The Impact of Data Representationson Hardware Based MLP Network Implementation

Norbert Ádám, Anton Baláž, Emília Pietriková, Eva Chovancová, Peter Feciľak

Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
norbert.adam@tuke.sk, anton.balaz@tuke.sk, emilia.pietrikova@tuke.sk, eva.chovancova@tuke.sk, peter.fecilak@tuke.sk

Abstract: Artificial neural networks are massively parallel systems containing large amounts of simple computing elements. Therefore, it is natural to try to implement them using parallel computing architectures. This paper deals with an implementation of a three layer multilayer perceptron artificial neural network. It summarises the impact of using various forms of data representation on the performance of the hardware implementation – a Kintex-7 XC7K325T-2FFG900C FPGA chip on a Xilinx Kintex KC705 board.

Keywords: fixed-point; floating-point; mlp; neural network; residue number system

1 Introduction

Artificial neural networks belong to the category of massively parallel architectures. Therefore, it is natural to try to implement these structures – interalia –as ASIC circuits (a notable example of this being the NI1000 architecture [1]), the CogniMem CM1K chip [2] or the SyNAPSE [3] chip with a brain-inspired non-von Neumann computer architecture with a million neurons and 256 million synapses. Most solutions provide high performance for the cost of lower flexibility. The change of structure of an artificial neural network (such as the number of neurons, the applied algorithm, etc.) requires to create a new block design, meeting the requirements for artificial neural networks [4]. Currently, significant effort is being invested in the implementations of artificial neural networks on re-configurable computer platforms [5]. The term 're-configurable' refers to the capability of achieving the required system properties [6] beyond those of the conventional architectures (such as the von-Neumann architecture).

Field Programmable Gate Arrays (FPGA) belong to the category of re-configurable hardware. They allow to designlogic circuits similarly to the way that programmers write programs and their performance is comparable to the performance of application-specific integrated circuits (ASIC).

When implementing artificial neural networks in FPGA chips, one has to take the limiting factors of this implementation into account. This factor is the relationship between area and precision. The issue is that greater precision requires a higher number of logical structures required for their implementation. Precision itself is a very important factor of the speed of convergence of the artificial neural network to the desired output. Single precision representation of numbers provides sufficient precision (i.e. in terms of minimal quantization error and minimal misclassification rate); however, due to the limited hardware resources of FPGA chips, it is less efficient than in case of fixed point numbers.

2 Multilayer Feedforward Networks

The basic architecture of the multilayer perceptron (MLP) feedforward artificial neural network consists of three layers of neurons: the input, hidden and output layers. In feedforward networks, the signal is led from the input of the unit to the output strictly only forward.

2.1 Structure of Feedforward Multilayer Artificial Neural Networks

Feedforward multilayer neural networks belong to the artificial neural networks, which are used most often as universal means of classification and prediction. A three-layer neural network (containing at least one layer of hidden neurons) can simulate an arbitrary function *F* of type

$$F: \mathbb{R}^n \to]0,1[\tag{1}$$

where F is a continuous function with the projection of *n* dimensional space $\mathbb{R}^n$to the $]0,1[$ open interval.

With this, we have a universal tool for performing both regression analysis of functions defined by a training set and for extrapolation of functional values beyond the training set, i.e. a tool to solve generalisation problems (prediction and classification).

2.2 Backpropagation

The backward propagation of errors, or backpropagation (BP) proposed by Rumelhart, et al. [7], is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent.

The algorithm consists of two phases (Fig. 1): The forward phase where the activations are propagated from the input to the output layer, and the backward phase, where the error between the observed actual and the requested nominal value in the output layer (desired output) is propagated backwards in order to modify the weights and bias values.

The pseudocode of the algorithm is as follows:

```
Assign all network inputs and output
    Initialize network weights (small random values, typically between -1 and 1)
      do
        for every pattern in the training set
          Present the pattern to the network
            //Propagated the input forward through the network:
            for each layer in the network
              for every node in the layer
                  1. Calculate the weight sum of the inputs to the node
                  2. Add the threshold to the sum
                  3. Calculate the activation for the node
                end
            end
            //Propagate the errors backward through the network
            for every node in the output layer
              calculate the error signal
            end
            for all hidden layers
              for every node in the layer
                  1. Calculate the node's signal error
                  2. Update each node's weight in the network
              end
            end
            //Calculate Global Error
              Calculate the Error Function
          end
        while ((maximum number of iterations < than specified) AND (Error Function is > than specified))
```

Figure 1

Back-propagation algorithm

2.3 Activation Function

The individual neurons of the artificial neural networks use activation functions to calculate their own activation (i.e. output value). The argument of the activation function is the sum of the products of the weights and the outputs of the neurons of the preceding layer (or layers), connected to the particular neuron. In the available literary sources, one may find multiple applicable activation functions. The choice of the activation function may significantly influence the speed of the learning phase. The choice of the activation function depends on the type of the task we want to solve by using the artificial neural network. For the purpose of the BP we may use functions, for which a first order derivative of the activation function may be produced. In this paper we decided to use the logistic function, the derivative of which is defined as $\mathrm{y}' = \mathrm{f}'^{(x)} = \mathrm{y} \times (1 - y)$. This is the most commonly used soft-limiting activation function. Because it squashes the input range into $]0{,}1[$ output range.

