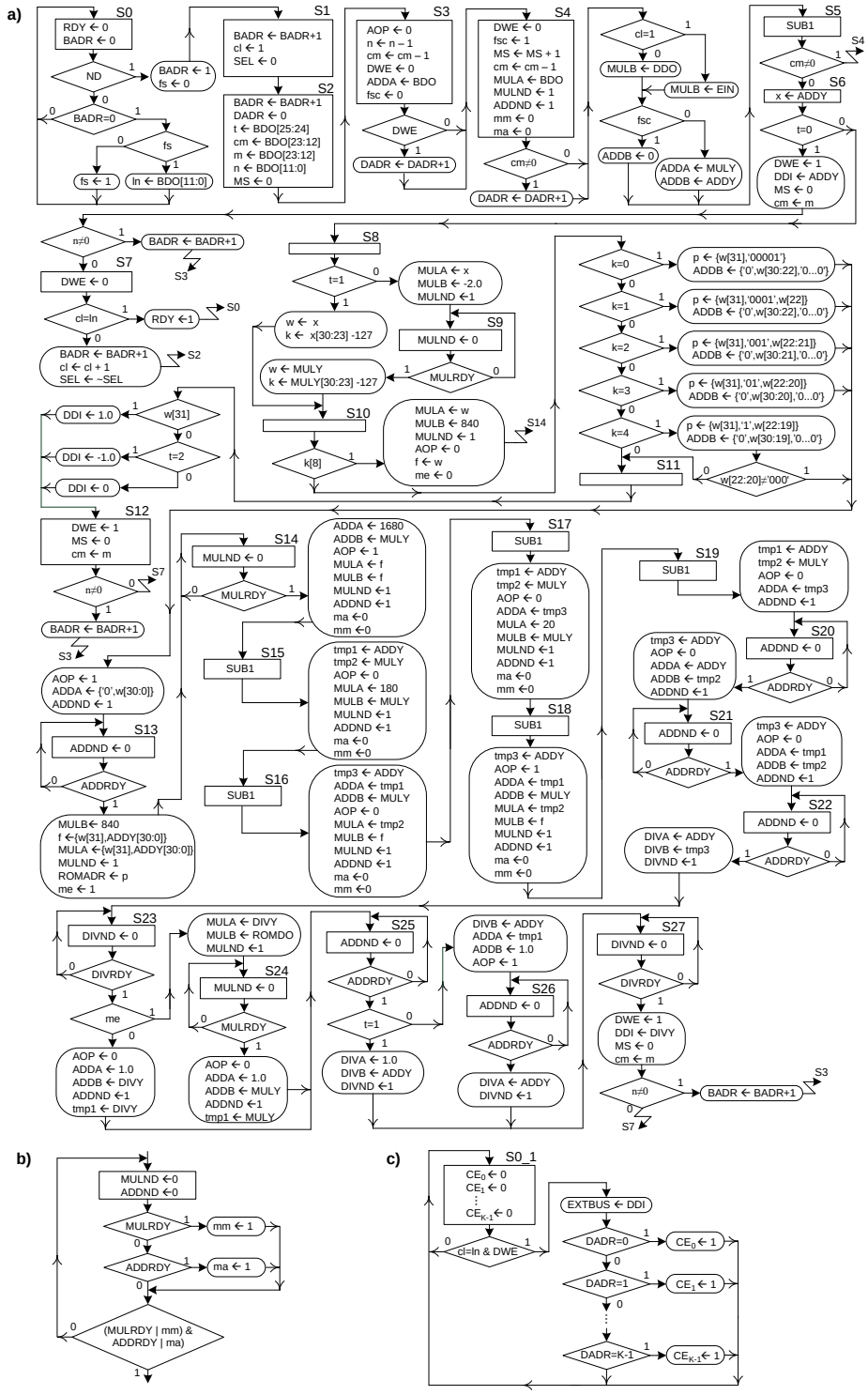

Fig. 2. The ASM for the control unit (a), structure of the SUB1 block (b) and sub-ASM for the output flip-flops (c)