3 Configuration of the Applied Artificial Neural Network

In addition to the applied target technology and platform, the overall performance of the artificial neural network is significantly influenced also by the applied algorithm, the form of data representation and the applied data structures. In this paper, when determining the overall performance of the proposed architecture, we will start out from the overall cost of implementing one forward and one backward phase of neural network training per epoch. For the configuration of the neural network, see Table 1.

The implemented neural network is aimed at the recognition of handwritten numbers, based on the MNIST database (Fig. 2). The MNIST database of handwritten digits has a training set of 60,000 samples (one epoch), and a test set of 10,000 samples. The digits have been size-normalized and centered in a fixed-size image.

The size of the individual figures – numbers ranging from 0 to 9 – is 28×28 pixels. Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black). Figure 2 shows a sample of handwritten digits.

The pixel inputs were normalised into a $[2^{-6}; 1 - 2^{-6}]$closed interval. The normalisation interval – other than $[0; 1]$ – was selected due to the applied activation function (sigmoid), the range of which is defined for the $]0; 1[$open interval. These normalised values were then used as input values of the input layer neurons. The required output values were also subjected to normalisation. The output layer consisted of 10 neurons, representing digits 0 to 9.

We used one-hot encoding, i.e. only a single neuron shall have the maximum value of $1 - 2^{-6}$, the other neurons will be set to the lowest value 2^{-6}, since the extreme values of 0 and 1 were otherwise unachievable, due to the sigmoid function.

Figure 2
MNIST digits (illustration)

The error function for training pattern p is given by

$$E_p = \frac{1}{2}\sum_k \left(d_{p,k} - y_{p,o,k}\right)^2 \tag{2}$$

where $d_{p,k}$ is the target output, and $y_{p,o,k}$ is the output layer output. If the error is 0.125 per output unit, the pattern error becomes

$$E_p = \frac{1}{2}\sum_{k=1}^{10} (0.125)^2 = 0.078125 \tag{3}$$

And this value was used as a threashold for determining if a pattern is trained or not. Weight change values were accumulated for a pattern p having $E_p > 0.078125$, whereas a pattern is trained when $E_p \leq 0.078125$.

Table 1
Configuration of the artificial neural network

Number of layers	***1 input layer, 1 hidden layer, 1 output layer***
Connection of neurons between the hidden layers	Full connection
Number of neurons in the input layer	28 × 28 = 768 neurons
Number of neurons in the hidden layer	500 neurons
Number of neurons in the output layer	10 neurons
Activation function applied to the neurons of the input layer	Linear mapping $y = f(x)$
Activation function applied to the neurons of the hidden and output layer	Sigmoid $y = f(x) = 1/\left(1+e^{-x}\right)$
Activation function applied to the neurons of the output layer	Sigmoid $y = f(x) = 1/\left(1+e^{-x}\right)$

4 Hardware Implementation of the Network

When designing the hardware, multiple aspects influencing the design have to be taken into account [8], [9]. These aspects are related to the structure of the artificial neural network, data representation, the set of operations, which have to be supported by the hardware [10], the choice and the method of implementation of the activation function, the method of storing and updating the data in the network and the algorithm used when training/testing the network [11].

Of the above aspects, the performance of the neurohardware is affected especially by the method of implementing the set of operations used in the learning phase of the network and the activation function. The implementation per se is affected also by the representation of the operands [12].

4.1 Floating Point Representation

Since the task described in section 3 deals with real numbers, it is natural to use floating point representation for the operands. In digital computers, real numbers are represented in accordance with standard IEEE-754, which defines three formats: single precision format (32-bit), double precision format (64-bit) and extended double format (80-bit).

The standard implementation of artificial neural networks uses the 32-bit floating point representation of data – the float data type. The advantage of the representation pursuant to standard IEEE-754 is the wide range of represented real numbers. The disadvantage is the inaccuracy of the real number representation. It's important to note that precision is about how exactly we can specify it (i.e. machine precision) an accuracy is about how close a value is to what it is meant to be; moreover, it is influenced by the applied rounding technique. A further disadvantage of the representation of real numbers pursuant to standard IEEE-754 is the time cost of implementation of the arithmetic operations.

4.2 Fixed Point Representation

Fixed point representation is used to represent real numbers in the way known from the decimal number system,±integerPart.fractionalPart.

When using a binary number system, we let IP be the number of bits used to represent the integer part, including the sign bit. Let FP be the number of bits assigned to represent the fractional part of the number. Then, the total number of bits used to represent the real number may be expressed as $W = IP + FP$.The range of numbersequals to – $[-2^{IP-1}, 2^{IP-1} - 2^{-FP}]$, while machine precision ε equals to 2^{-FP} ($ulp = 2^{-FP}$).

There are numerous forms of notation applicable to the fixed point number representation. In this paper, we will use the Qw.i notation. In this notation, w represents the word length in bits, i is the number of bits used to represent the integer value. For example, Q32.9 describes a number with 9 integer bit and 23 fractional bits stored as a 32-bit 2's complement integer.