The diagram applies syntax elements similar to those used in the Verilog hardware description language, particularly the bit-select, part-select and concatenation operators. The empty rectangular boxes (i.e., S8 and S11) indicate that no unconditional operations are performed in the current state. States S1-S7 of the ASM from Fig. 2a are responsible for the subsequent neurons and the ANN's layers calculations, whereas states S8-S27 deal with the activation function calculations. Three types of activation functions are supported, namely the hyperbolic tangent, sigmoid and linear.

The realization of a neuron's non-linear activation function constitutes a pivotal element of any FPGA implementation of ANNs. A number of solutions for this issue have been proposed, e.g., [4, 11, 12]. Unfortunately, they are usually focused on a mathematical modeling of the activation function providing too little details for a tangible digital reimplementation. The realization method of the activation functions, applied by the ASM from Fig. 2a, is based on the idea shortly described in [13]. However, instead of the McLaurin series interpolation of the exponential function, presented in detail in [13], the Padé approximation has been applied. This enables a shorter calculations time with only a slightly lower accuracy. The Padé approximation is only valid for a fractional part of the exponential function's argument; therefore in states S10-S13 of the ASM from Fig. 2a, the integer and fractional part (the *p* and *f* variables respectively) are determined. A binary representation of single precision FP numbers is heavily exploited for the determination of the integer part, whereas the fractional part is computed by a subtraction of the absolute values of the original exponential function's argument and the FP representation of the integer part. The exponential function value for the fractional part, approximated by the Padé rational function given by equation (1), is then calculated in states S14-S23.

$$e^{-x} \approx \frac{1680 - 840x + 180x^2 - 20x^3 + x^4}{1680 + 840x + 180x^2 + 20x^3 + x^4}. \qquad (1)$$

The exponential function values for the integer part of the function's argument are stored in the distributed ROM memory (actually, 18 values are stored) and, if necessary, a certain value is multiplied by the result of the calculation of equation (1). This happens in state S24. The remaining states (S25-S27) concludes the activation function calculations taking into consideration the actual function's type (hyperbolic tangent or

sigmoid). The conducted experiments show that, for the hyperbolic tangent function, the maximum absolute error, calculated according to the proposed method for 1E6 points equally spaced within the [-10, 10] interval, amounts to 2.384E-7 (which is a rather very high accuracy as the short survey form [13] reveals). It is of note that the increase of the Padé approximation order beyond the value from equation (1) as well as the increase of the number of values stored in the distributed ROM memory does not lead to the increase of the calculations accuracy of the activation function. It is also of note that the accuracy of the activation function calculations has an essential impact on the overall calculations accuracy of an ANN.

The actual structure of the ANN can be defined by a certain layout of the Block RAM memory content. The layout used by both considered implementations is depicted in Fig. 3. The Block RAM memory arranges a number of 32-bit words. The first word (at the 0000h address) contains the number of ANN's layers (L) and then follows the data of subsequent layers. Each data layer, in turn, consists of the layer information word (LIW) and the subsequent weights and biases for all neurons in the layer. The layer information word includes the number of neurons in the layer (N), number of weights/inputs of the single neuron (M), and the activation function type (T).