The disadvantage of the fixed point representation lies in the significantly lower range of numbers, which may be represented using this format, in comparison with the floating point representation, when using the same number of bits for representation. The advantage of the fixed point representation is that we may use integer arithmetic to implement operations on real numbers, which positively influences the requirements on hardware resources.

An interesting data representation category is the residue number system.

4.3 Residue Number System

Data representations described above employ a linear and positional number system, where the value of each symbol is influenced by the radix of the number system and the position of the symbol in the symbol series. Another group of numerical value representations are non-positional number systems. A significant non-positional number system is the residue number system.

If q and r are the quotient and remainder, respectively, of the integer division of a by m, that is, $a = q.m + b$, then, by definition, we have a ≡ b (mod m). The number b is said to be the residue of a with respect to m, and we shall usually denote this by $r = |a|_m$. The number m is a modulus (aka base). The set of m smallest values, $\{0, 1, 2, \dots, m - 1\}$, that the residue may assume is called the set of least positive residues modulo m.

For example, numbers of the set $\{1, 3, 5, 7, \dots, 2i + 1\}$ belong to the residue class mod 2, while for an arbitrary couple of numbers (a, b) of this set, the following applies: a ≡ b (mod 2).

Let $\{m_1, m_2, m_3, \dots, m_N\}$ be a set of N pairwise relatively prime moduli. Let their product be M, i.e. $M = \prod_{i=1}^{N} m_i$. Then every number $X < M$ has a unique representation in the residue number system, which is the set of residues $\{|X|_{m_i} : 1 \leq i \leq N\}$. The number M is called the period (aka dynamic range) of the RNS, because the number of numbers that can be represented is M. For unsigned numbers, that range is [0, M-1]. For example, the decimal number X, represented in the conventionally weighted number system as $X = 23$ may be expressed as $\langle 1, 2, 3 \rangle_{2,3,5}$ in a system with a period of $M = 30$ ($m_1 = 2, m_2 = 3, m_3 = 5$), where 1 is the result of the operation *X mod 2*, 2 is the result of the operation *X mod 3*, while 3 is the result of the operation *X mod 5*. Representations in a system in which the moduli are not pairwise relatively prime will be not be unique: two or more numbers will have the same representation (Tab. 2)

Number conversion to RNS and from RNS, the standard arithmetic operations of addition/subtraction and multiplication are easily implemented [13], depending on the choice of the moduli, but division is much more difficult.

In the error backpropagation algorithm, the operation of division is used only when evaluating the logical activation function. For the implementation of the activation function we used an approximation, eliminating the need of performing a division.

Table 2
The result of comparing in pairs with the final result

N	***pairwise relatively prime moduli***		***relatively non-prime moduli***	
	$m_1 = 2$	$m_2 = 3$	$m_1 = 2$	$m_2 = 4$
0	0	0	0	0
1	1	1	1	1
4	0	1	0	0
5	1	2	1	1

5 Neural Network Blockset Design

In this work, we decided to implement the artificial neural network on an FPGA chip. The following chapter contains the list of related works (partially in chronological order), aimed at the implementation of artificial neural networks in hardware.

5.1 Related Works

In [14], the authors researched whether FPGA chips were appropriate for speeding-up floating point calculations. They came to the following conclusion: „*...if we can achieve comparable performance from a pure floating-point application, it is a good indication that applications which require a few floating point operations intermixed with fixed-point computations can now be considered as implementation targets for reconfigurable computing. Furthermore, these results indicate that if device density and speed continue to increase, reconfigurable computing platforms may soon be able to offer a significant speedup to pure floating-point applications*“.

Iwata, et al. [15] implemented a BP algorithm using a 24 bit floating point number representation.

The paper written by Holt & Hwang suggests, that an 8 and 16-bit number representation is sufficient for the implementation of BP in hardware [16].

For MLP networks using the BP algorithm, Holt and Baker [17] pointed out that in fixed point number representations, 16 bits are the least to maintain functionality of the network, on condition of normalising the input data to the interval [0,1] and using a sigmoid activation function.

Hammerstrom [18] proposed a neuro-architecture based on an 8 to 16-bit fixed point representation.

Aibe, et al. [19]implemented a probabilistic neural network (PNN). They used floating point representation of operands.

The ASIC implementation of an MLP network using floating point representation for weights and biases is the work of Ayela, et al. [20].

Moussa. et al. demonstrated implementations of MLP on FPGAs using fixed and floating point representations [21].

Wang, et al. [22] proposed a re-configurable architecture for the VLSI implementation of the BP algorithm based on systolic fields.

The authors of [23] focused on the efficient implementation of multiplication in a Maxout network. They trained a set of state-of-the-art neural networks (Maxout networks) on three benchmark datasets: MNIST, CIFAR-10 and SVHN. During the training, they used the following number representations: Goodfellow, et al. Format [24] (32 bits for propagations, 32 bits for parameter updates), single precision floating point (32 bits for propagations, 32 bits for parameter updates), half precision-floating point (16 bits for propagations, 16 bits for parameter updates), Fixed point (20 bits for propagations, 20 bits for parameter updates) and dynamic fixed point (10 bits for propagations, 12 bits for parameter updates). For each of those datasets and for each of those formats, they assess the impact of the precision of the multiplications on the final error after training. The authors of this work came to the conclusion that it is possible use a lower precision number representation not only during the life phase of the network, but also during the training phase. For example, their results achieved on the Startix V Altera FPGA chip suggest that a 10 bit number representation for propagation and a 12-bit representation for the parameter updates of the Maxout network are sufficient.

Gupta, et al [25] studied the effect of limited precision on neural network training and proposed a Xilinx Kintex325T FPGA based hardware accelerator. Their results showed that deep networks could be trained using only 16-bit wide fixed-point number representation with stochastic rounding.

Park & Sung, et al. [26] developed an FPGA based fixed-point deep neural network (DNN) system using only on-chip memory. They used Xilinx XC7Z045 for the implementation. They tested the solution to recognise MNIST handwritten digits. Due to the memory limitations, they used fixed pointrepresentation for data, 3 bits for the input and hidden layers, 8 bits for the output layer, more sensitive to quantisation.

In [27], the authors proposed a roofline-model-based method for FPGA acceleration of convolutional neural networks. In this method they first optimize CNN’s computation and memory access. For data representation, they used 32-bit floats. They implemented the design using the Vivado HLS (v2013.4) environment on a Xilinx VC707 board.

Nakahara & Sasao [28] proposed a hardware implementation of a deep convolutional neural network (DCNN), based on a residue number system. Since the 2D convolutional operation performs massive multiply-accumulation, they propose to use nested RNS (NRNS), which recursively decompose the RNS. In the DCNN using the NRNS, a 48-bit multiply-accumulation unit is decomposed into 4-bit ones realized by look-up tables of the FPGA. The DCNN using the NRNS was implemented on a Xilinx Virtex VC707 evaluation board.

6 Our Proposal

The FPGA (Field Programmable Gate Array) is a kind of logical integrated circuits. It is a programmable gate array used to design digital systems [29]. FPGAs have a wide variety of uses [30]. The main idea is the use of programmable logic elements to perform simple logical functions. These elements are called look-up tables (LUTs) – they may perform arbitrary logical functions with a specific number of inputs and a single output.

Currently, there are many FPGA vendors. Xilinx, Altera, Actel and Atmel belong to the most popular. The FPGA structure depends on the vendor; however, the idea behind the FPGA remains the same – to use a logical block array based on look-up tables and registers. In this work, we used the Xilinx Kintex KC705 development board as an implementation platform.

In the sections below, we describe the implementation of a MLP network with error backpropagation in a Kintex-7 XC7K325T-2FFG900C FPGA chip [31], utilising a Xilinx Kintex KC705 [32] board.

The structure of the proposed neuro-accelerator is depicted in figure 3. In addition to the proposed (*ffnn*) IP module, the proposed hardware contains supporting components necessary for the following: to control the input and output of the IP module (microblaze_0 : MicroBlaze soft processor), to connect the *ffnn* module with the soft processor (microblaze_0_axi_periph : AXI Interconnect), to control interrupts (microblaze_0_axi_intc : AXI Interrupt Controller), to access memory (axi_dma_0 : AXI Direct Memory Access), module to control access to DDR3 SDRAM memory (mig_7series_0 : Memory Interface Generator), to measure the time required to perform the operations both on the soft processor and on the proposed module (axi_timer_0 : AXI Timer), to provide console access to the system (axi_uartlite_0 : AXI Uartlite), a debugging module (mdm_1 : MicroBlaze Debug Module) and a local MicroBlaze memory (microblaze_0_local_memory).

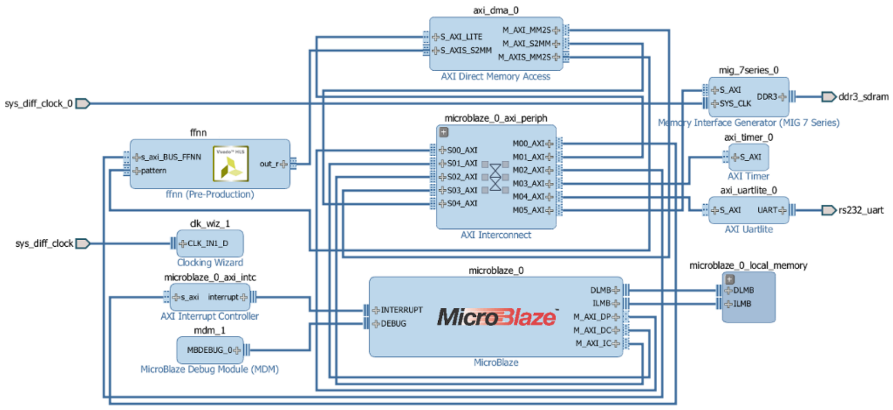

Figure 3
The NeuroAccelerator

6.1 Implementation using Floating Point Representation

Decreasing / increasing the number of bits used for number representation decreases / increases accuracy of the calculations [33]. The minimum bit count necessary to represent the numerical information – while maintaining the necessary level of accuracy and the meeting requirement of convergence of the artificial neural network to the solution – is significantly influenced by the type of the artificial neural network and its algorithm.