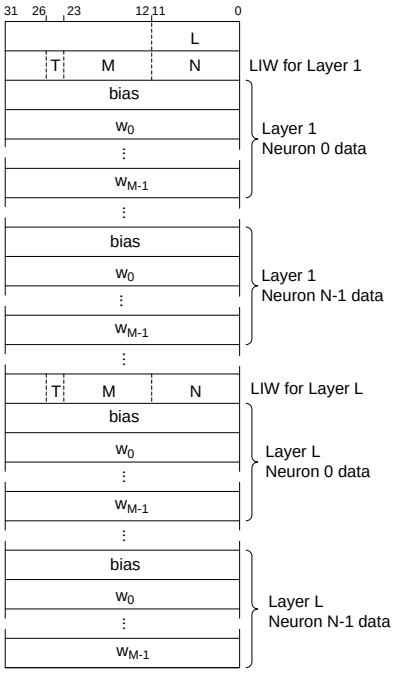

Fig. 3. Layout of the RAM memory content defining the ANN's structure

3. Parallel implementation

Contrary to the resource-saving implementation the parallel implementation arranges simultaneous calculations of all neurons within a single ANN's layer. The pivotal component of the implementation is the neuron block whose general architecture (for the version with two pairs of FP multiplier-adder blocks) is presented in Fig. 4. The neuron block calculates the neuron's output value (Y) taking into account the neuron's inputs (i_0,...,i_{M-1}), weights (w_0,...,w_{M-1}), bias input, and auxiliary (aux) information which include the activation function type and number of currently used inputs/weights. The ND and RDY signals, as well as signals connecting the FP blocks and distributed ROM block with the control unit, perform the same function as for the architecture from Fig. 1.