In this work, we used an MLP network with error backpropagation. We performed two experiments with the aim to find out the impact of using float and double data types on the overall performance of the artificial neural network.

For the purpose of implementing error backpropagation, we used the "Vivado(TM) HLS – High-Level Synthesis from C, C++ and System C Version 2016.3" tool. The source code of the hardware structure was created in C++. In the first experiment, we used the float (32-bit) data type. A summary of the results of the experiment is available in Table 3.

When using the float data type, due to the memory requirements of the algorithm and the resources of the FPGA chip available for the implementation of this IP, we could use at most 300 neurons in the hidden layer. The remaining hardware resources were used for the implementation of the support components of the chip. The misclassification rate for a network with this configuration, using the test set of 10,000 patterns was 3.76%.

Table 3
Hardware utilization of FFNN for data type Float

Timing [ns]	*Hidden Layer [neurons]*	*BRAM_18K*	*DSP48E*	*FF*	*LUT*
8.61 (116MHz)	200	525 (58%)	75 (8%)	8596 (2%)	88011 (43%)
	250	533 (59%)	75 (8%)	8604 (2%)	107227 (52%)
	300	533 (59%)	75 (8%)	8627 (2%)	126487 (62%)
	350	1045 (117%)	75 (8%)	8638 (2%)	145703 (71%)

In our second experiment we tried to find out: the impact of using the double data type on the network configuration; the performance of the hardware implementation; and the achievable misclassification rate. A summary of the results of the experiment is available in Table 4.

Table 4
Hardware utilization of FFNN for data type Double

Timing [ns]	*Hidden Layer [neurons]*	*BRAM_18K*	*DSP48E*	*FF*	*LUT*
8.68 (115MHz)	100	528 (59%)	158 (18%)	12583 (3%)	91477 (44%)
	150	536 (60%)	158 (18%)	12613 (3%)	129928 (63%)
	200	1048 (117%)	158 (18%)	12624 (3%)	168347 (82%)

In this case, the maximum number of neurons in the hidden layer was set to 150 neurons. The remaining hardware resources were used for the implementation of the support components of the chip. The misclassification rate for a network with this configuration amounted to 4.51%.

6.2 Implementation using Fixedpoint Representation

In this phase of the design we tried to find out what was the lowest bit count required to sufficiently train an MLP network, using floating point decimal data representation.

A summary of the results is available in the chart above (Fig. 4) – from this it is evident that up to 17 bits, the misclassification rate stays below 5%. Beyond this level, the misclassification rate grew fast. Another finding of the experiment was that when using 300 hidden neurons, the integer part of the fixed point representation did not exceed the value of 2. Therefore, in this implementation, we started out from the data representation using the Q17.2 two's complement form.

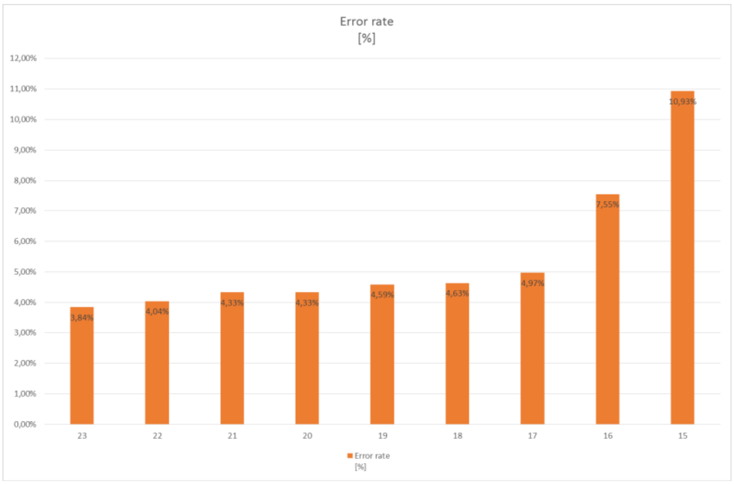

Figure 4

Misclassification rate

The overall hardware implementation costs are available in Table 5.

Table 5

Hardware utilization of FFNN for data type Q17.2

Timing [ns]	*Hidden Layer [neurons]*	*Name*	*BRAM_18K*	*DSP48E*	*FF*	*LUT*
8.58 (116MHz)	300	Utilization	555 (62%)	48 (5%)	5926 (3%)	11749 (5%)

The misclassification rate for a network with this configuration, using the test set of 10,000 patterns was 4.97%.

6.3 Implementation using the Residue Number System

Based on the results of the fixed pointdata representation experiment, we found that the lowest number of bits, which may be used to train the network with a misclassification rate below 5% was 17 bits. We used this information when implementing the *ffnn* IP module, based on calculations using the residue number system. As it is known, the operations of addition, subtraction and multiplication can be easily implemented in residue number systems. Due to the character of the residue number system and the aforementioned operations, these operations do not require carrybit propagation, therefore their execution is faster than the execution of these operations using fixed point number representation. Summaries of the time characteristics of selected mathematical operations using the residue number system and in a positional binary system were presented in [34].

However, division and comparison are more complex in this system. In the error backpropagation algorithm, the operation of division is used only to calculate the value of the logistic function. In the next chapter, we describe the solution to eliminate division from the error backpropagation.

6.3.1 Activation Function Implementation

It is important to note that the performance and precision of networks depend on the efficient implementation of the activation function on FPGA chips [35]. In MLP networks, the most common activation function is the sigmoid function. Direct implementation of sigmoid activation function on FPGA is difficult due to its division and exponential function. The operation of division and the implementation of an exponential function are time-consuming; moreover, they require also significant hardware resources [36].

According to [37], to decrease the requirements hardware resources and the time necessary to evaluate the function, the following may be used: uniform lookup table methods (LUT), linear approximation methods, piecewise linear approximation method (PWL), piecewise linear approximation of a nonlinear function (PLAN) approximation, A-law approximation, Allipi and Storti-Gajani approximation, piecewise second-order approximation and lookup table method with linear interpolation method. Compared to the approximation methods, solutions using LUT provide high speed, though at a cost of higher memory requirements.

Approximation methods have lower memory requirements. When combined with fixed point number representation, the speed differences are not as significant in comparison with the LUT methods [38].

In our proposal, we used the PLAN approximation of the sigmoid activation function, as published in [38]. To increase speed, we used only first-order functions for the approximation. The approximation of the function output was based on the evaluation of a linear function, used for the approximation of the sigmoid in a defined interval of input values (Table 6). We have set the coefficients with a precision of $ulp = 2^{-12}$.

Table 6
PLAN approximation of sigmoid activation function

Interval	*Function Form*	*Absolute approximation error*
$[\mathbf{2};\mathbf{3}[$	$y = f(x) = 0.07110596 * x + 0.74377441$	0.0051893
$[\mathbf{1};\mathbf{2}[$	$y = f(x) = 0.14950562 * x + 0.58935547$	0.0078025
$]-\mathbf{1};\mathbf{1}[$	$y = f(x) = 0.23828125 * x + 0.50000000$	0.0072125
$[-\mathbf{2};-\mathbf{1}]$	$y = f(x) = 0.14950562 * x + 0.41058350$	0.0078635
$[-\mathbf{3};-\mathbf{2}[$	$y = f(x) = 0.07110596 * x + 0.25610352$	0.0053031

6.3.2 Choice of the Residue Number System Parameters

The error backpropagation algorithm uses addition and multiplication operations. In the binary positional system, the result of adding two *n* bit numbers is a sum having at most *n+1* bits, the result of their multiplication is the product having *2n* bits. In a residue number system, carrybit propagation does not occur. The range and the precision of the calculation are functions of the period*M*. In the previous section we showed that when using 17 bits (1b sign + 16b modulus), the misclassification rate remains under 5%. The result of the multiplication of two 17-bit signed numbers is a 34-bit number (1b sign + 33b modulus).

To cover the required range of 34-bit signed numbers, we used the *R* residue number system with moduli $\{11,23,31,73,89,127,128\}$. Note that these moduli have to be pairwise relatively prime. The system period is $M = 828340264576$,, i.e. $M > 2^{34}$.

When transforming a binary number in fixed pointnumber representation Q17.2 to the residue number system, we used the properties of addition and multiplication in the residue number system. The following example shows the procedure of transforming a binary number into the R system with the set of moduli $\{5,7\}$.

Example: Consider the system *R* with the set of moduli $m_1 = 5, m_2 = 7$. M = 35. Let $X = (26)_{10}$. Then, by using binary decomposition: $X = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1$ Moreover: $X =_R \langle 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 \rangle = \langle 1 \times 2^4 \rangle + \langle 1 \times 2^3 \rangle + \langle 1 \times 2^1 \rangle = \langle 1 \rangle \times \langle 2^4 \rangle + \langle 1 \rangle \times \langle 2^3 \rangle + \langle 1 \rangle \times \langle 2^1 \rangle$ Thus $X =_R \langle |1|_5 \times |2^4|_5 + |1|_5 \times |2^3|_5 + |1|_5 \times |2^1|_5, |1|_7 \times |2^4|_7 + |1|_7 \times |2^3|_7 + |1|_7 \times |2^1|_7 \rangle = \langle 1 \times 1 + 1 \times 3 + 1 \times 2, 1 \times 2 + 1 \times 1 + 1 \times 2 \rangle = \langle |6|_5, 5 \rangle = \langle 1,5 \rangle$.

6.3.3 Implementation of BP Algoritm in Residue Number System

Residue number systems do not have metrics. Therefore, the comparison of two numbers represented in such a system is time-consuming. When comparing numbers, we may first transform the residue number system representation to a positional system, perform the comparison or – under certain circumstances – the comparison may be performed also by subtracting the two numbers and determining the sign of the operation [39]. Further algorithms are available in [13].