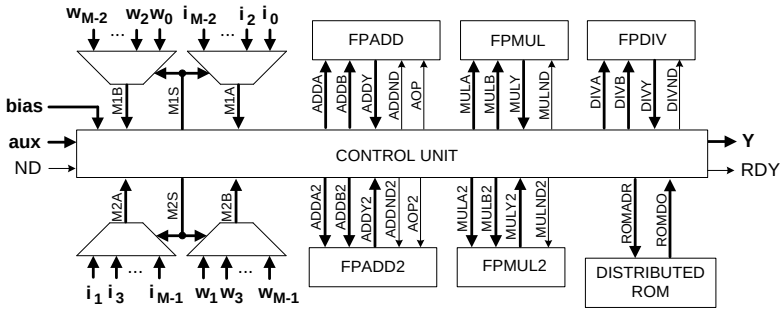

Fig. 4. Architecture of the neuron block for the parallel implementation

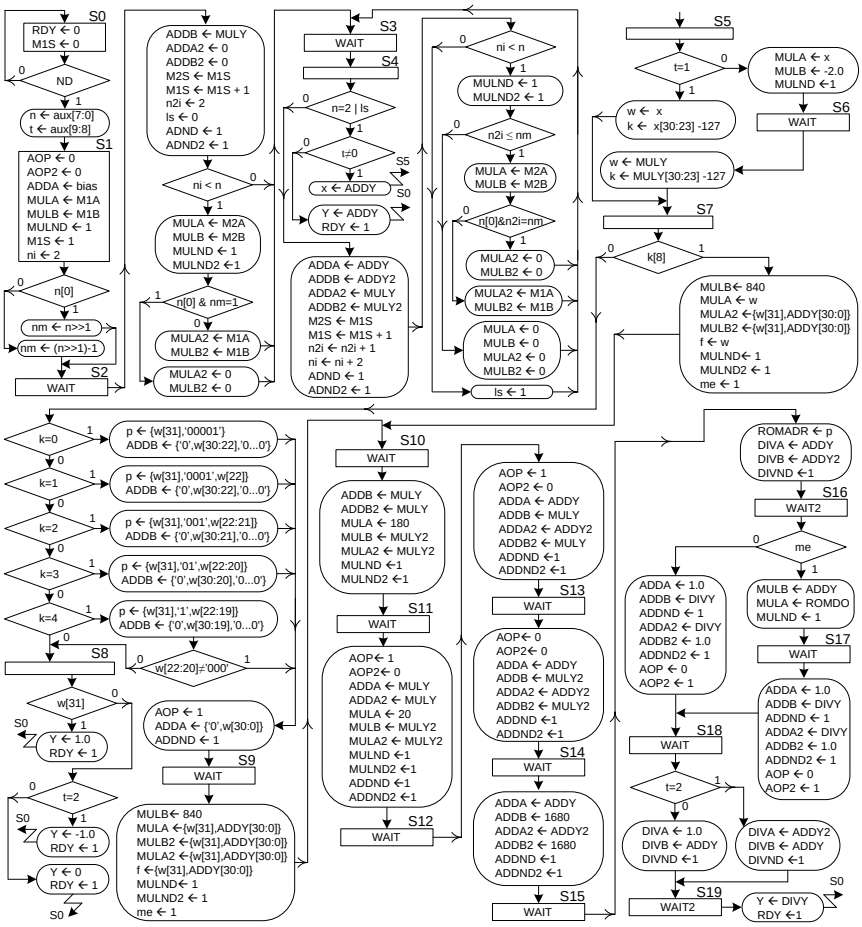

Fig. 5. The ASM for the control unit from Fig. 4

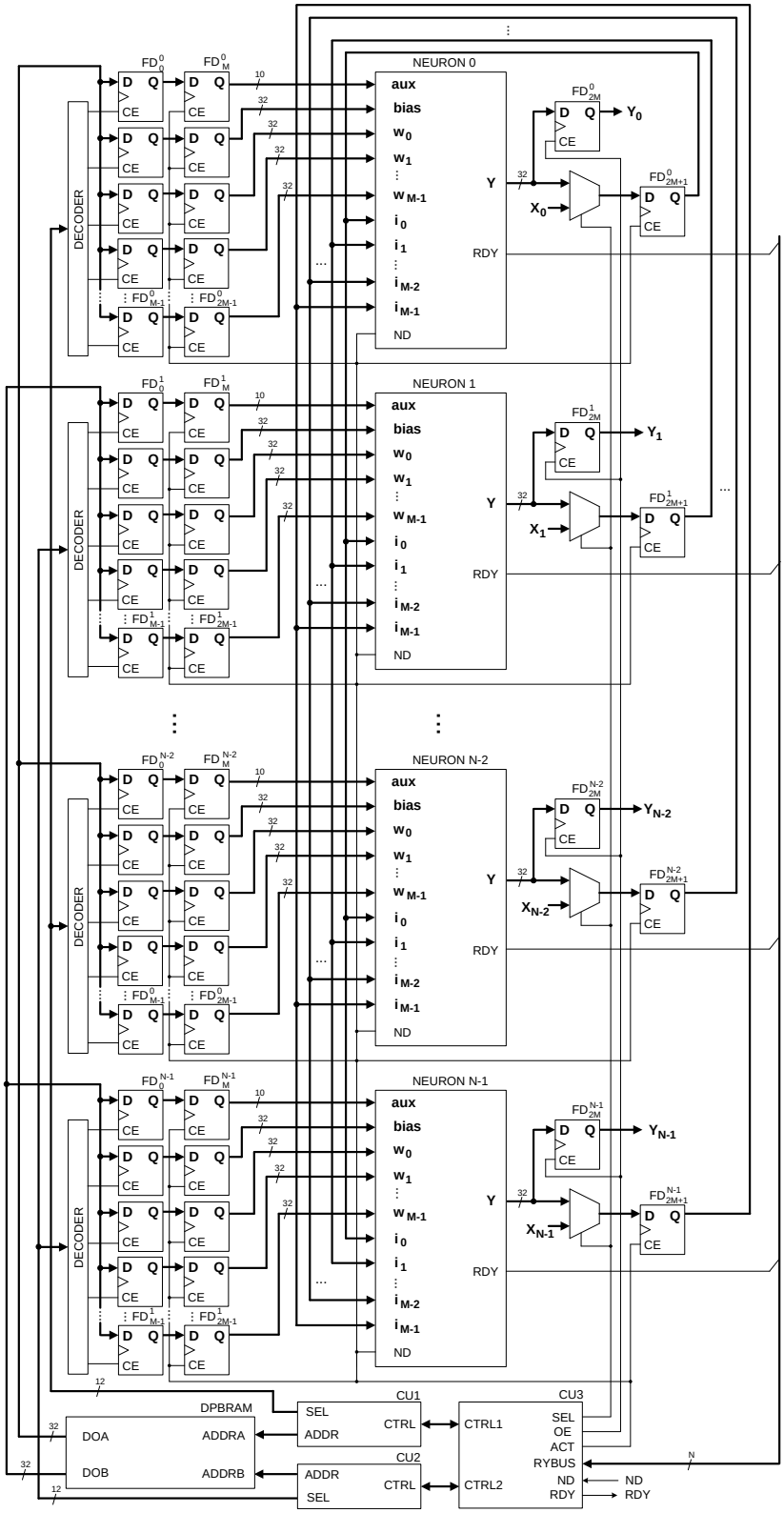

Fig. 6. Architecture of the parallel implementation

The ASM diagram describing operations performed by the neuron block is featured in Fig. 5. Similarly to the previously considered ASM, the signals and variables names denoted using capital letters (with the

exception of the aux and bias signals) relate to external signals and buses depicted in Fig. 4. States S1-S4 of the ASM are responsible for the calculation of the sum of products of the neuron's inputs and weights. The exponential function value is computed within states S5-S16 (the same idea with the Padé approximation of the fractional part is exploited). The actual neuron's activation function value is calculated in states S17-S19. Contrary to the architecture from Fig. 1 and the corresponding ASM, the readiness signals of the FP arithmetic blocks are not exercised by the considered ASM. Instead, a simple wait operation for a certain number of clock cycles, within which the FP block completes its calculations, is applied. Thus, the WAIT and WAIT2 operational blocks from Fig. 5 simply inserts the delay of 2 and 8 clock cycles respectively (the FP arithmetic blocks have been configured to perform their operations within as little clock cycles as possible, maintaining at the same time as high maximum clock frequency as possible). Although this solution is slightly less versatile, it enables a certain reduction of the FPGA resources requirement.

The complete architecture of the parallel implementation of ANNs is presented in Fig. 6. It consists of a number of the neuron blocks, dual-port block RAM memory (DPBRAM) for the storage of the ANN's structure, and three separate control units (CU1, CU2, CU3). Each of the neuron blocks is surrounded by two sets of D flip-flops (FD^j_0,..., FD^j_{M-1} and FD^j_M,..., FD^j_{2M-1} respectively, where j denotes the neuron's number) with the decoder block for the storage of weights, bias and auxiliary information, ANN's output D flip-flops (FD^j_{2M+1} and FD^j_{2M} respectively), and ANN's input multiplexers. The external buses X_0,..., X_{K-1} and Y_0,...,Y_{K-1} are the inputs and outputs of the ANN respectively, whereas the ND and RDY signals perform the same function as in the case of the resource-saving implementation.

It is of note that at the same time when the neuron blocks perform calculations, the weights and biases for the next ANN's layer are simultaneously read from the DPBRAM memory and written to the selected D flip-flops from the first set of flip-flops. Since the DPBRAM is a true dual-port memory, the weights for two neuron blocks are loaded simultaneously (the first port of DPBRAM deals with even neuron blocks whereas the second engages odd neuron blocks). This shortens the time needed for the preparation of data for the calculations of the next ANN's layer.

The CU3 block controls all of the calculations and initiates the operation of the CU1 and CU2 blocks. The latter two blocks control, in turn, the data exchange process between the DPBRAM and the $FD^j_0,...,FD^j_{M-1}$ sets of flip-flops. The information from these flip-flops are copied to the $FD^j_M,..., FD^j_{2M-1}$ sets of flip-flops when all data pertaining to the processed layer are read and all of the neuron blocks are ready to start new calculations.