Table 7

Hardware utilization of FFNN for RNS *R* with moduli set {11,23,31,73,89,127,128}

Timing [ns]	*Hidden Layer [neurons]*	*Name*	*BRAM_18K*	*DSP48E*	*FF*	*LUT*
8.18 (122MHz)	300	Utilization	500 (56%)	336 (40%)	38519 (9,4%)	145337 (71%)

We chose to use the reverse transformation to a binary number system. The overall hardware implementation costs are available in Table 7.

With this solution, the misclassification rate amounted to 4.01%.

6.4 Comparison of the Solutions

We compared our solution with an implementation running on a desktop computer with an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz processor. In this paper, when determining the overall performance of the proposed architecture, we will start out from the overall cost of implementing one forward and one backward phase of neural network training per epoch.The results are available in Table 8.

If the sole criterion of performance is the speed of data training, the winner is the Desktop CPU. However, if we consider also energy efficiency, the winner is the MLP network implementation running on an FPGA chip. If we require higher overall recognition precision, the binary positional system using the float data type is a good candidate for representing data in the network, otherwise the use of a fixed pointnumber representation is advisable.

The residue number system falls back behind the solution using fixed pointnumber representation in terms of speed; however, it's misclassification rate is lower.

Table 8

Comparison of the solutions with an Intel i7-4790 CPU

	Clock frequency [MHz]	*Power Consumption [W]*	*Timing (1 epoch, 1 iter) [s]*	*Overall Error-rate (requested < 5%)*
Desktop CPU (float)	3.60GHz	84 (TDP)	28	3,76%
FPGA – float	116MHz	3.569	37	3,76%
FPGA – double	115MHz	7.298	52	4,51%
FPGA – fixed	116MHz	3.123	30	4,97%
FPGA – RNS	122MHz	4.560	36	4.01%

Conclusions

This paper deals with the implementation possibilities of multilayer neural networks aimed at the recognition of handwritten digits. The applied neural network belongs to the artificial neural networks with supervised learning. We used the error backpropagation algorithm as the training algorithm. The training and testing set was taken from the MNIST database. As the target platform for the hardware implementation, we used a Kintex-7 XC7K325T-2FFG900C FPGA chip on a Xilinx Kintex KC705 board. We researched the impact of the choice of number representation (floating point vs. fixed point decimal representation), the impact of the number of bits used to represent the numbers in fixed point notation and the choice of the number system (conventionally weighted positional number system vs. residue number system) on both the utilisation of hardware resources and the neural network performance (training speed and misclassification rate). A summary of our findings is available in Table 8.

Acknowledgement

This work was supported by KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 077TUKE-4/2015 „Promoting the interconnection of Computer and Software Engineering using the KPIkit“. This support is very gratefully acknowledged.

References

[1] M. Perron and L. Cooper, "The Ni1000: High Speed Parallel VLSI for Implementing Multilayer Perceptrons," 1995.

[2] K. Berkolds, "Image Recognition with Hardware Neural Networks," in *Engineering for Rural Development*, Jelgava, Latvia, 2016.

[3] S. Esser, A. Alexander, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico†, S. Carpin†, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. Kusnitz, T. Wong, W. Risk and McQui, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, 2013.

[4] J. Ban, M. Féder, M. Oravec and J. Pavlovičová, "Non-Conventional Approaches to Feature Extraction for Face Recognition," *Acta Polytechnica Hungarica,* vol. 8, no. 4, 2011.

[5] R. Lovassy, L. T. Kóczy and L. Gál, "Function Approximation Performance of Fuzzy Neural Networks," *Acta Polytechnica Hungarica,* vol. 7, no. 4, 2010.

[6] G. Györök, "Reconfigurable Control in Robust Systems by FPAA," in *Intelligent Systems and Informatics, 2008: Proceedings of SISY 2008, 6th International Symposium.*, Subotica, Serbia, 2008.

[7] D. Rumelhart, G. Hinton and R. Wiliams, "Learning internal representations by error propagation," in *Parallel distributed processing: explorations in the microstructure of cognition*, vol. 1, D. Rumelhart and J. McClelland, Eds., MA, USA, MIT Press Cambridge, MA, USA, 1986, pp. 318-362.

[8] Š. Hudák, Š. Korečko and S. Šimoňák, "Reachability analysis of time-critical systems," in *Petri Nets Applications*, InTech, 2010, pp. 253-280.

[9] M. Novák and M. Biňas, "An architecture overview of the smart-home

system based on OSGi," in *SCYR 2011*, Herľany, 2011.

[10] L. Vokorokos, B. Madoš, A. Baláž and N. Ádám, "Architecture of Multi-Core Computer with Data Driven Computation Model," *Acta Electrotechnica et Informatica,* vol. 10, no. 4, pp. 20-23, 2010.

[11] L. Vokorokos, N. Ádám and A. Baláž, "Training Set Parallelism In Pahra Architecture," *Acta Electrotechnica et Informatica,* vol. 7, no. 3, pp. 1-6, 2007.

[12] F. Silváši and S. Šimoňák, "Architecture Dependent Program Optimizations," in *Electrical Engineering and Informatics 4: Proceeding of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice*, Košice, 2013.