4. Resources requirement, calculations speed and accuracy

In order to evaluate the FPGA resources requirement, the developed architectures (described in Verilog hardware description language) have been implemented in the ZedBoard evaluation board with the moderate density Xilinx Zynq XC7Z020 chip. For the resources-saving implementation it was assumed that the maximum number of neurons within a layer as well as inputs of a single neuron block and inputs/outputs of the whole ANN amounts to 32. For the parallel implementation this number was limited to 16. The number of ANN's layers for both implementations is only limited by the capacity of the block-RAM memory (2048 32-bit words were actually used).

Table 1. Number of the utilized resources and maximum clock frequency for the Xilinx Zynq XC7Z020

Version	LUTs	FFs	DSPs	F_{MAX} [MHz]
A	2232	1210	2	118.7
	(4.2%)	(1.1%)	(0.9%)	
B	3306	1326	4	119.8
	(6.2%)	(1.3%)	(1.8%)	
C	41297	33395	33	112.5
	(77.6%)	(31.4%)	(15.0%)	
D	51028	35655	65	109.3
	(95.9%)	(33.5%)	(29.5%)	

The implementation results including the number of used 6-input look-up tables (LUTs), flip-flops (FFs), digital signal processing blocks (DSPs), and maximum allowable clock frequency (F_{MAX}) are presented in Table 1. The device utilization in percentages, related to the maximum number of available resources, is also portrayed in parenthesis. Both resources-saving and parallel implementations have been prepared with two subversions, namely with the usage of the single pair of the FP multiplier-adder blocks and with two pairs of these FP blocks. The A and B versions from Table 1 refer to the resource saving implementations,

whereas the C and D versions indicate the parallel implementations. The A and C versions exploit a single pair of the FP multiplier-adder blocks, whereas the remaining versions apply two pairs of the FP multiplier-adder blocks. Note that for the parallel implementation the term pair of FP multiplier-adder blocks refers to the single neuron block whereas for the resources-saving implementation it denotes the absolute number of the FP blocks in the implementation.

For the evaluation of the calculations speed and accuracy an example of an auto-associative, 4-layer ANN with 16-12-16-5 neurons in subsequent layers and 5 inputs/outputs has been considered. The hyperbolic tangent activation function has been used with the exception of the 2-nd and 4-th layers (the bottleneck and output layers) where the linear activation function has been applied. The network has been trained using the Matlab software and then the developed simple program script has been used for the preparation of the DPBRAM memory content, according to the format from Fig. 3. The task of the trained ANN was to mimic on its outputs certain values delivered to the ANN's inputs.

Table 2. Calculations times of the test application

Platform	Calculations time [ms]
PC desktop	8
Intel Core i7-950 @ 3.1GHz	
Raspberry Pi 2	53.3
ARM Cortex-A7 @ 900MHz	
Xilinx Zynq (Processing System part)	52.7
ARM Cortex-A9 @ 667MHz	
Xilinx Zynq (Programmable Logic part)	33.1
Version A @118.2MHz	
Xilinx Zynq (Programmable Logic part)	24.7
Version B @118.2MHz	
Xilinx Zynq (Programmable Logic part)	5.7
Version C @112.5MHz	
Xilinx Zynq (Programmable Logic part)	3.5
Version D @109.1MHz	

Table 3. Maximum absolute, average and MSE errors

Version	Maximum error	Average error	MSE errors
A	1.847E-6	2.905E-7	1.476E-13
B	1.549E-6	2.860E-7	1.525E-13
C	1.847E-6	2.906E-7	1.571E-13
D	1.430E-6	3.053E-7	1.624E-13

The test application performed 1E3 calculations of the prepared ANN with the ANN's inputs values changing within a certain range. The obtained test results, in terms of the calculations times and accuracy, are portrayed in Table 2 and Table 3. Apart from FPGA implementations the test application has also been implemented using popular microprocessor platforms (the software was written in C/C++). The calculations times for all platforms with the exception of the Intel processor have been measured by the external universal counter. For the PC platform the calculations time has been obtained by reading the system time before and after the calculations.

The obtained results from Table 1-2 show that the most resources-consuming implementation version (D) requires 22.9 times more LUTs than the most resource-saving implementation (A). However, the calculations time of the test application for the D version is only 9.5 times shorter than for the A version.

It is of note that the calculations times of the test application provided by all of the developed implementations are shorter than the times obtained by the software platforms with the ARM processors. Additionally, the two parallel implementations calculate the test application faster (1.4 and 2.3 times respectively) than the relatively high performance PC computer.

Table 3 shows the obtained accuracy of the FPGA implementations of the test application, conceived in terms of the maximum absolute, average and mean square (MSE) errors. The errors have been calculated taking into account the simulation results of the test application for all of the implementation versions and the calculations results yielded by the PC software platform, used as the reference. The obtained errors only slightly differ between the implementation versions. Yet, the parallel implementation with two pairs of the FP multiplier-adder blocks seems to have the lowest maximum absolute error under the highest average error. It is of note that the obtained maximum error value for the calculations of the test ANN is almost one order of magnitude higher than the maximum error for the activation function calculations (1.847E-6 vs. 2.384E-7). This is possibly related to the cumulative rounding errors. Under the development of the neuron's activation function implementation it was also observed numerous times that the sequence in

which the FP calculations are performed has a considerable impact on the overall accuracy. The calculations sequence, particularly for the sum of products of the neuron's inputs and weights, in the case of the FPGA and software implementation is in fact different. This may contribute to the increase of the observed errors.

Making a direct ANN's calculations accuracy comparison between the proposed implementations and other solutions is not straightforward. The published works either do not mention the obtained accuracy or apply different errors measures and different ANN's structures for an accuracy test. However, one of the reliable results is reported by [10] where the MSE error no higher than 5E-8 has been obtained. The other work of [7] reports, in turn, the maximum error amounting to 8E-2. For the proposed implementation the MSE and maximum errors are five and four orders of magnitude lower, respectively.

5. Conclusions

It has been shown that the FPGA implementations of ANNs may characterize high flexibility as well as high calculations speed and reasonable accuracy in comparison with software realization of ANNs. On account of the application of floating point arithmetic and very high accuracy of the activation function calculation, the ANN can be trained off-line, e.g. by the Matlab software or using processing systems with the ARM processors in such platforms as the Xilinx Zynq, and then the calculated weights can be directly used by the developed implementations. The feasibility of the alteration of the ANN's structure by a simple change of the RAM memory content makes the developed solution more flexible. Any existing methods or the previously developed communication module for the P1-TS system [14] can be exploited for the replacement of the FPGA-embedded block RAM memory content. The developed implementations can also be applied as hardware function blocks for the previously developed multiprocessor programmable controller [15], accelerating the calculations of ANNs.

The calculations speed of the developed implementations of ANNs seems to be high as well. The conducted tests reveal that even the simplest resources-saving implementations carries out the calculations faster than the ARM processors clocked with much higher frequency. The developed parallel implementations can also

be reasonably faster than the relatively high performance PC computer. However, it is of note that the exact relation between the calculations times of the software platforms and the developed parallel implementations depends on the actual ANN's structure (the more neurons a ANN's layer contains, which can be calculated in parallel, the more the parallel implementation is faster in relation to a software solution). As far as the calculations accuracy of the proposed implementations is concerned, it is not as high as the accuracy of the activation function implementation itself. However, it can still be considered as reasonable high.

The prepared architectures of the two implementation versions (i.e., the resource-saving and parallel) also constitute a good illustration for a known fact that the calculations speed advantage of hardware implementations over high performance software platforms comes from the parallelization of executed operations. The parallelism, however, can be obtained at the expense of a high FPGA resources requirement (higher density and more expensive FPGA chips are needed, etc.). Yet, if the resources requirement for the proposed parallel implementation of ANNs exceeds the acceptable level (e.g., the complete design is too large and does not fit into a selected FPGA chip) and a lower calculations speed is still acceptable, then the other presented implementation, namely resource-saving, can be applied.