[13] A. Omondi and B. Premkumar, Residue Number Systems : Theory and Implementation, Advances in Computer Science and Engineering: Texts – Vol. 2 ed., vol. 2, London: Imperial College Press, 2007.

[14] W. Ligon III, S. McMillan, G. Monn, K. Schoonover, F. Stivers and K. Underwood, "A Re-evaluation of the Practicality of Floating Point Operations on FPGAs," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 1998.

[15] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato and N. Suzumura, "An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors," in *International 1989 Joint Conference on Neural Networks*, 1989.

[16] J. L. Holt and J. N. Hwang, "Finite-precision error analysis of neural network hardware implementations," *IEEE Transactions on Computers,* vol. 42, no. 3, p. 280–290, 1993.

[17] J. L. Holt and T. Baker, "Backpropagation simulations using limited precision calculations," in *Proc. of International Joint Conference on Neural Networks (IJCNN-91),* Seattle, WA, USA, 1991.

[18] D. Hammerstom, "A highly parallel digital architecture for neural network simulation," in *VLSI for Artificial Intelligence and Neural Networks*, J. Delgado-Frias and W. Moore, Eds., Plenum Press, 1991.

[19] N. Aibe, M. Yasunaga, I. Yoshihara and J. H. Kim, "A probabilistic neural network hardware system using a learning-parameter parallel architecture," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '02)*, Honolulu, Hawaii, 2002.

[20] J. L. Ayala, A. G. Lomeña, M. López-Vallejo and A. Fernández, "Design of a

pipelined hardware architecture for real-time neural network computations," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems (MWSCAS '02)*, Tulsa, Okla, USA, 2002.

[21] M. Moussa, S. Areibi and K. Nichols, "On the arithmetic precision for implementing back-propagation networks on FPGA: a case study," in *FPGA Implementations of Neural Networks*, Berlin, Germany, Springer, 2006, pp. 37-61.

[22] Q. Wang, A. Li, Z. Li and Y. Wan, "A Design and Implementation of Reconfigurable Architecture for Neural Networks Based on Systolic Arrays," in *Advances in Neural Networks*, Lecture Notes in Computer Science ed., vol. Vol 3973, Berlin, Heidelberg, Springer, 2006.

[23] M. Courbariaux, J.-P. David and B. Y., "Training Deep Neural Networks With Low Precision Multiplications," *arXiv e-prints*, vol. 1412.7024, 2014.

[24] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, "Maxout networks," Universite de Montreal, 2013.

[25] S. Gupta, A. Agrawal, K. Gopalakrishnan and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. of the 32nd International Conference on International Conference on Machine Learning*, 2015.

[26] J. Park and W. Sung, "Fpga based implementation of deep neural networks using on-chip memory only," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[27] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proc. of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, California, USA, 2015.

[28] H. Nakahara and T. Sasao, "A Deep Convolutional Neural Network Based on Nested Residue Number System," in *25th International Conference on Field Programmable Logic and Applications (FPL)*, Lausanne, Switzerland, 2015.

[29] L. Vokorokos, B. Madoš, J. Perháč and M. Chovanec, "Architecture of DFC-1 computer with data driven computation model," in *The 6th International Symposium on Applied Machine Intelligence and Informatics*, Herľany, Slovakia, 2008.

[30] O. Kainz, F. Jakab, M. Michalko a P. Feciľak, „Detection of Persons and Height Esatimation in Video Sequence," *International Journal of Engineering Sciences & Research Technology*, zv. 5, %1. vyd.3, 2016.

[31] „7 Series FPGAs Data Sheet: Overview," Xilinx, March 28, 2017.

[32] „KC705 Evaluation Board for the Kintex-7 FPGA - User Guide," Xilinx, July 8, 2016.

[33] L. Kónya and J. Kopják, PIC mikrovezérlők alkalmazástechnikája, PIC programozás C nyelven, Budapest: ChipCAD Elektronikai Disztribúció Kft., 2009.

[34] E. Olsen, „Introduction of the Residue Number Arithmetic Logic Unit With Brief Computational Complexity Analysis," 2015.

[35] L. D. J. Xiaobin, "A mixed Parallel Neural Networks Computing Unit Implemented in FPGA," in *IEEE Intl. Conference Neural Networks & Signal Processing*, China, 2003.

[36] A. Gomperts and A. Ukil, "Development and Implemenation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications," *IEEE Transaction on industrial informatics,* vol. 7, no. 1, 2011.

[37] K. Lakshmi and D. M. Subadra, "A Survey on FPGA based MLP Realization for On-chip Learning," *International Journal of Scientific & Engineering Research,* vol. 4, no. 1, pp. 1-9, 2013.

[38] M. Panicker and C. Babu, "Efficient FPGA Implementation of Sigmoid and Bipolar Sigmoid Activation Functions for Multilayer Perceptrons," *IOSR Journal of Engineering (IOSRJEN),* vol. 2, no. 6, pp. 1352-1356, 2012.

[39] L. Sousa, "Efficient Method for Magnitude Comparison in RNS Based on Two Pairs of Conjugate Moduli," in *18th IEEE Symposium on Computer Arithmetic (ARITH'07),* Washington, DC, USA, 2007.