References

[1] J. Misra, I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress", Neurocomputing, vol. 74, issues 1-3, pp. 239-255, 2010

[2] F. Morgado-Dias, R. Borralho, P. Fontes, A. Antunes, "FTSET-a software tool for fault tolerance evaluation and improvement", Neural Computing and Applications, vol. 19, issue 5, pp. 701-712, 2010.

[3] F.D. Baptista, F. Morgado-Dias, "Automatic general-purpose neural hardware generator", Neural Computing and Applications, vol. 28, issue 1, pp. 25-36, 2017.

[4] V. Tiwari, N. Khare, "Hardware implementation of neural network with Sigmoidal activation functions using CORDIC", Microprocessors and Microsystems, vol. 39, pp. 373-381, 2015.

[5] V.P. Nambiar, M. Khalil-Hani, R. Sahnoun, M.N. Marsono, "Hardware implementation of evolvable block-based neural networks utilizing a cost efficient sigmoid-like activation function", Neurocomputing, vol. 140, pp. 228-241, 2014.

[6] T. Orlowska-Kowalska, M. Kaminski, "FPGA Implementation of the Multilayer Neural Network for the Speed Estimation of the Two-Mass Drive System", IEEE Trns. on Industrial Informatics, vol. 7, no. 3, pp. 436-445, 2011.

[7] J.G. Oliveira, R.L. Moreno, O. de Oliveira Dutra, T.C. Pimenta, "Implementation of a reconfigurable neural network in FPGA", Int. Caribbean Conf. on Devices, Circuits and Systems, 2017, DOI: 10.1109/ICCDCS.2017.7959699

[8] A. Youssef, K. Mohammed, A. Nassar, "A Reconfigurable, Generic and Programmable Feed Forward Neural Network Implementation in FPGA", Int. Conf. on Computer Modelling and Simulation, 2012, DOI: 10.1109/UKSim.2012.12

[9] J. Renteria-Cedano, C. Peréz-Wences, L.M. Aguilar-Lobo, J.R. Loo-Yau, S. Ortega-Cisneros, P. Moreno, J.A. Reynoso-Hernández, "A novel configurable FPGA architecture for hardware implementation of multilayer feedforward neural networks suitable for digital pre-distortion technique", European Microwave Conference, 2016, DOI: 10.1109/EuMC.2016.7824478

[10] P. Ferreira, P. Ribeiro, A. Antunes, F. Morgado-Dias, "A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function", Neurocomputing, vol. 71, issue 1-3, pp. 71–77, 2007.

[11] I. Nascimento, R. Jardim, F. Morgado-Dias, "A new solution to the hyperbolic tangent implementation in hardware: polynomial modeling of the fractional exponential part", Neural Computing and Applications, vol. 23, issue 2, pp. 363-369, 2013.

[12] D. Baptista, F. Morgado-Dias, "Low-resource hardware implementation of the hyperbolic tangent for artificial neural networks", Neural Computing and Applications, Vol. 23, issue 3, pp. 601-607, 2013.

[13] Z. Hajduk, "High accuracy FPGA activation function implementation for neural networks", Neurocomputing, vol. 247, pp. 59-61, 2017.

[14] J. Kluska, Z. Hajduk, "Hardware implementation of P1-TS fuzzy rule-based systems on FPGA", Proc. Artif. Intell. Soft Comput., vol. 7894, pp. 282-293, 2013.

[15] Z. Hajduk, B. Trybus, J. Sadolewski, "Architecture of FPGA Embedded Multiprocessor Programmable Controller", IEEE Trans. Ind. Electron., vol. 62, no. 5, pp. 2952–2961, 2015.

Zbigniew Hajduk received Ph.D. degree (with honors) in computer engineering from the University of Zielona Góra, Zielona Góra, Poland, in 2006. He is an assistant professor at the Department of Computer and Control Engineering, Rzeszów University of Technology. His main area of interest includes digital systems design with FPGAs.